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

This document is a MIPI Specification. MIPI member companies' rights and obligations apply to this Specification as defined in the MIPI Membership Agreement and MIPI Bylaws.

This document is subject to further editorial and technical development.

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

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 (I^{2}C): CCI supporting I^{2}C.

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

ILR	Interpacket Latency Reduction
JFIF	JPEG File Interchange Format
JPEG	Joint Photographic Expert Group
LE	Line End
LFSR	Linear Feedback Shift Register
LLP	Low Level Protocol
LS	Line Start
LSB	Least Significant Bit
LSS	Least Significant Symbol
LP	Low-Power; identifier for operation mode
LP-RX	Low-Power Receiver (Large-Swing Single Ended)
LP-TX	Low-Power Transmitter (Large-Swing Single Ended)
LRTE	Latency Reduction Transport Efficiency
MSB	Most Significant Bit
MSS	Most Significant Symbol
PDQ	Packet Delimiter Quick (PHY generated and consumed signaling used in LRTE)
PF	Packet Footer
PH	Packet Header
PI	Packet Identifier
PT	Packet Type
PHY	Physical Layer
PPI	PHY Protocol Interface
PRBS	Pseudo-Random Binary Sequence
RGB	Color representation (Red, Green, Blue)
RX	Receiver
SCL	Serial Clock (for CCI)
SDA	Serial Data (for CCI)
SLM	Sleep Mode
SoT	Start of Transmission
TX	Transmitter
ULPS	Ultra Low Power State
VGA	Video Graphics Array
YUV	Color representation (Y for luminance, U & V for chrominance)

3 References

[NXP01] UM10204, *I^2C bus specification and user manual*, Revision 6, NXP Semiconductors N.V., 4 April 2014.

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

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

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

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

[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

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

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

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

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

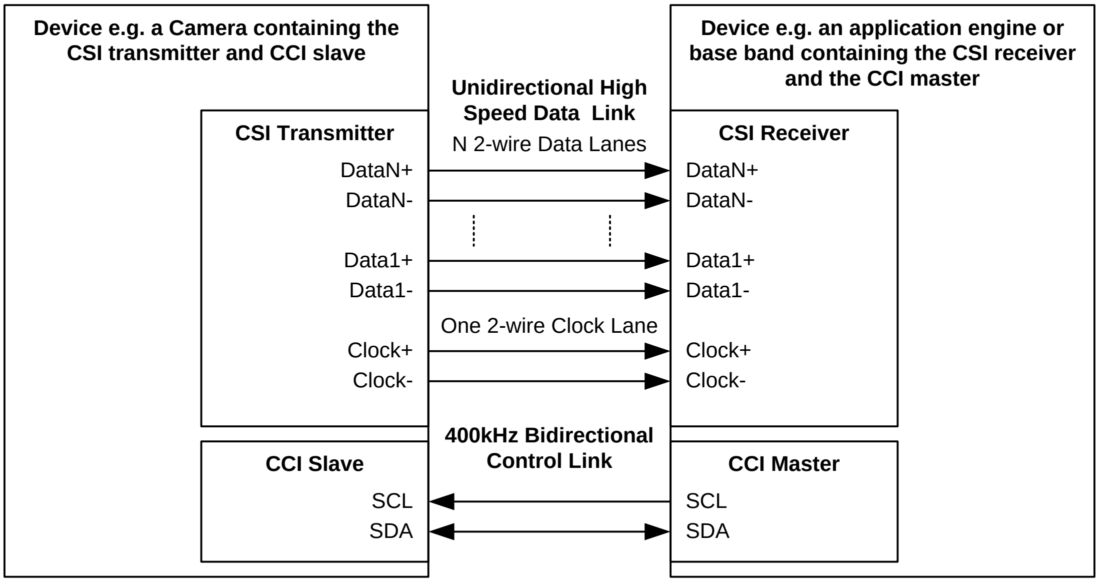
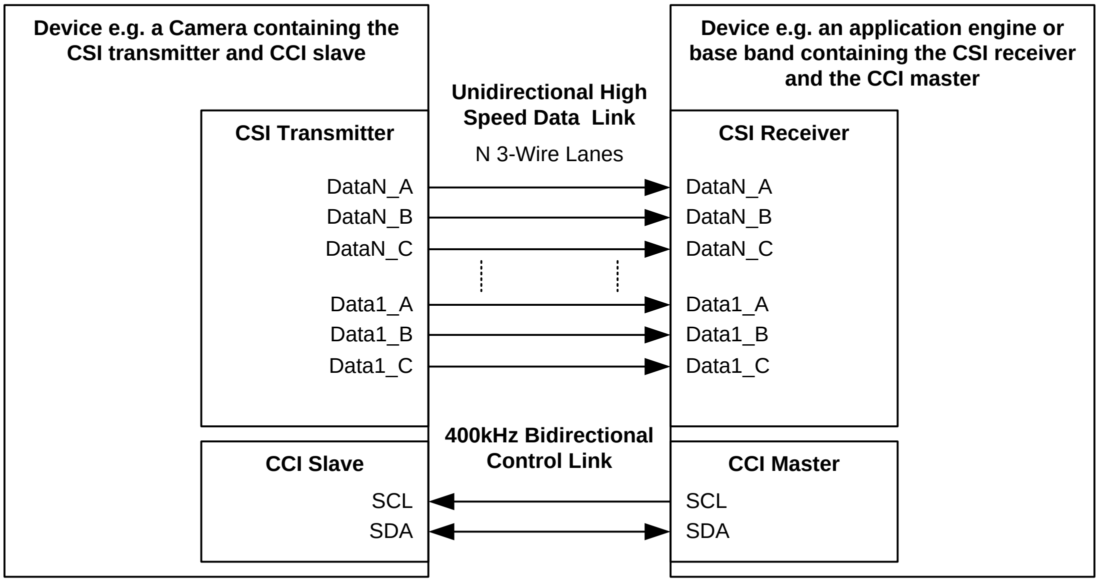

Device e.g. a Camera containing the CSI transmitter and CCI slave
CSI Transmitter
DataN+
DataN-
Data1+
Data1-
Clock+
Clock-
CCI Slave
SCL
SDA
Unidirectional High Speed Data Link
N 2-wire Data Lanes
One 2-wire Clock Lane
400kHz Bidirectional Control Link
Device e.g. an application engine or base band containing the CSI receiver and the CCI master
CSI Receiver
DataN+
DataN-
Data1+
Data1-
Clock+
Clock-
CCI Master
SCL
SDA

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


Device e.g. a Camera containing the CSI transmitter and CCI slave
Device e.g. an application engine or base band containing the CSI receiver and the CCI master
Unidirectional High Speed Data Link
N 3-Wire Lanes
CSI Transmitter
CSI Receiver
DataN_A
DataN_B
DataN_C
Data1_A
Data1_B
Data1_C
DataN_A
DataN_B
DataN_C
Data1_A
Data1_B
Data1_C
400kHz Bidirectional Control Link
CCI Slave
CCI Master
SCL
SDA
SCL
SDA

152 **Figure 2 CSI-2 and CCI Transmitter and Receiver Interface for C-PHY**

5 CSI-2 Layer Definitions

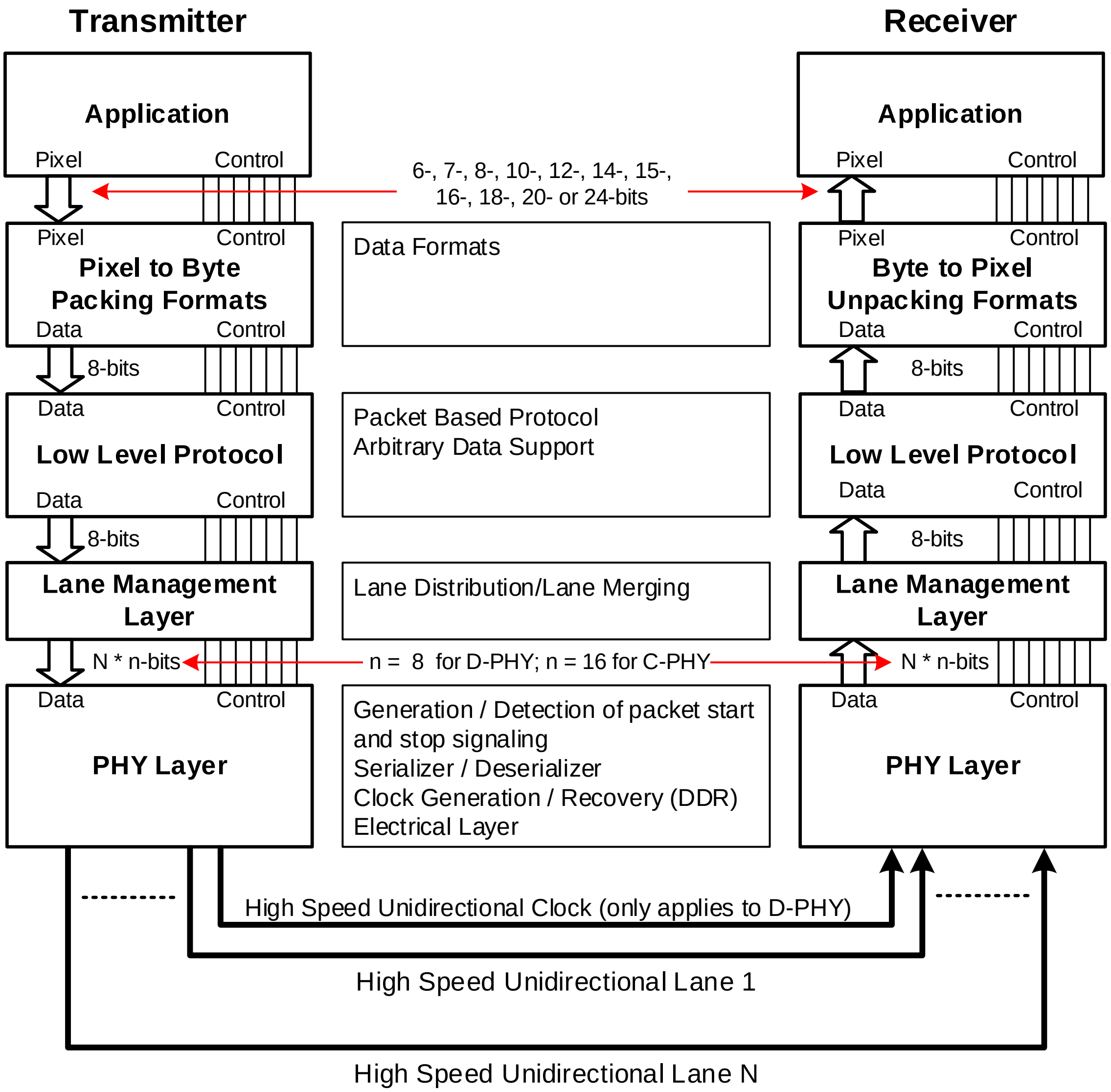

Transmitter
Receiver
Application
Pixel
Control
Application
Pixel
Control
6-, 7-, 8-, 10-, 12-, 14-, 15-, 16-, 18-, 20- or 24-bits
Pixel
Control
Pixel to Byte Packing Formats
Data
Control
Data Formats
Pixel
Control
Byte to Pixel Unpacking Formats
Data
Control
8-bits
8-bits
Data
Control
Low Level Protocol
Data
Control
Packet Based Protocol
Arbitrary Data Support
Data
Control
Low Level Protocol
Data
Control
8-bits
8-bits
Lane Management Layer
Lane Distribution/Lane Merging
Lane Management Layer
N * n-bits
n = 8 for D-PHY; n = 16 for C-PHY
N * n-bits
Data
Control
PHY Layer
Generation / Detection of packet start and stop signaling
Serializer / Deserializer
Clock Generation / Recovery (DDR)
Electrical Layer
Data
Control
PHY Layer
High Speed Unidirectional Clock (only applies to D-PHY)
High Speed Unidirectional Lane 1
High Speed Unidirectional Lane N

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]***.

- **Protocol Layer.** The Protocol layer is composed of several layers, each with distinct responsibilities. The CSI-2 protocol enables multiple data streams using a single interface on the host processor. The Protocol layer specifies how multiple data streams may be tagged and interleaved so each data stream can be properly reconstructed.
  - **Pixel/Byte Packing/Unpacking Layer.** The CSI-2 specification supports image applications with varying pixel formats. In the transmitter this layer packs pixels from the Application layer into bytes before sending the data to the Low Level Protocol layer. In the receiver this layer unpacks bytes from the Low Level Protocol layer into pixels before sending the data to the Application layer. Eight bits per pixel data is transferred unchanged by this layer.
  - **Low Level Protocol.** The Low Level Protocol (LLP) includes the means of establishing bit-level and byte-level synchronization for serial data transferred between SoT (Start of Transmission) and EoT (End of Transmission) events and for passing data to the next layer. The minimum data granularity of the LLP is one byte. The LLP also includes assignment of bit-value interpretation within the byte, i.e. the “Endian” assignment.
  - **Lane Management.** CSI-2 is Lane-scalable for increased performance. The number of data Lanes is not limited by this specification and may be chosen depending on the bandwidth requirements of the application. The transmitting side of the interface distributes (“distributor” function) bytes from the outgoing data stream to one or more Lanes. On the receiving side, the interface collects bytes from the Lanes and merges (“merger” function) them together into a recombined data stream that restores the original stream sequence. For the C-PHY physical layer option, this layer exclusively distributes or collects byte pairs (i.e. 16-bits) to or from the data Lanes. Scrambling on a per-Lane basis is an optional feature, which is specified in detail in ***Section 9.15***.

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

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

The normative sections of the Specification only relate to the external part of the Link, e.g. the data and bit 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

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

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

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

- START or Repeated START condition
- Slave address with read/write bit
- Acknowledge from slave
- Sub address (INDEX) for pointing at a register inside the slave device (not used in Single Read from Current Location)
- Acknowledge signal from slave (not used in Single Read from Current Location)

And then either:

- For a write operation:
  - Data byte from master
  - Acknowledge/negative acknowledge from slave, and
  - 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 signal from slave (not used in Single Read from Current Location)
  - data byte from the slave
  - acknowledge or negative acknowledge from the master, and
  - STOP or Repeated START condition.

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

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

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 question defines what Message type is used.

6.1.2 CCI (I^2C) Read/Write Operations

A **CCI (I^2C)** compatible device shall support the four read operations and two write operations shown in ***Table 1***, as detailed in the following sub-sections:

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***

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

6.1.2.1 CCI (I^2C) Single Read from Random Location

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

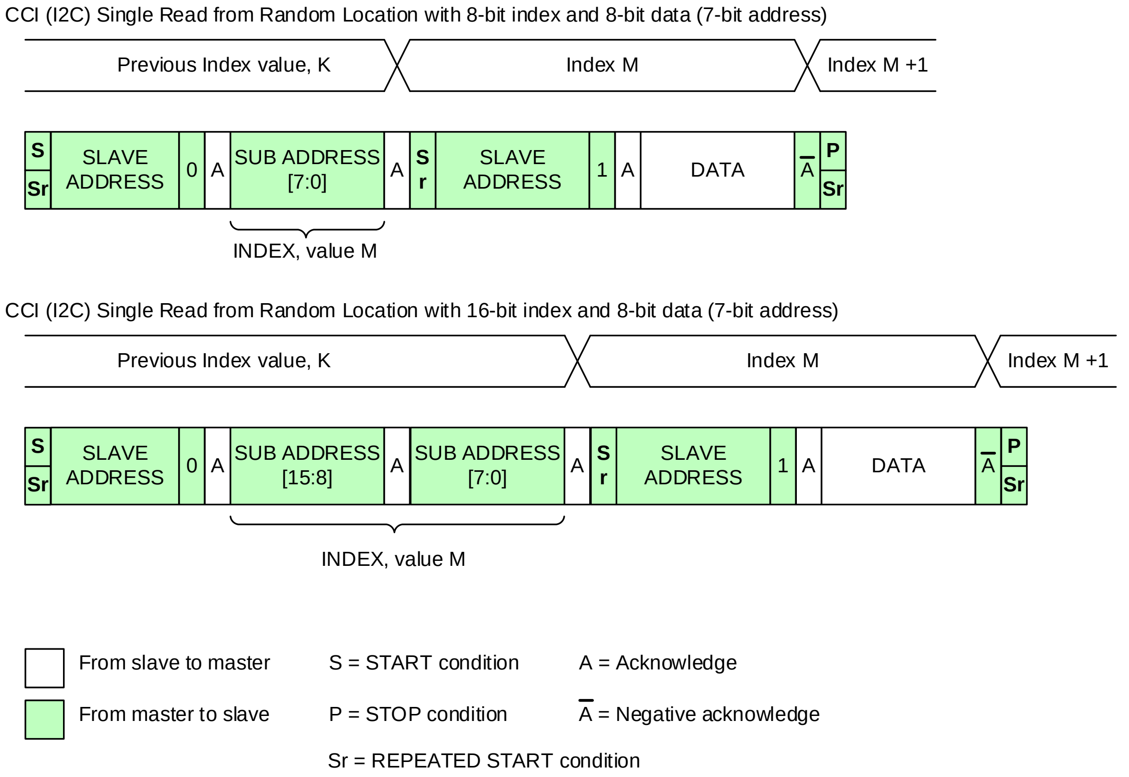


Figure 4 CCI (I^2C) Single Read from Random Location

6.1.2.2 CCI (I^{2}C) Single Read from Current Location

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

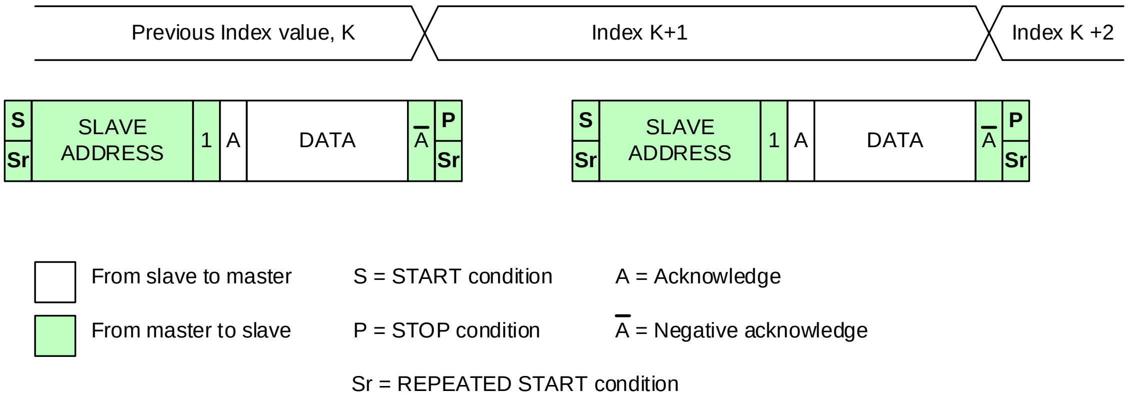


Figure 5 CCI (I^{2}C) Single Read from Current Location

6.1.2.3 CCI (I^2C) Sequential Read Starting from Random Location

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

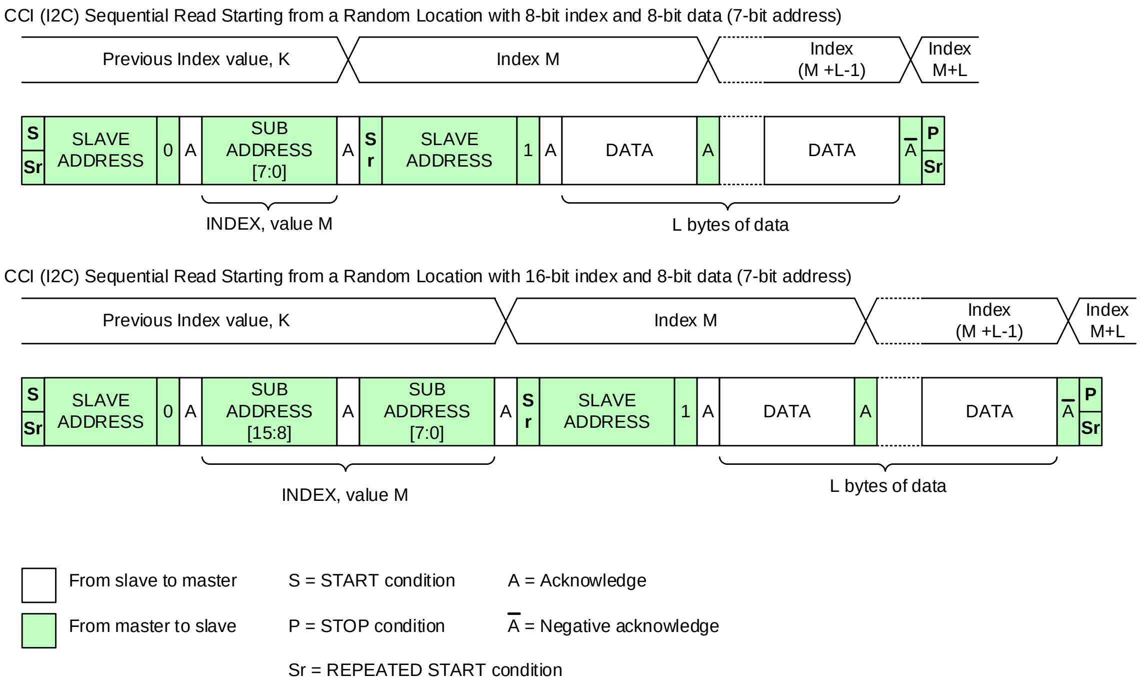


Figure 6 CCI (I^2C) Sequential Read Starting from Random Location

6.1.2.4 CCI (I^{2}C) Sequential Read Starting from Current Location

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

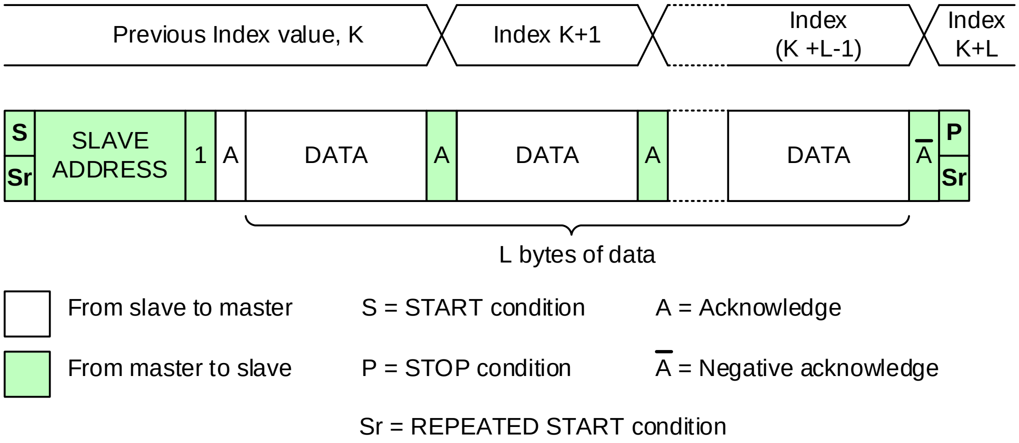


Figure 7 CCI (I^{2}C) Sequential Read Starting from Current Location

6.1.2.5 CCI (I^{2}C) Single Write to Random Location

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

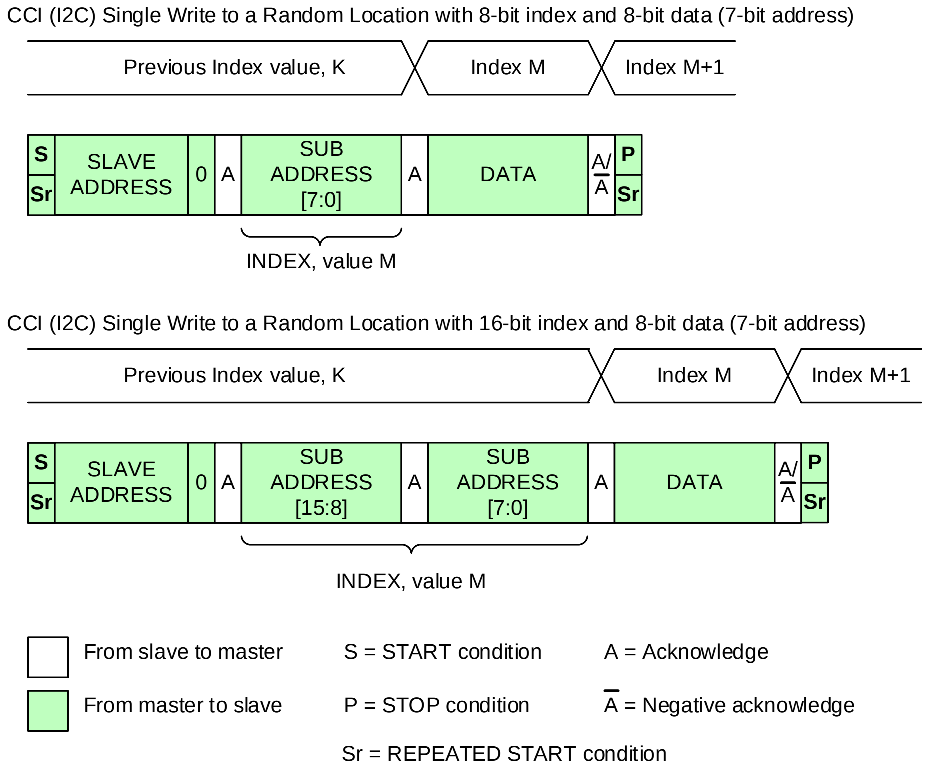


Figure 8 CCI (I^{2}C) Single Write to Random Location

6.1.2.6 CCI (I^{2}C) Sequential Write Starting from Random Location

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

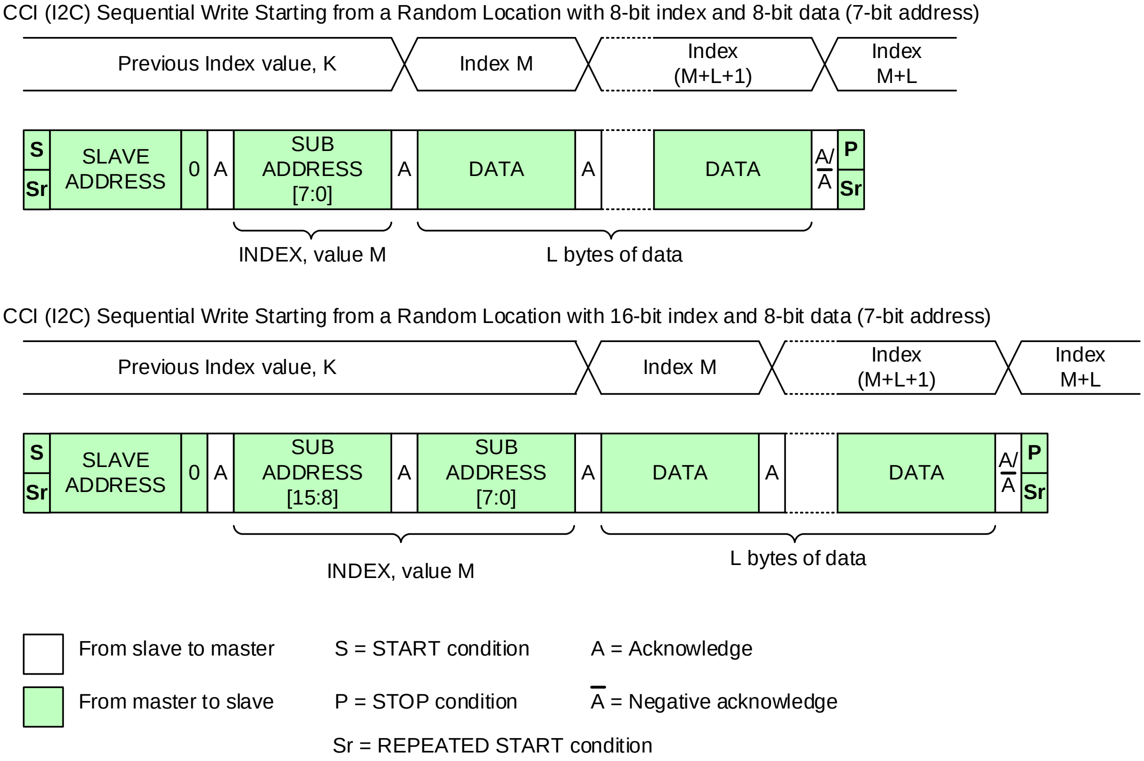


Figure 9 CCI (I^{2}C) Sequential Write Starting from Random Location

6.2 CCI (I3C) Data Transfer Protocol

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

If **CCI (I3C)** is supported, then CCI **(I3C SDR)** shall be supported and **CCI (I3C DDR)** may be supported. The master shall get the slave's Max Read Length (MRL) and Max Write Length (MWL) via transmitting 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

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

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

- START condition
- 7'h7E with write bit
- Acknowledge from slave
- 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.

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

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

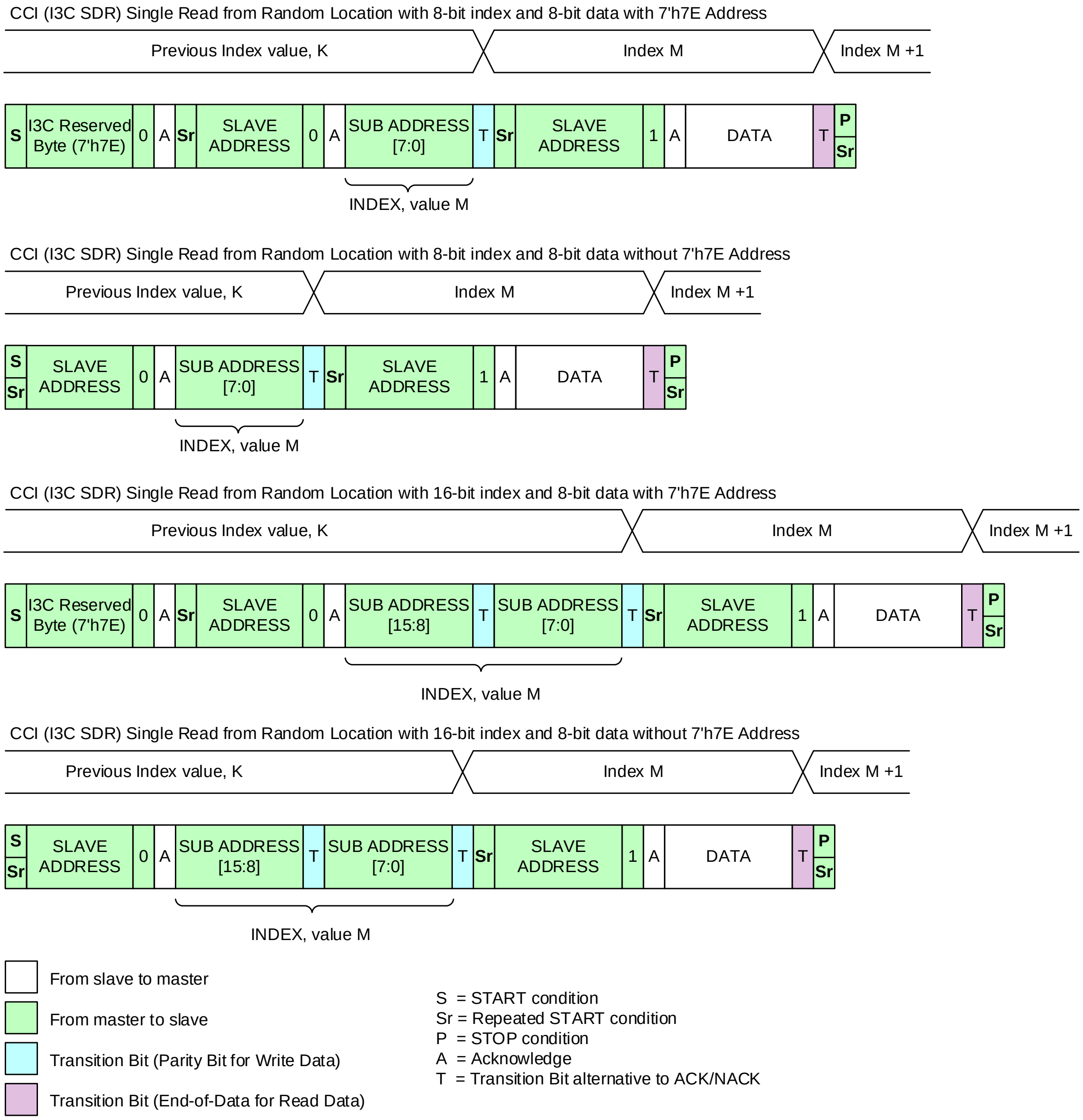

CCI (I3C SDR) Single Read from Random Location with 8-bit index and 8-bit data with 7'h7E Address
Previous Index value, K
Index M
Index M +1
S
I3C Reserved Byte (7'h7E)
0
A
Sr
SLAVE ADDRESS
0
A
SUB ADDRESS [7:0]
T
Sr
SLAVE ADDRESS
1
A
DATA
T
P
Sr
INDEX, value M
CCI (I3C SDR) Single Read from Random Location with 8-bit index and 8-bit data without 7'h7E Address
Previous Index value, K
Index M
Index M +1
S
Sr
SLAVE ADDRESS
0
A
SUB ADDRESS [7:0]
T
Sr
SLAVE ADDRESS
1
A
DATA
T
P
Sr
INDEX, value M
CCI (I3C SDR) Single Read from Random Location with 16-bit index and 8-bit data with 7'h7E Address
Previous Index value, K
Index M
Index M +1
S
I3C Reserved Byte (7'h7E)
0
A
Sr
SLAVE ADDRESS
0
A
SUB ADDRESS [15:8]
T
SUB ADDRESS [7:0]
T
Sr
SLAVE ADDRESS
1
A
DATA
T
P
Sr
INDEX, value M
CCI (I3C SDR) Single Read from Random Location with 16-bit index and 8-bit data without 7'h7E Address
Previous Index value, K
Index M
Index M +1
S
Sr
SLAVE ADDRESS
0
A
SUB ADDRESS [15:8]
T
SUB ADDRESS [7:0]
T
Sr
SLAVE ADDRESS
1
A
DATA
T
P
Sr
INDEX, value M
□ 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

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

6.2.1.2.2 CCI (I3C SDR) Single Read from Current Location

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

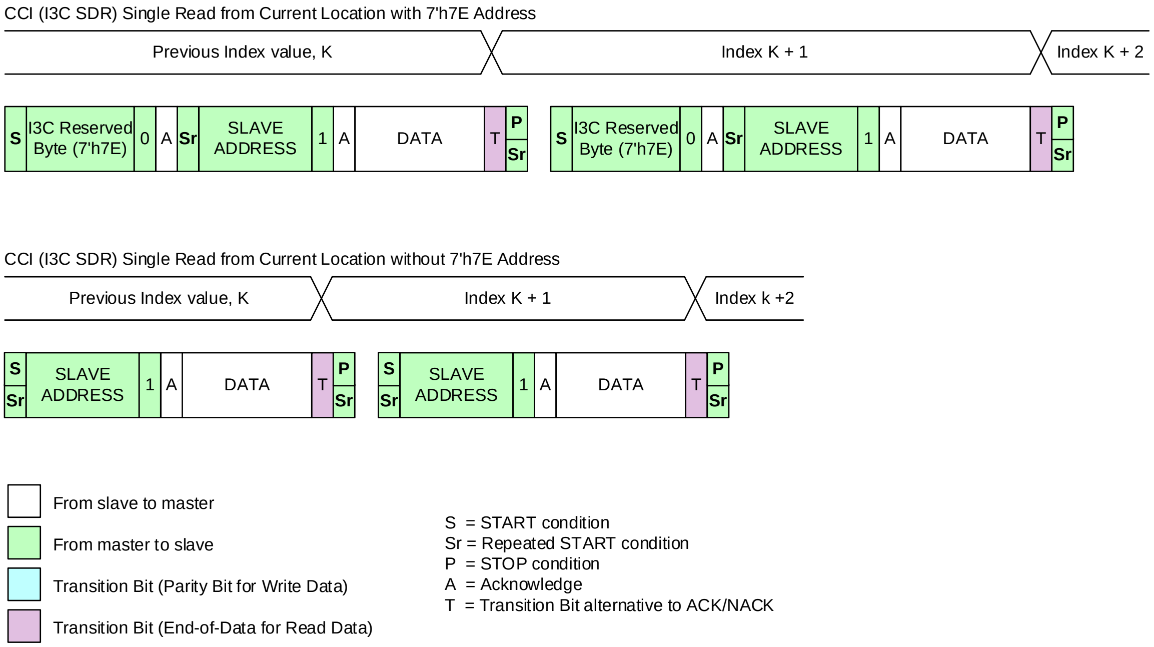

CCI (I3C SDR) Single Read from Current Location with 7'h7E Address
Previous Index value, K
Index K + 1
Index K + 2
S
I3C Reserved Byte (7'h7E)
0
A
Sr
SLAVE ADDRESS
1
A
DATA
T
P
Sr
CCI (I3C SDR) Single Read from Current Location without 7'h7E Address
Previous Index value, K
Index K + 1
Index k +2
S
Sr
SLAVE ADDRESS
1
A
DATA
T
P
Sr
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

Figure 11 CCI (I3C SDR) Single Read from Current Location

6.2.1.2.3 CCI (I3C SDR) Sequential Read from Random Location

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

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).*

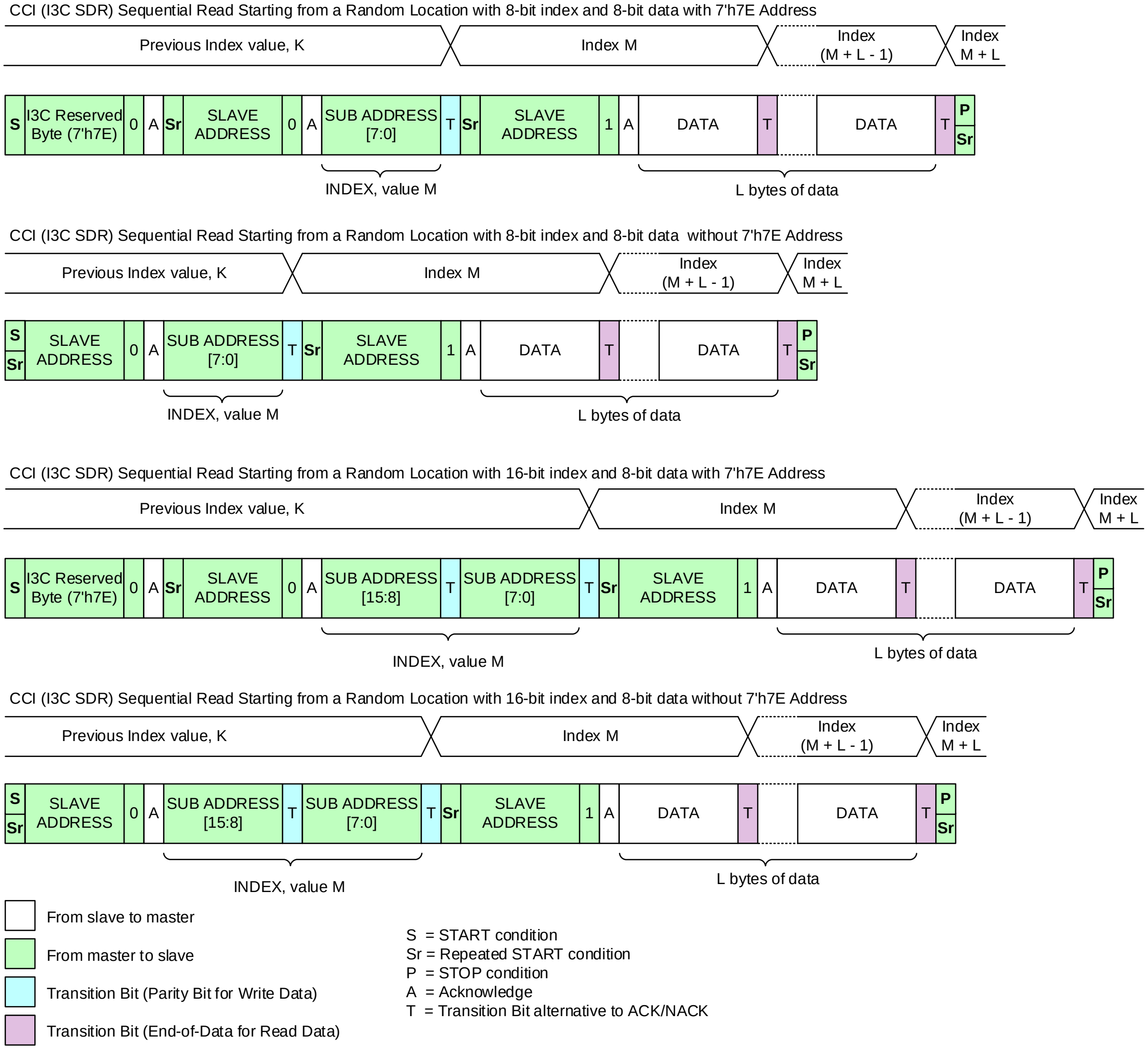


Figure 12 CCI (I3C SDR) Sequential Read Starting from Random Location

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.

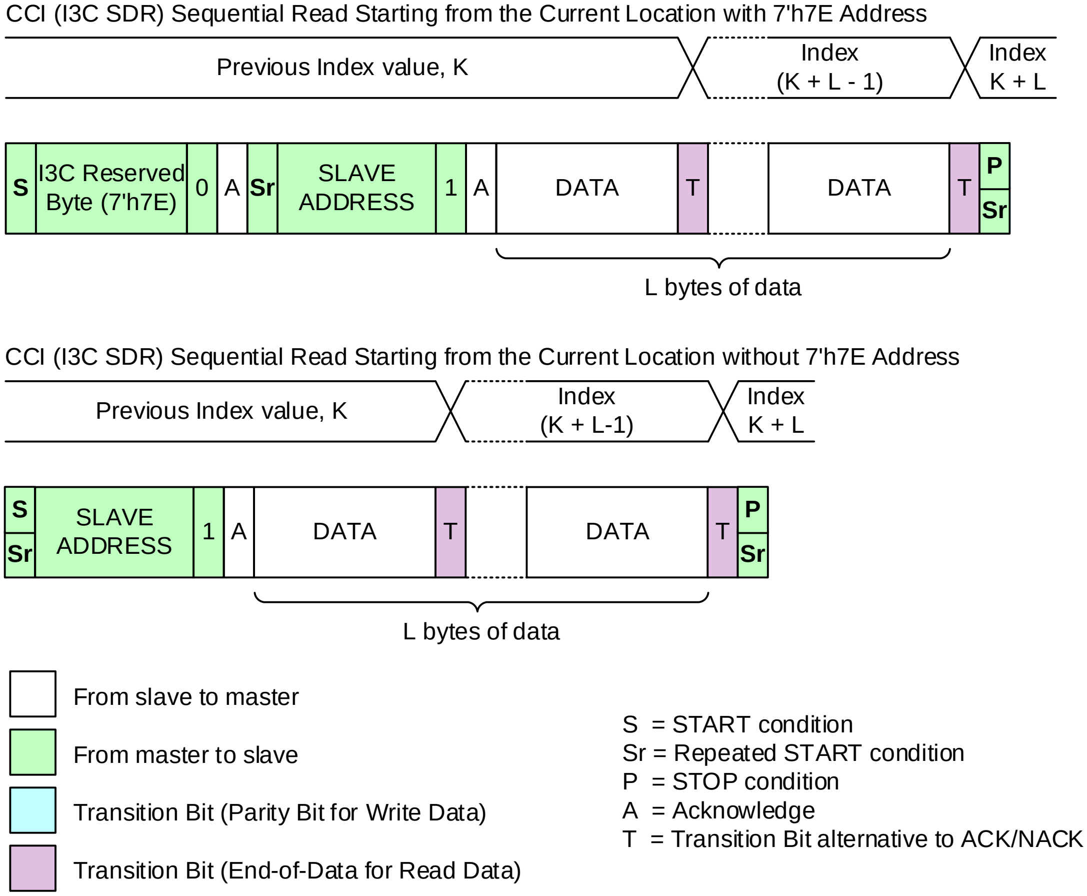

CCI (I3C SDR) Sequential Read Starting from the Current Location with 7'h7E Address
Previous Index value, K
Index (K + L - 1)
Index K + L
S
I3C Reserved Byte (7'h7E)
0
A
Sr
SLAVE ADDRESS
1
A
DATA
T
DATA
T
P
Sr
L bytes of data
CCI (I3C SDR) Sequential Read Starting from the Current Location without 7'h7E Address
Previous Index value, K
Index (K + L-1)
Index K + L
S
Sr
SLAVE ADDRESS
1
A
DATA
T
DATA
T
P
Sr
L bytes of data
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

Figure 13 CCI (I3C SDR) Sequential Read Starting from Current Location

6.2.1.2.5 CCI (I3C SDR) Single Write to Random Location

A write operation to a random location is illustrated in ***Figure 14***. The master issues a write operation to the slave, then issues the INDEX and data after the slave has acknowledged the write operation. The write operation is terminated with a STOP or Repeated START condition from the master.

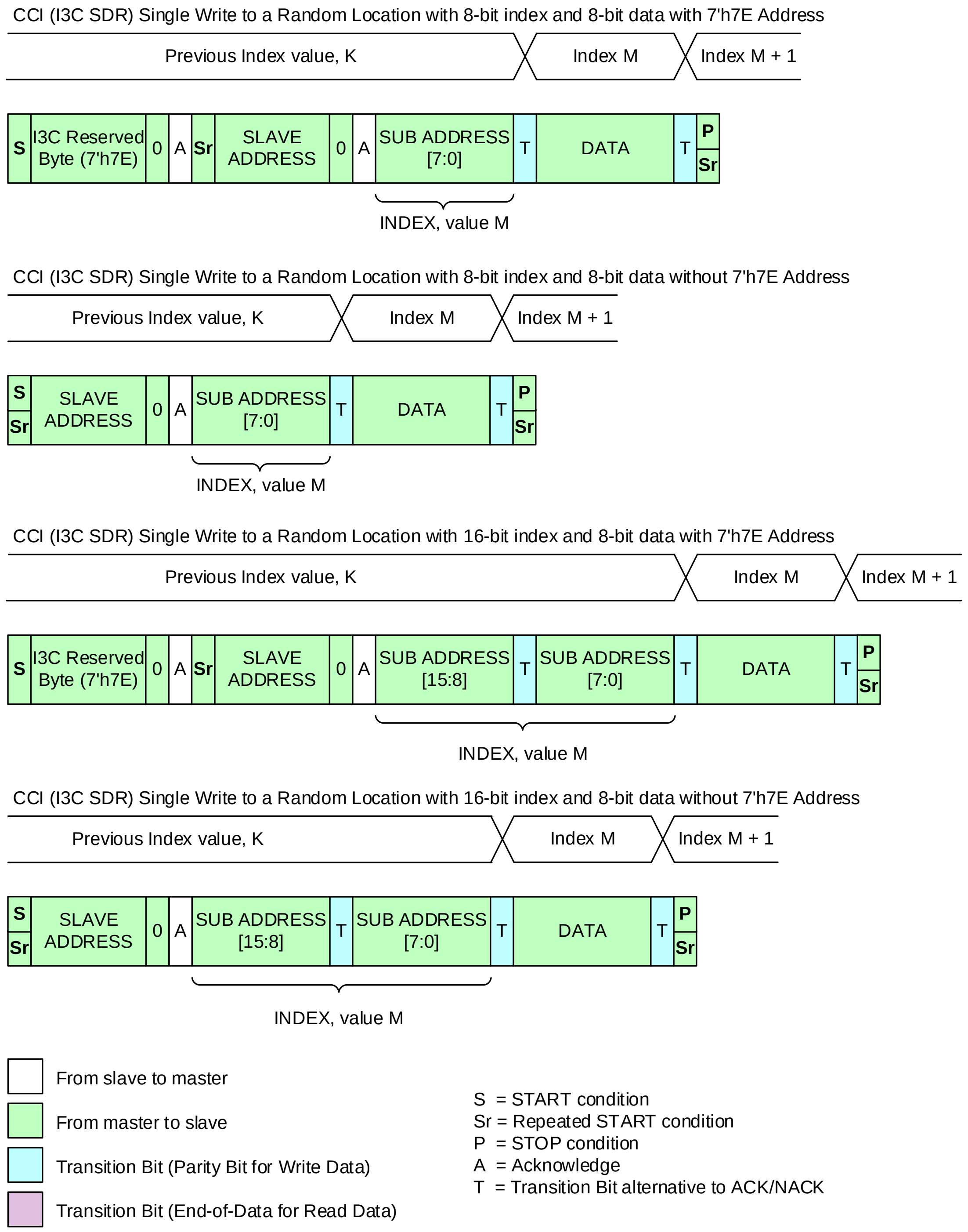

CCI (I3C SDR) Single Write to a Random Location with 8-bit index and 8-bit data with 7'h7E Address
Previous Index value, K
Index M
Index M + 1
S
I3C Reserved Byte (7'h7E)
0
A
Sr
SLAVE ADDRESS
0
A
SUB ADDRESS [7:0]
T
DATA
T
P
Sr
INDEX, value M
CCI (I3C SDR) Single Write to a Random Location with 8-bit index and 8-bit data without 7'h7E Address
Previous Index value, K
Index M
Index M + 1
S
Sr
SLAVE ADDRESS
0
A
SUB ADDRESS [7:0]
T
DATA
T
P
Sr
INDEX, value M
CCI (I3C SDR) Single Write to a Random Location with 16-bit index and 8-bit data with 7'h7E Address
Previous Index value, K
Index M
Index M + 1
S
I3C Reserved Byte (7'h7E)
0
A
Sr
SLAVE ADDRESS
0
A
SUB ADDRESS [15:8]
T
SUB ADDRESS [7:0]
T
DATA
T
P
Sr
INDEX, value M
CCI (I3C SDR) Single Write to a Random Location with 16-bit index and 8-bit data without 7'h7E Address
Previous Index value, K
Index M
Index M + 1
S
Sr
SLAVE ADDRESS
0
A
SUB ADDRESS [15:8]
T
SUB ADDRESS [7:0]
T
DATA
T
P
Sr
INDEX, value M
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

Figure 14 CCI (I3C SDR) Single Write to Random Location

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

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

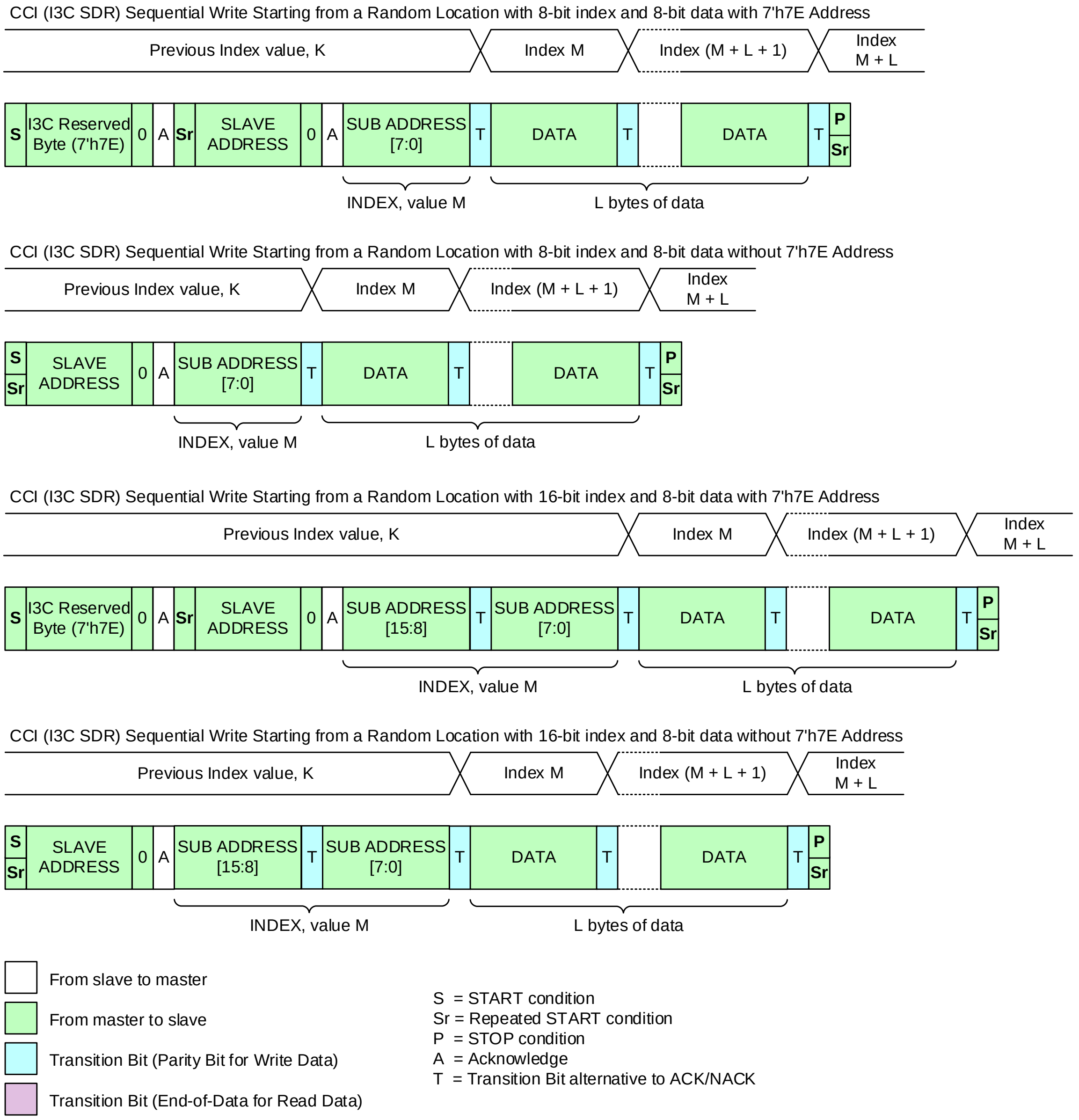


Figure 15 CCI (I3C SDR) Sequential Write Starting from Random Location

6.2.2 CCI (I3C DDR) Data Transfer Protocol

6.2.2.1 CCI (I3C DDR) Message Type

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

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

CCI (I3C DDR) supports either:

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

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

or:

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

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

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

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

Examples:

- 0 LENGTH means 1 byte
- 255 LENGTH means 256 bytes

When a multi-byte register is accessed via **CCI (I3C DDR)**, the transmission byte order described in ***Section 6.6*** shall be the same as for **CCI (I^{2}C)** and **CCI (I3C SDR)**.

Example:

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

6.2.2.2 CCI (I3C DDR) Read/Write Operations

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

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***

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

6.2.2.2.1 CCI (I3C DDR) Command Definitions

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

This single Command Code is sufficient, because the slave can still distinguish between three different 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”.

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

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

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.

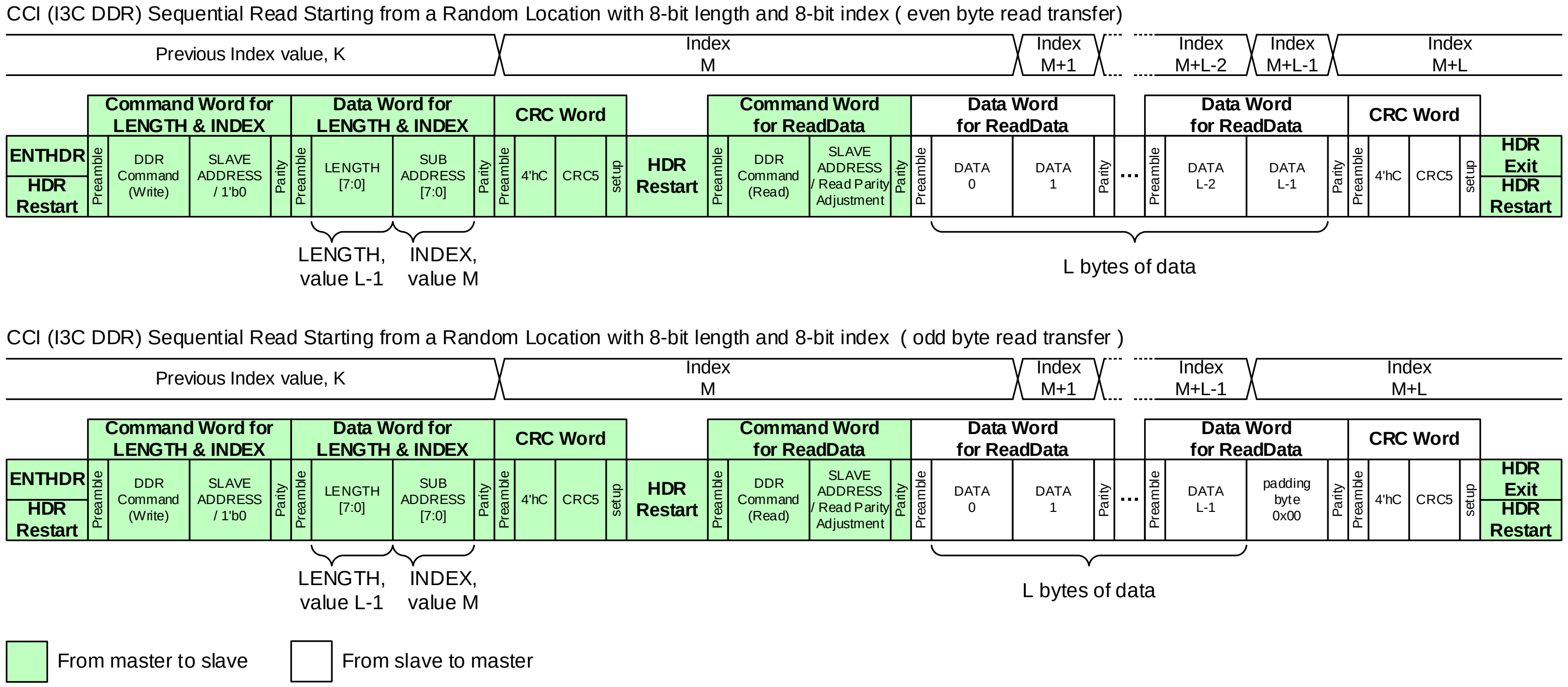
6.2.2.2.2 CCI (I3C DDR) Sequential Read From Random Location

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

- The master shall transmit:
  - The HDR-DDR Command Word for LENGTH and INDEX
  - The HDR-DDR Data Word, including LENGTH and INDEX
  - The HDR-DDR CRC Word
  - The HDR Restart Pattern
  - The HDR-DDR Command Word for ReadData
- Then the slave shall send one or more HDR-DDR Read Data Words followed by the HDR-DDR CRC Word
- Finally the master shall send either the HDR Restart Pattern or the HDR Exit Pattern.

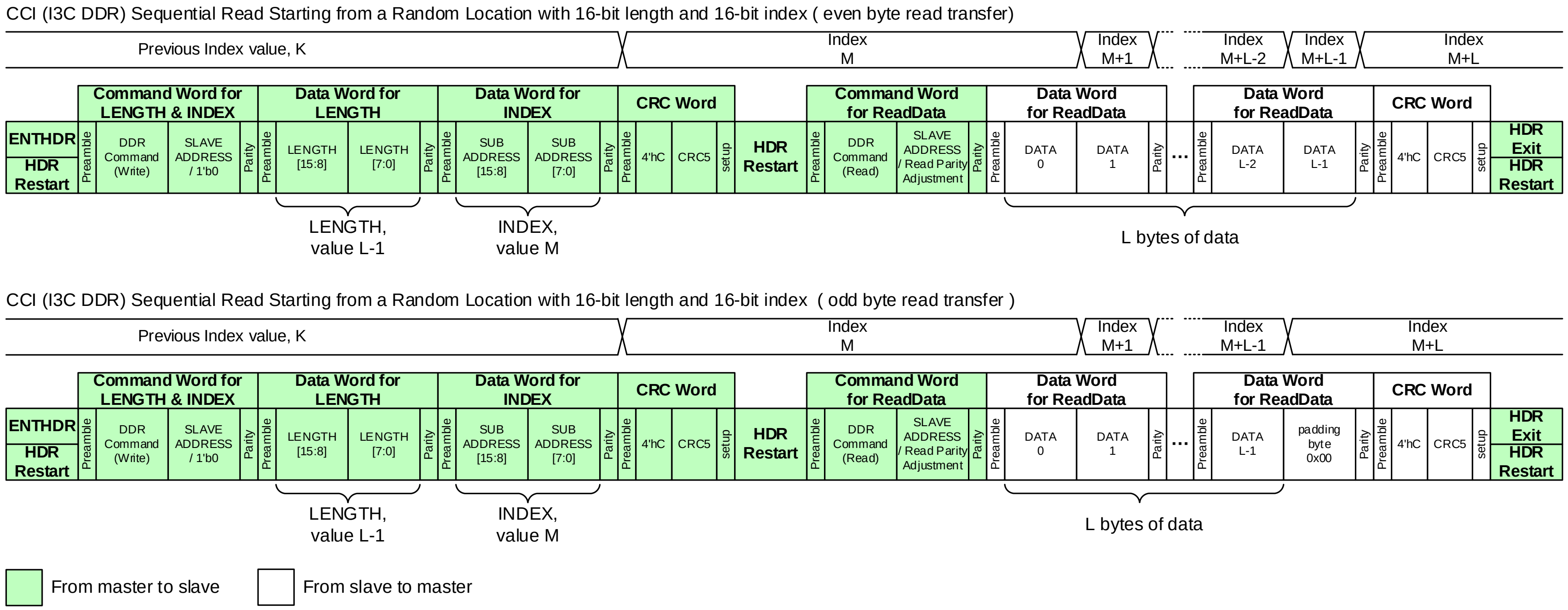
If the number of 8-bit data words read is odd (i.e. the value in the LENGTH field is even), then the slave shall insert one padding byte in the second byte of the last data word, with value 8'h00. The slave shall not increment INDEX by the padding byte. The master shall take into account that the data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

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

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)

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).*

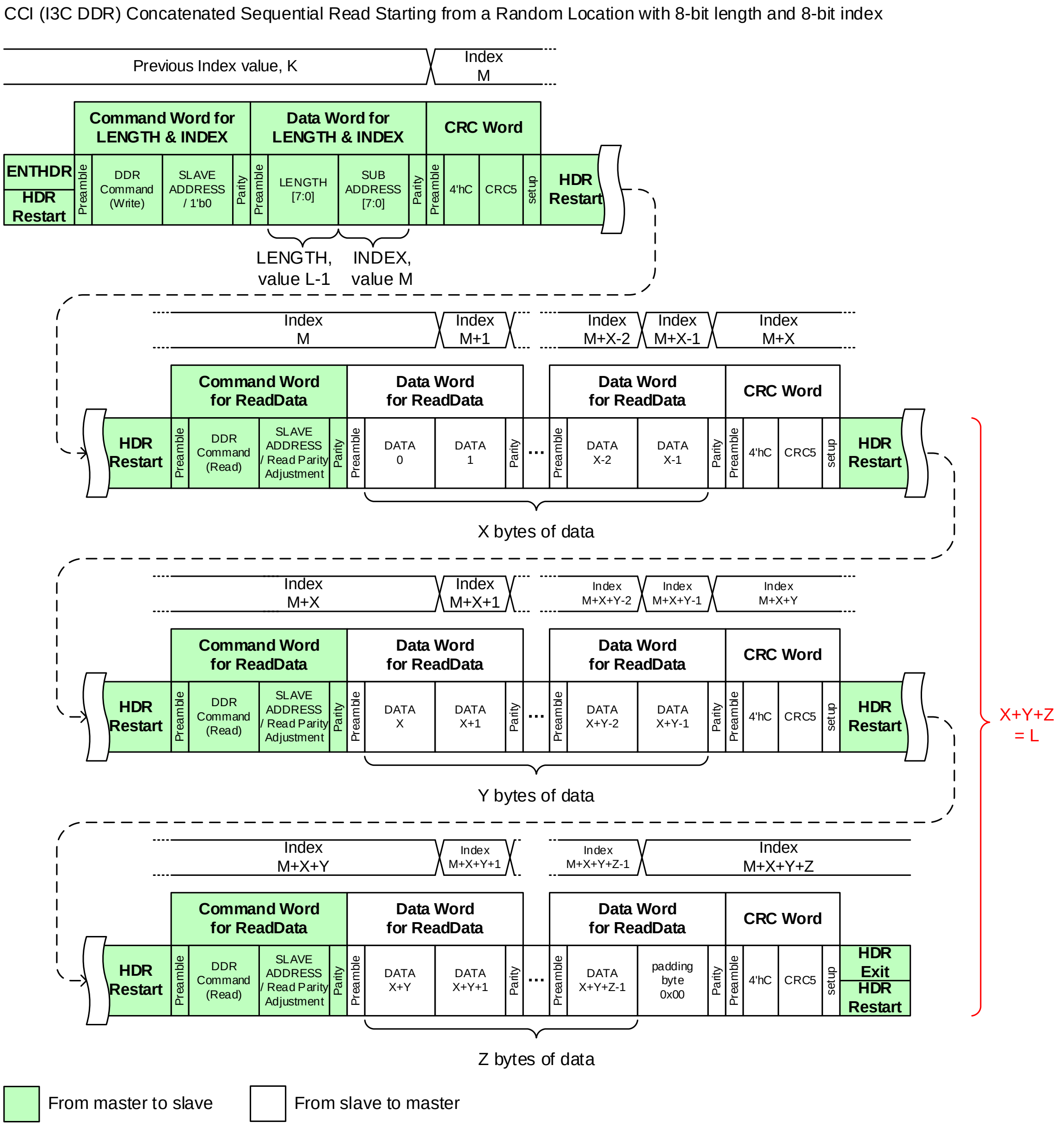


Figure 18 CCI (I3C DDR) Concatenated Sequential Read, Random Location: 8-bit LENGTH & INDEX

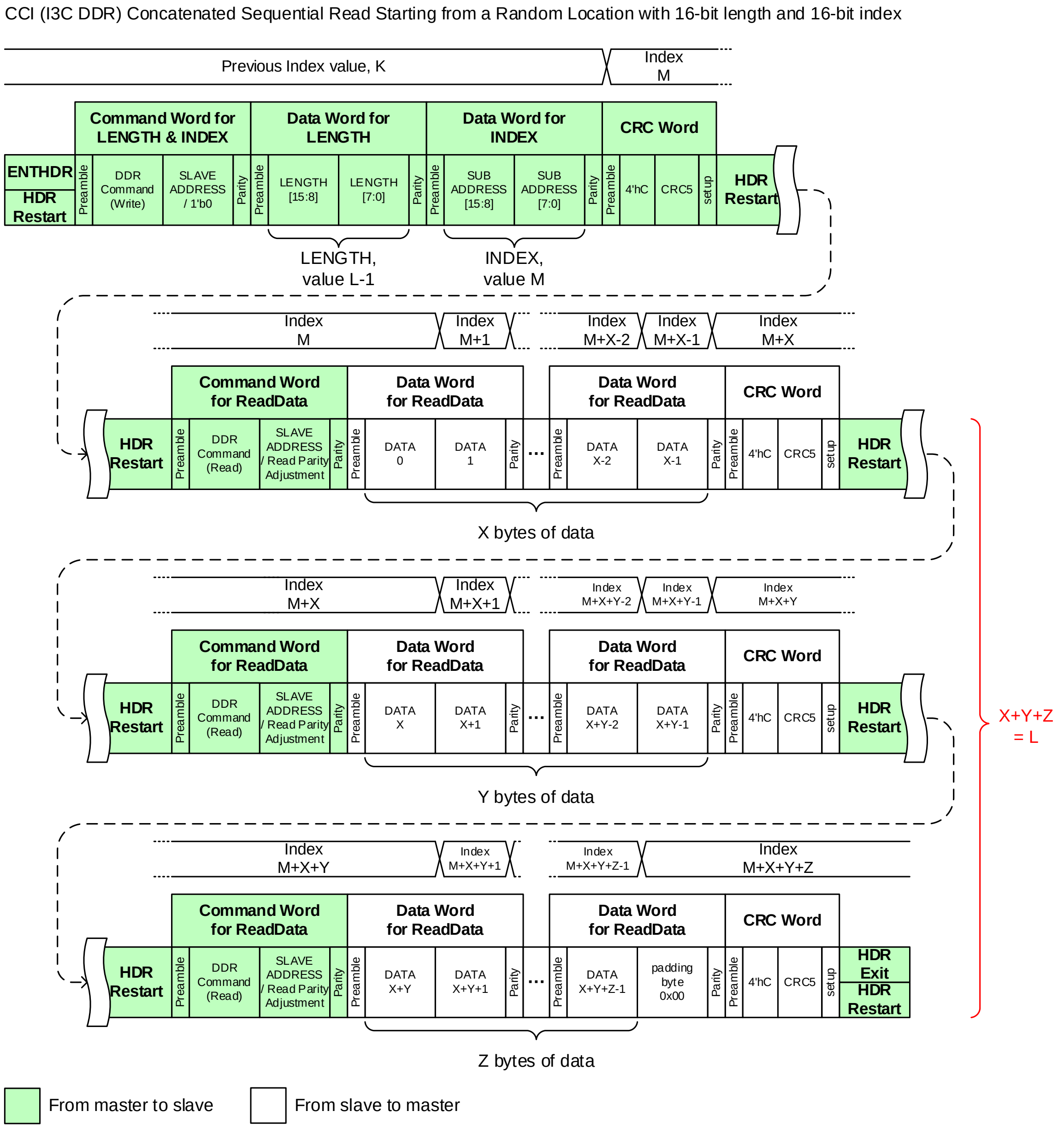


Figure 19 CCI (I3C DDR) Concatenated Sequential Read, Random Location: 16-bit LENGTH & INDEX

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

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

- The HDR-DDR Command Word
- The HDR-DDR Data Word including LENGTH and INDEX
- One or more HDR-DDR Write Data Words, and
- The HDR-DDR CRC Word.

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

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

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

Example:

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

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

Note:

> *When selecting a suitable value for MWL, 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 MWL of 68 bytes or larger (i.e. 64 bytes for Data payload + 2 bytes for a Data Word for INDEX + 2 bytes for a Data Word for LENGTH).*

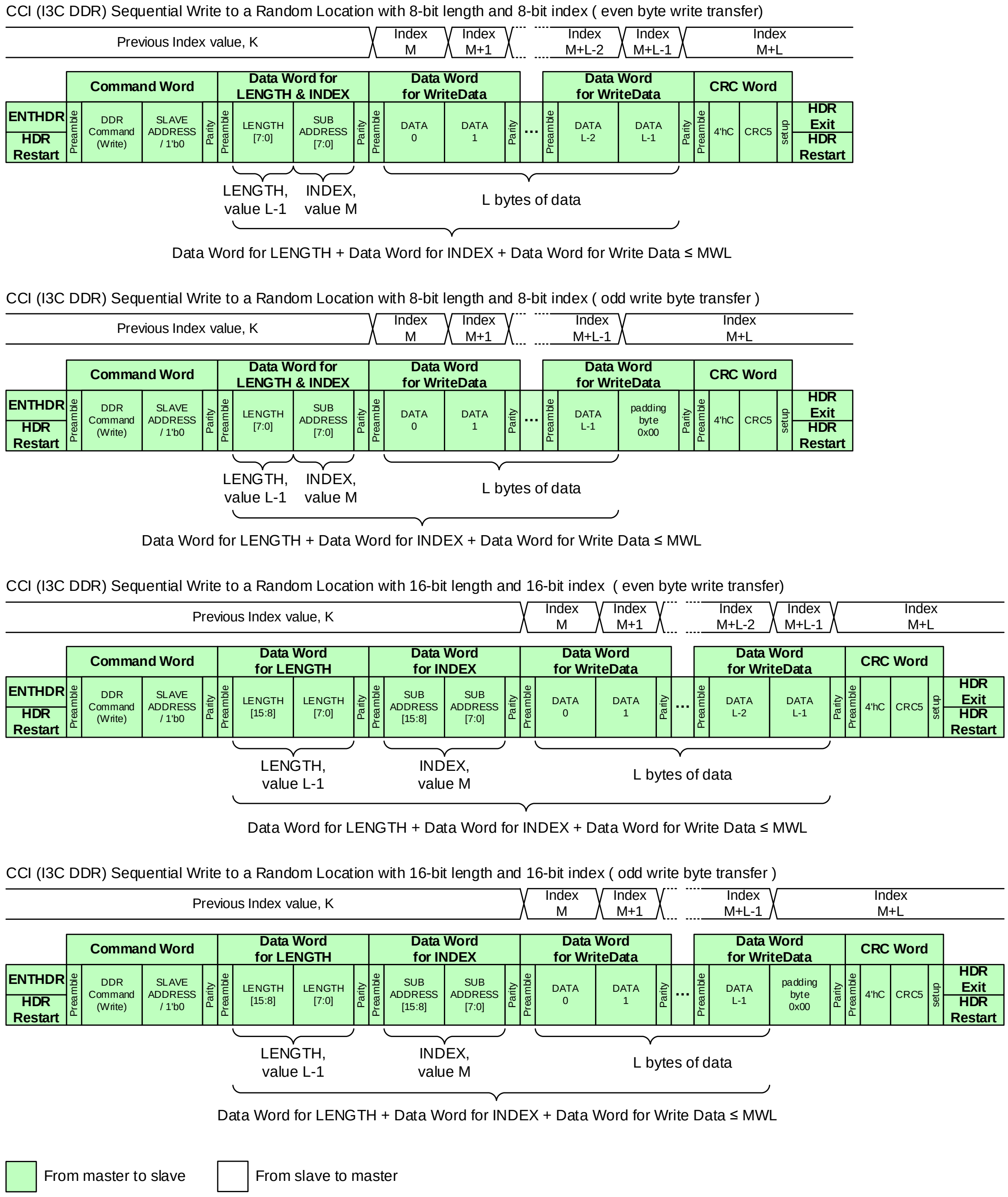


Figure 20 CCI (I3C DDR) Sequential Write Starting from Random Location

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.

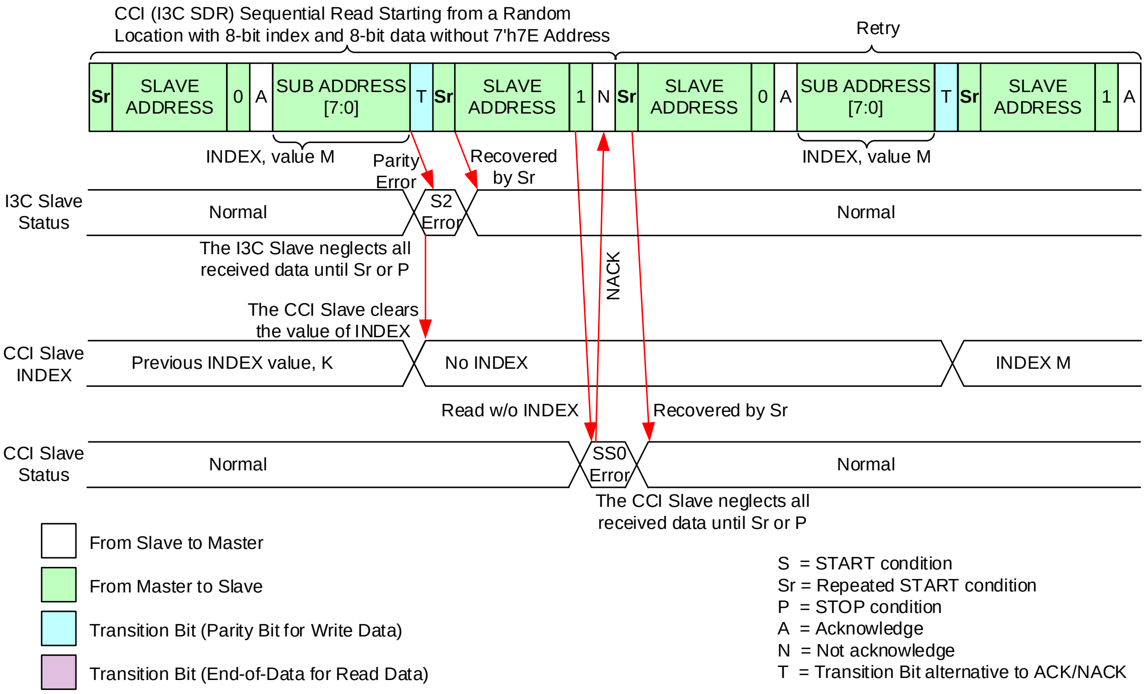


Figure 21 Example of SS0 Error Detection

6.3.2 CCI (I3C DDR) Error Detection and Recovery Method

The error detection and recovery methods specified in this Section are provided in order to avoid fatal conditions when errors occur. The CCI (I3C DDR) 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 DDR) compatible device shall support both the methods defined by I3C and the 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

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

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

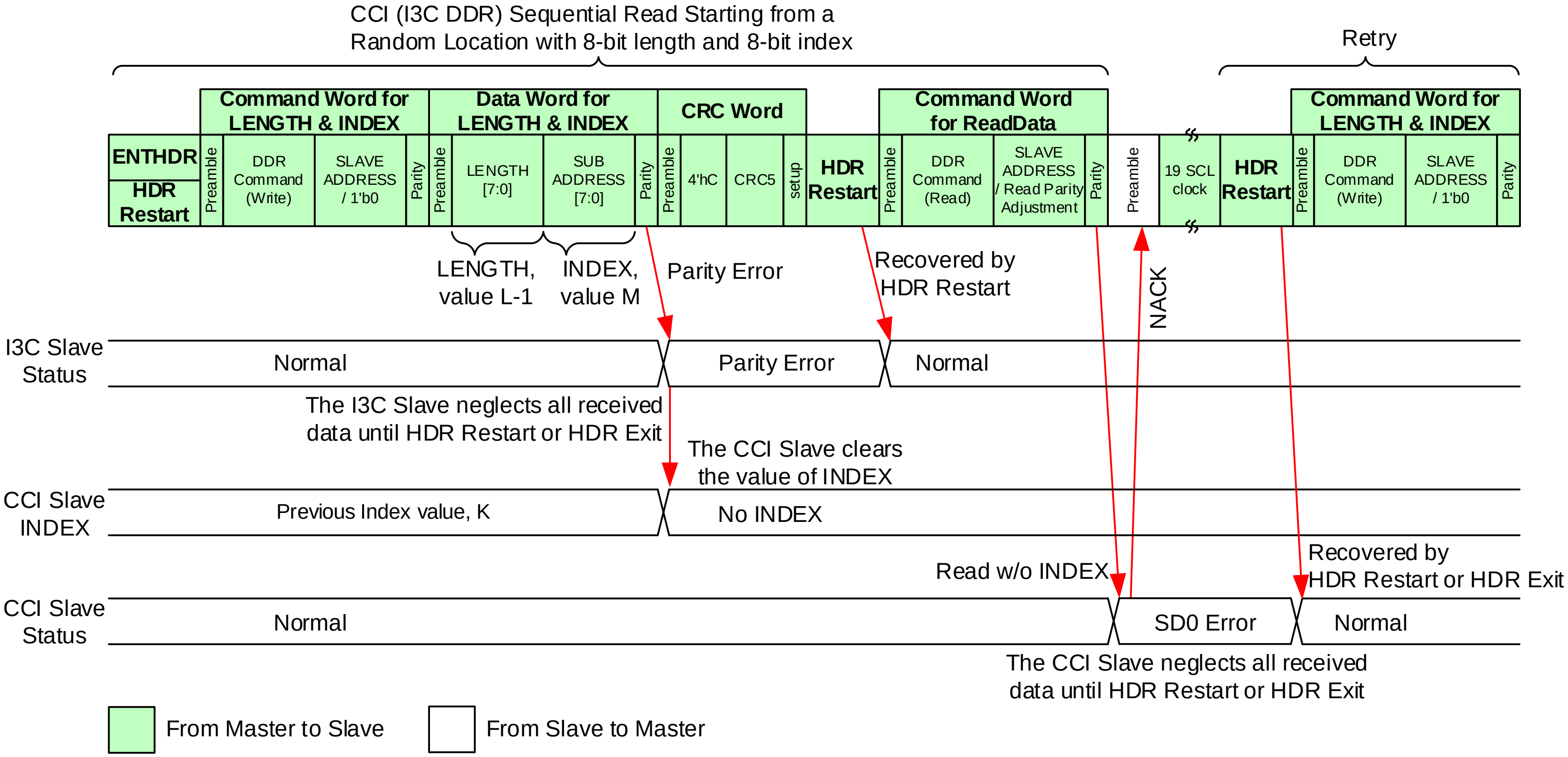
The CCI Slave shall clear the INDEX value when the I3C Slave detects an I3C DDR error defined in the I3C specification (Framing Error, Parity Error, CRC5 Error, or optional Monitoring Error) during the "CCI (I3C DDR) Read/Write Operations" in ***Table 3***. Note that this rule shall not be applicable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

If a parity error occurs during the sub address in a CCI (I3C DDR) Read Operation (i.e. Sequential or Concatenated Sequential Read from Random Location), 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.

6.3.2.1.2 SD0 Error

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

566



567

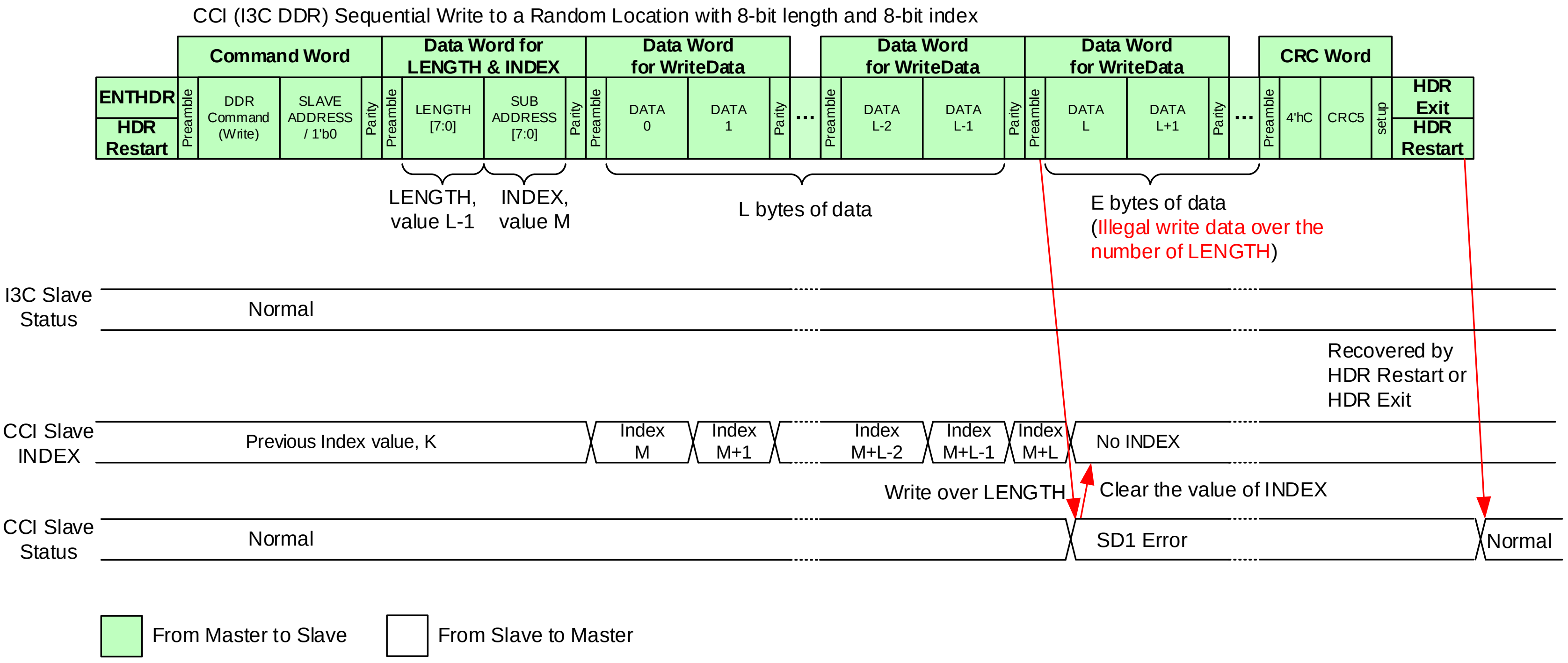
Figure 22 Example of SD0 Error Detection

6.3.2.1.3 SD1 Error

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

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

574



575

Figure 23 Example of SD1 Error Detection

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

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

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

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

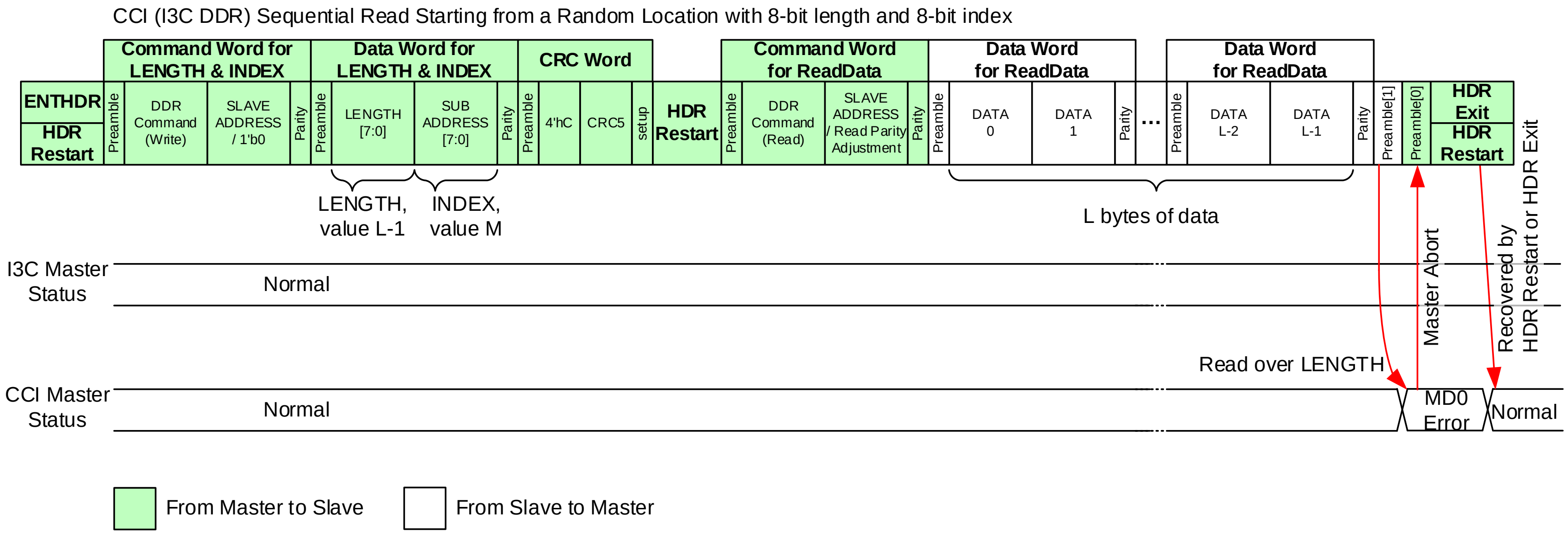


Figure 24 Example of MD0 Error Detection

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 (I^2C) 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

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

6.6.1 Overview

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

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

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

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

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

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

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

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

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

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

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

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

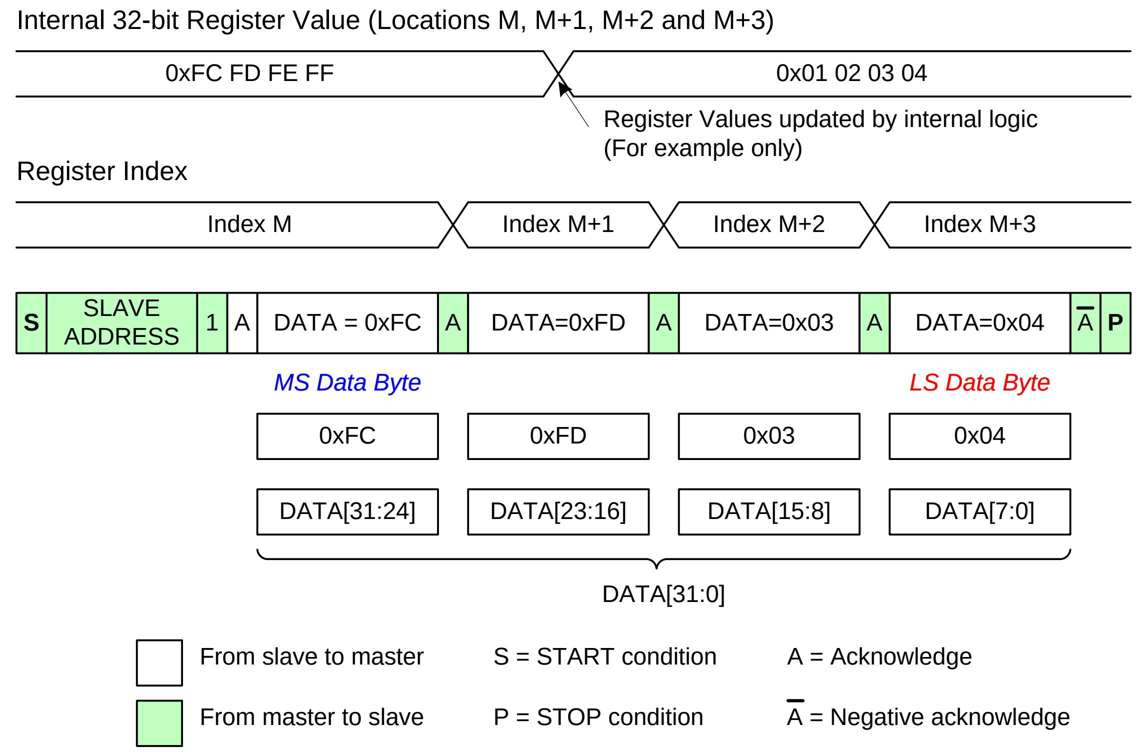


Figure 25 Corruption of 32-bit Register During Read Message

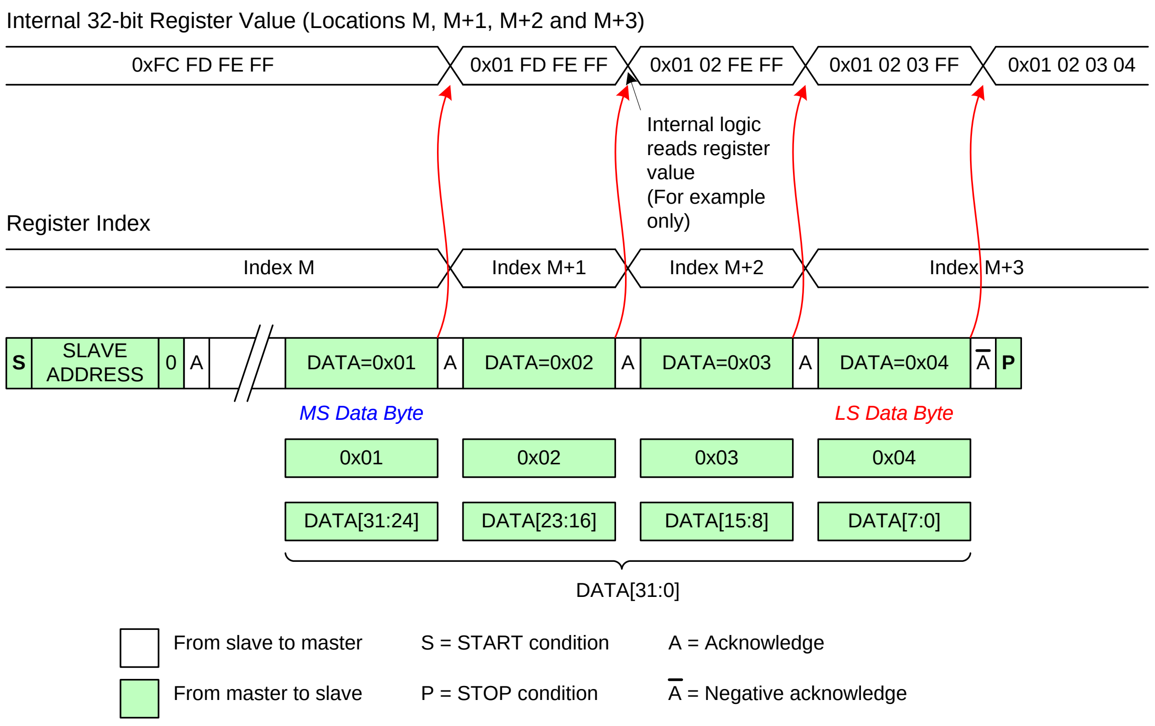


Figure 26 Corruption of 32-bit Register During Write Message

6.6.2 Transmission Byte Order for Multi-Byte Register Values

Figure 27, ***Figure 28***, and ***Figure 29*** illustrate the requirement that the first byte of a CCI Message shall always be the MS byte of a multi-byte register, and the last byte of the CCI Message shall always be the LS byte of the multi-byte register.

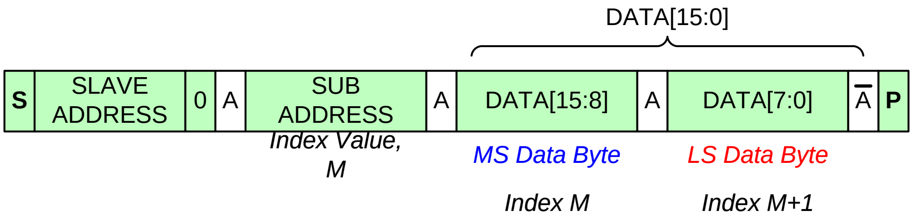


Figure 27 Example 16-bit Register Write

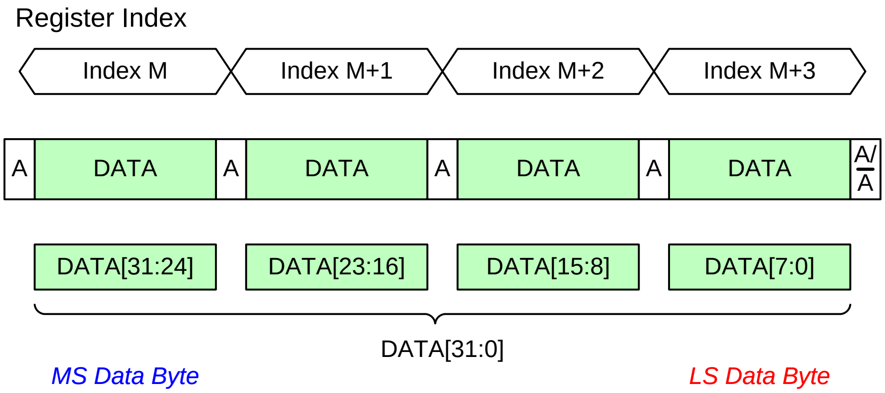


Figure 28 Example 32-bit Register Write (Address Not Shown)

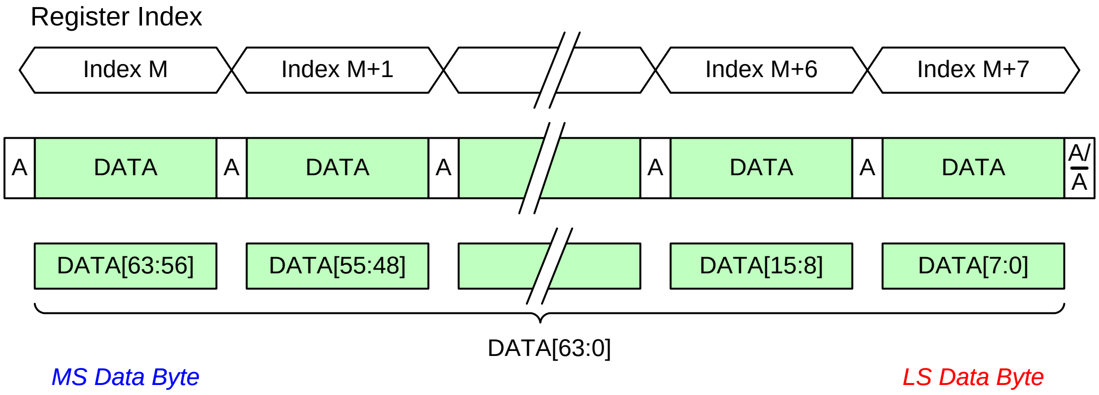


Figure 29 Example 64-bit Register Write (Address Not Shown)

6.6.3 Multi-Byte Register Protocol (Informative)

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

6.6.3.1 Reading Multi-Byte Registers

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

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

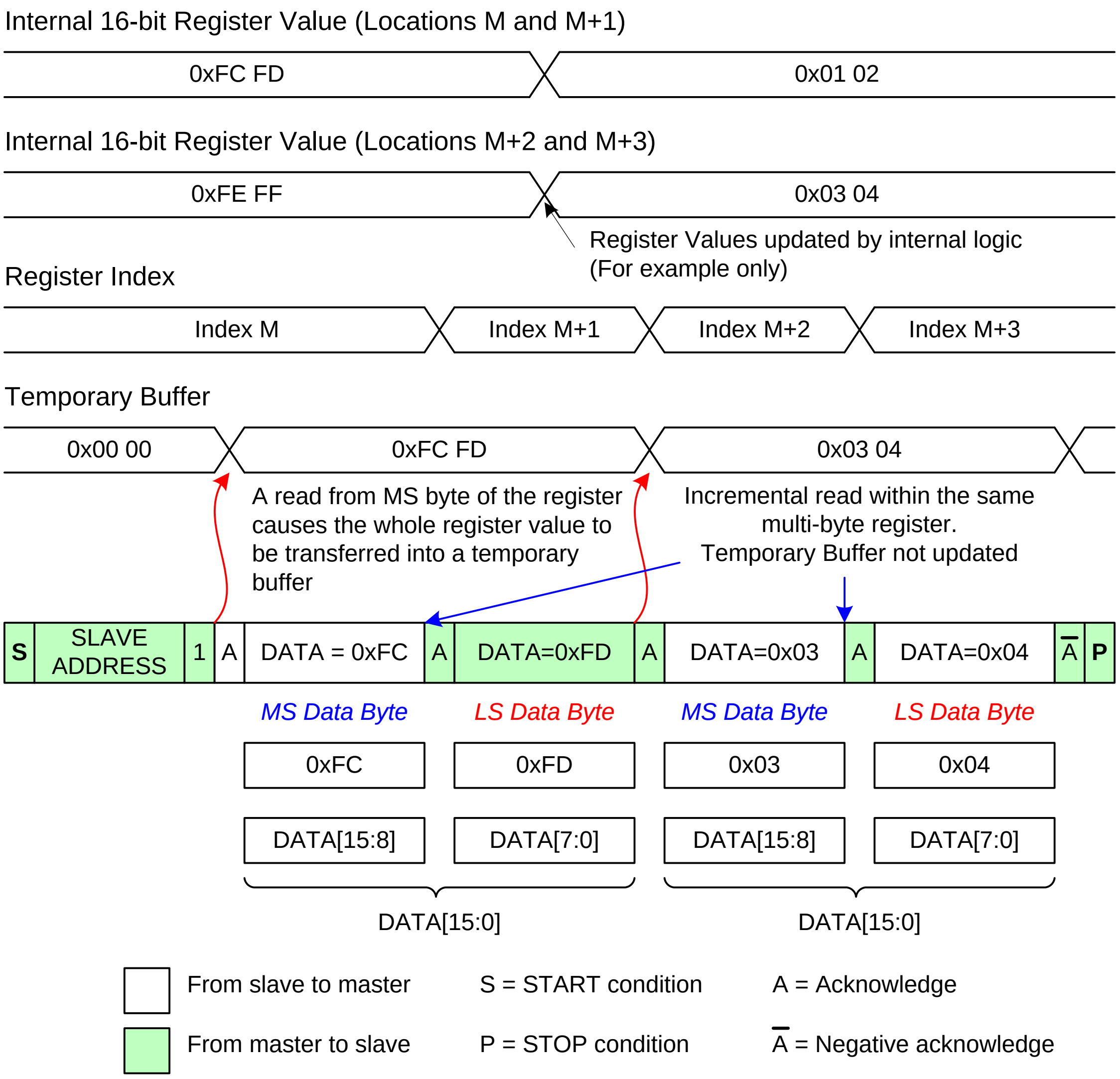


Figure 30 Example 16-bit Register Read

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

Examples of when the temporary buffer is updated include:

- When the MS byte of a register is accessed
- When the INDEX has crossed a multi-byte register boundary
- Successive single-byte reads from the same INDEX location
- When the INDEX value for the byte about to be read is ≤ the previous INDEX

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

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

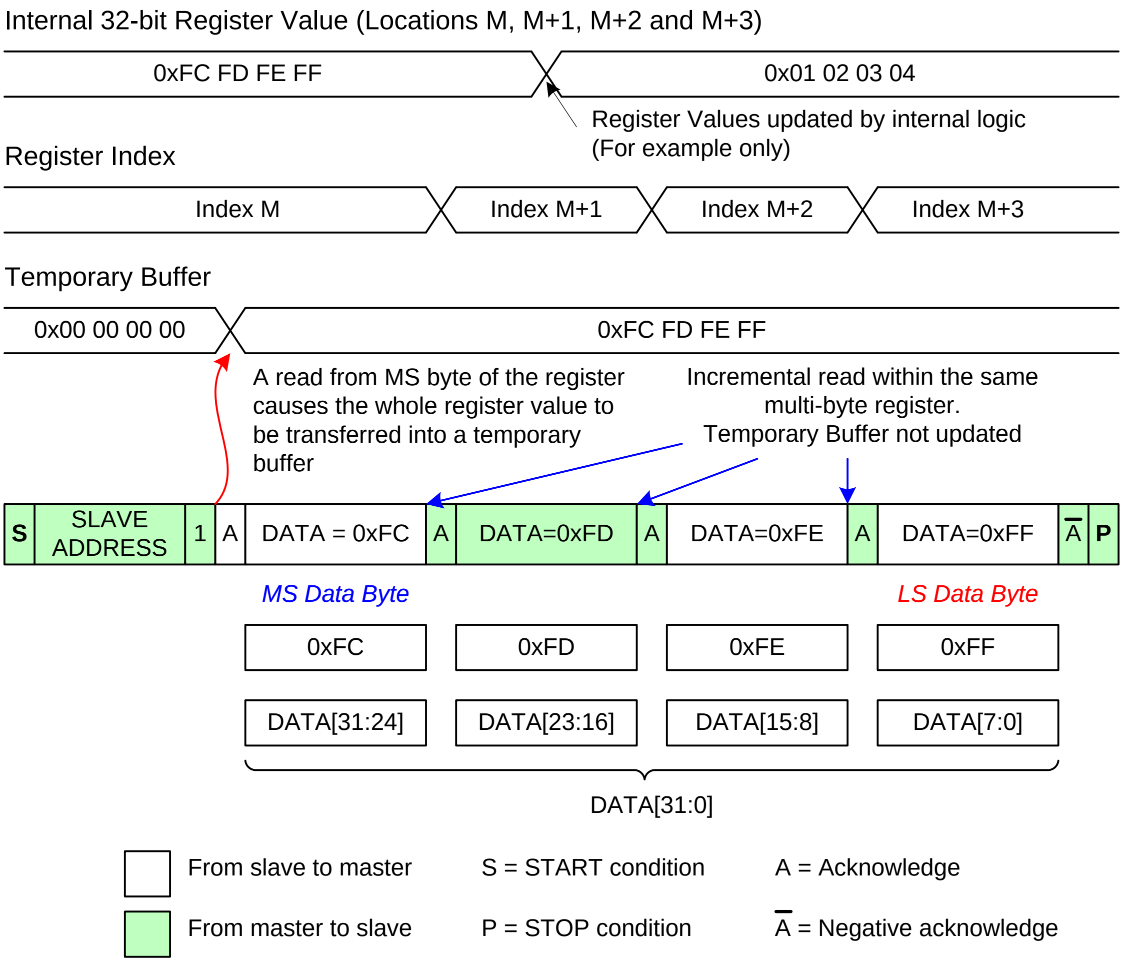


Figure 31 Example 32-bit Register Read

6.6.3.2 Writing Multi-Byte Registers

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

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

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

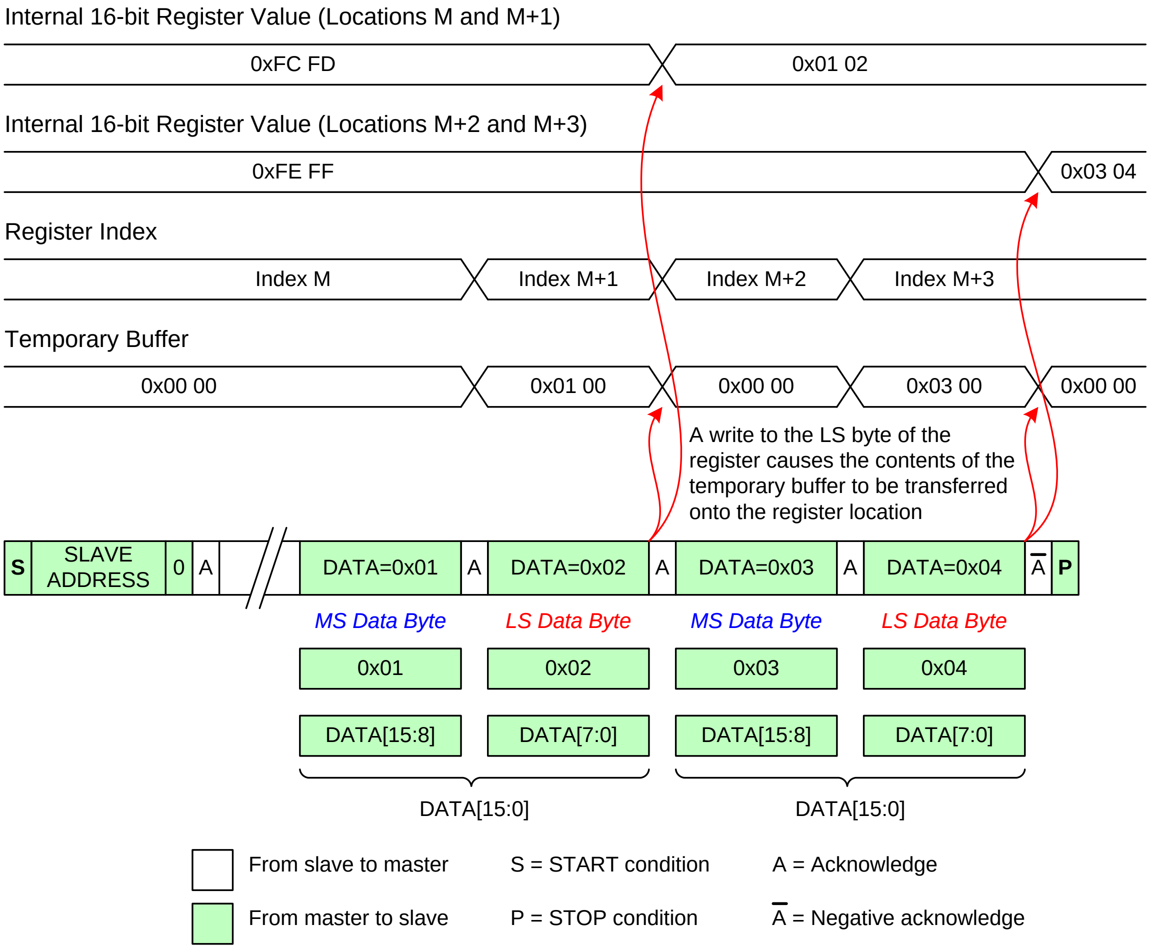


Figure 32 Example 16-bit Register Write

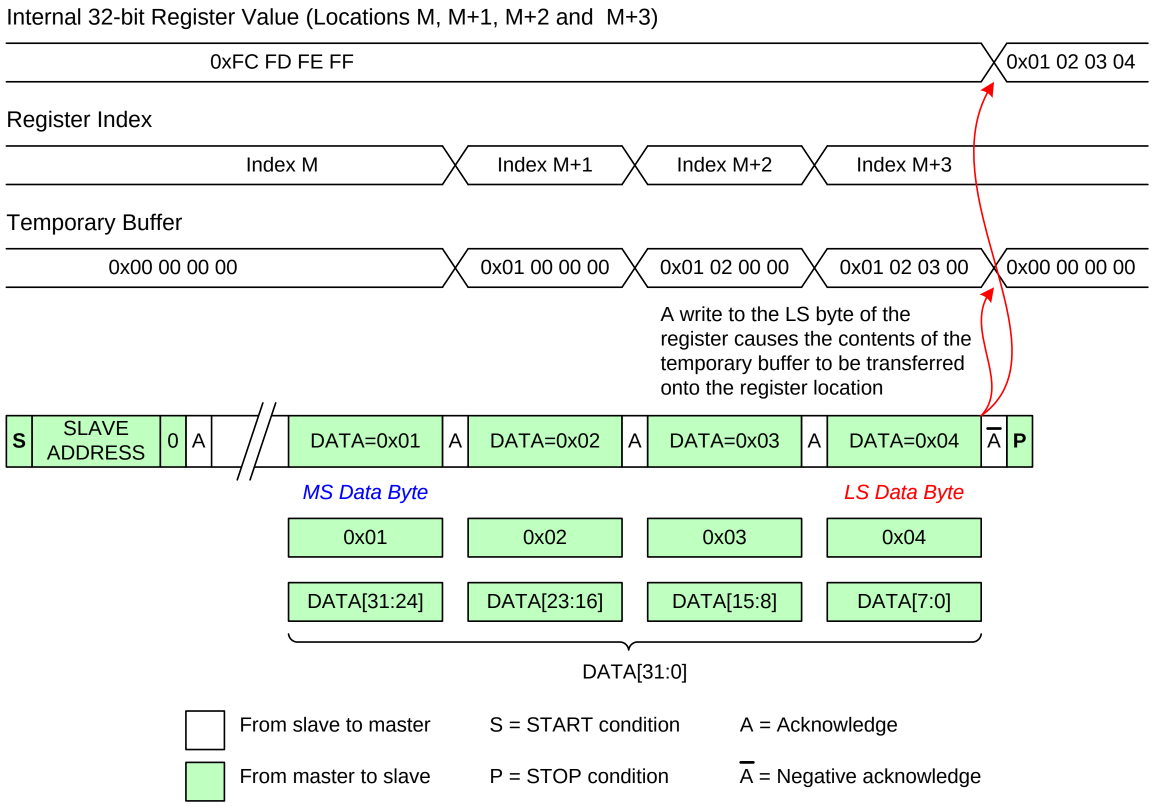


Figure 33 Example 32-bit Register Write

6.7 CCI I/O Electrical and Timing Specifications

The CCI I/O stages electrical specifications (***Table 8***) and timing specifications (***Table 9***) conform to I^{2}C Fast-mode and Fast-mode Plus devices. Information presented in ***Table 8*** is from ***[NXP01]***.

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

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:

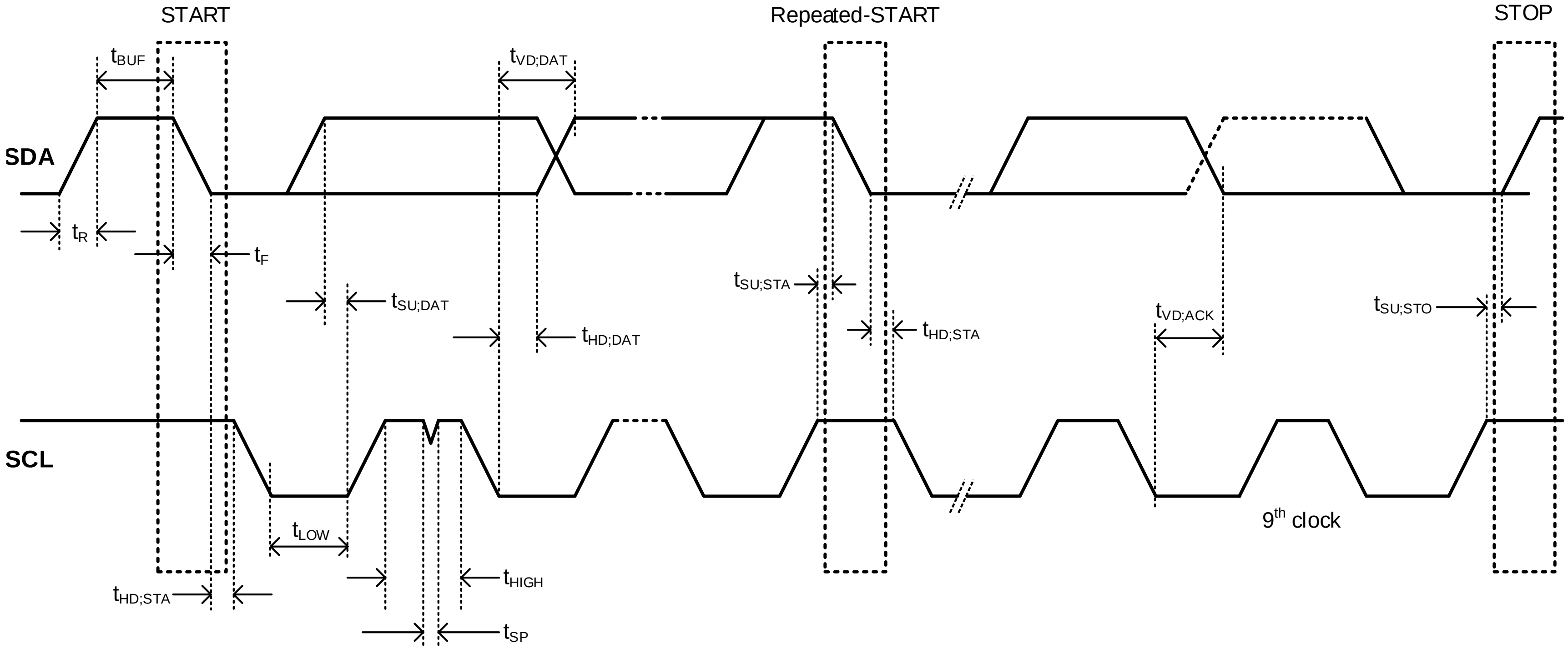
1. *Maximum VIH = V_{DDmax} + 0.5V*
2. *I/O pins of Fast-mode and Fast-mode Plus devices shall not obstruct the SDA and SCL line if V_{DD} is switched off*

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 I^2C 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)*



694

Figure 34 CCI I/O Timing

This page intentionally left blank. 695

7 Physical Layer

The CSI-2 lane management layer interfaces with the D-PHY and/or C-PHY physical layers described in ***[MIPI01]*** and ***[MIPI02]***, respectively. A device shall implement either the C-PHY 1.2 or the D-PHY 2.1 physical layer and may implement both. A practical constraint is that the PHY technologies used at both 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 physical layer are fully interoperable, provided certain new D-PHY v2.1 features are disabled as permitted by the D-PHY v2.1 specification. Such features include the Alternate Calibration Sequence, Preamble Sequence, and Extended Sync pattern as described in ***Section 6.13*** and ***Section 6.14*** of ***[MIPI01]***.

These features allow system interfaces to more robustly compensate for variations such as temperature and 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

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

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

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

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

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

8 Multi-Lane Distribution and Merging

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

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

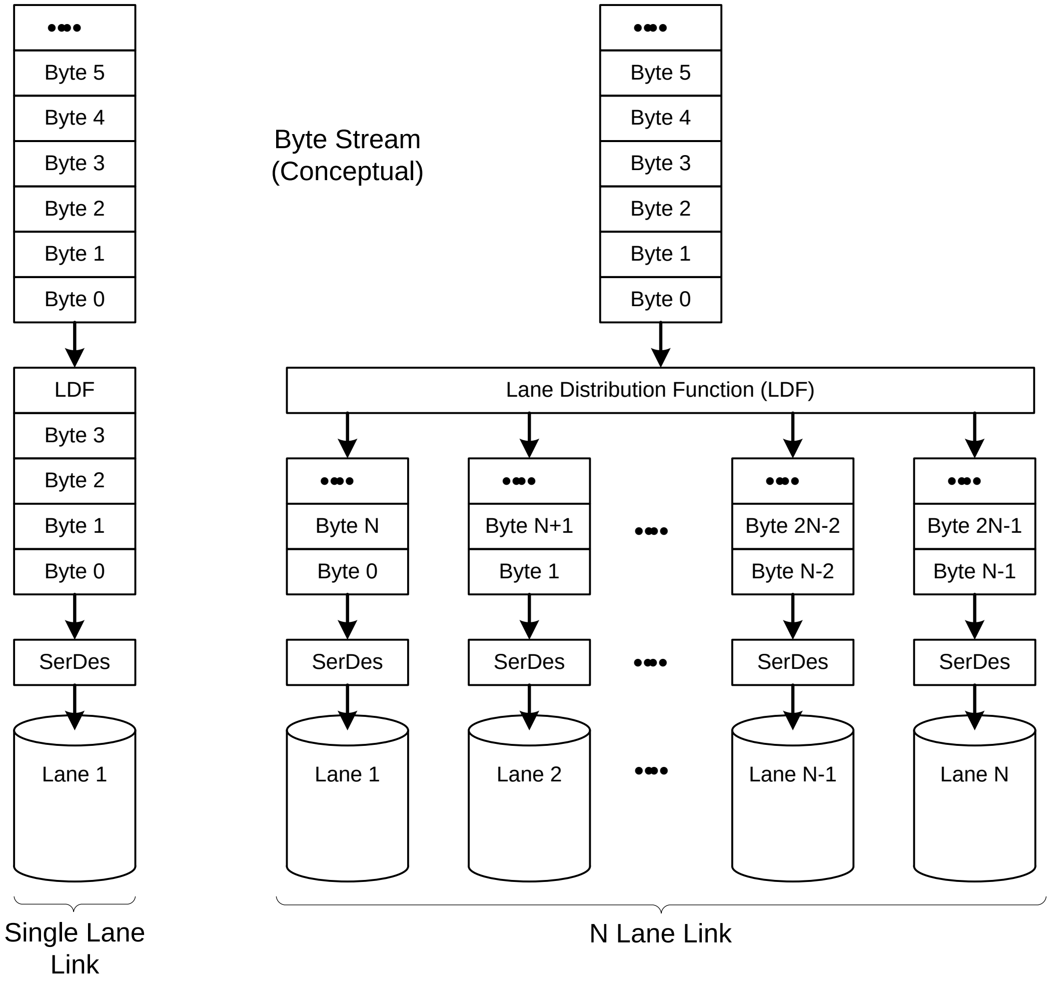


Figure 35 Conceptual Overview of the Lane Distributor Function for D-PHY

770

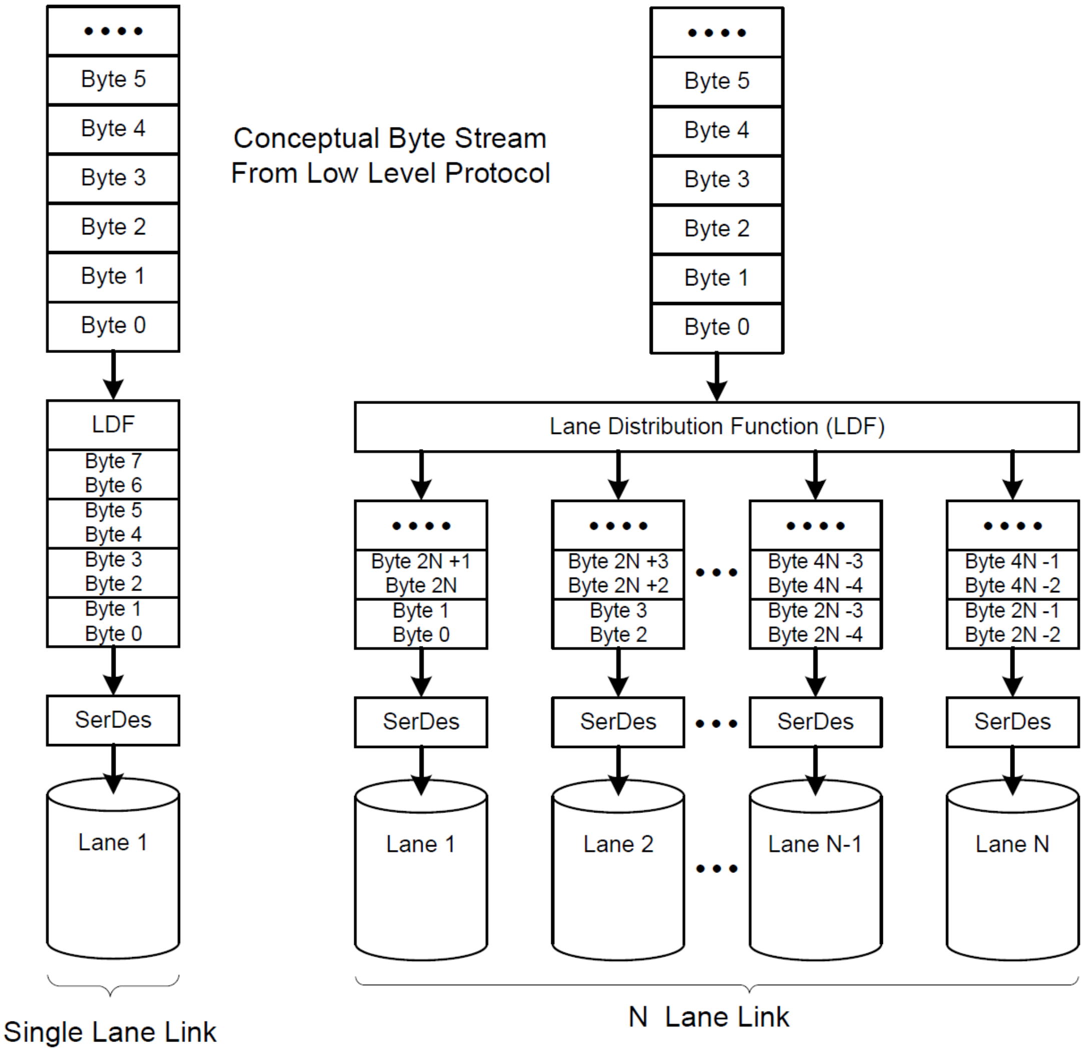
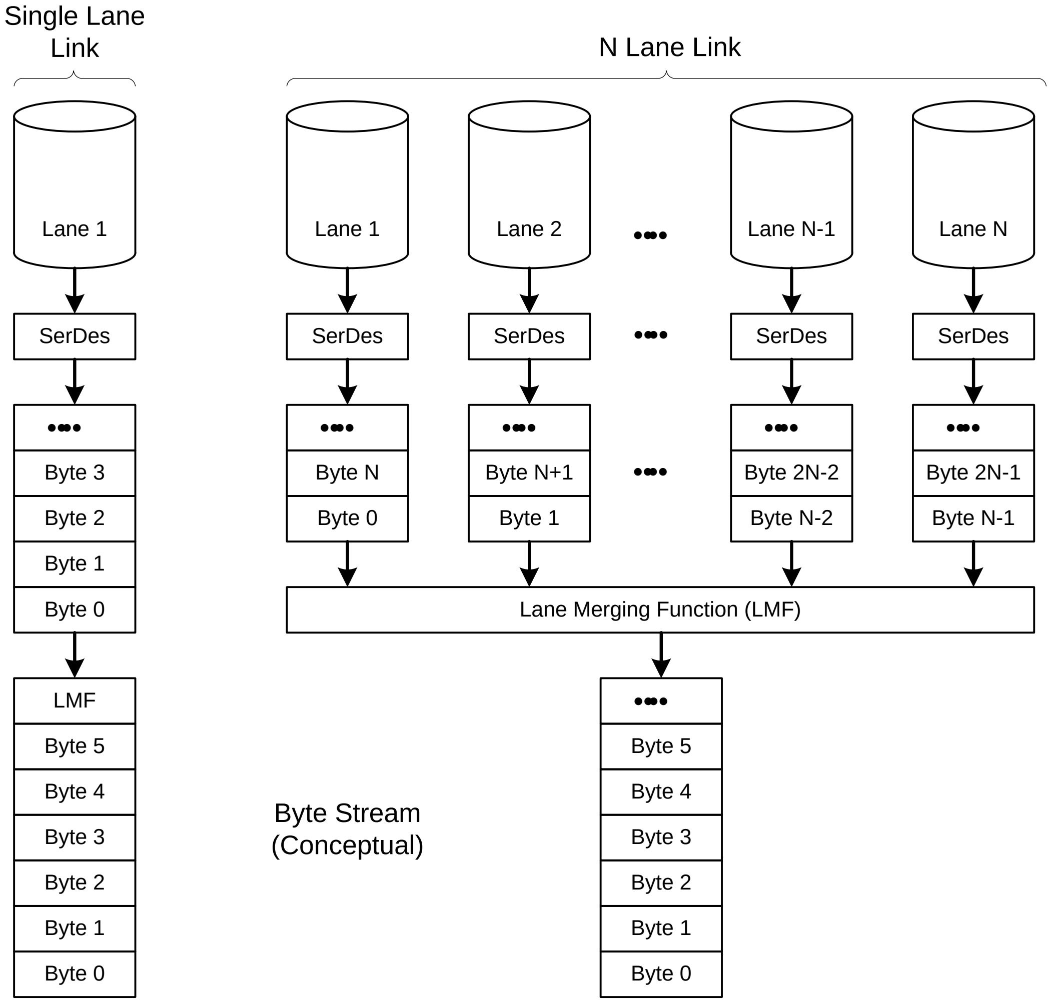


Figure 36 Conceptual Overview of the Lane Distributor Function for C-PHY



771

Figure 37 Conceptual Overview of the Lane Merging Function for D-PHY

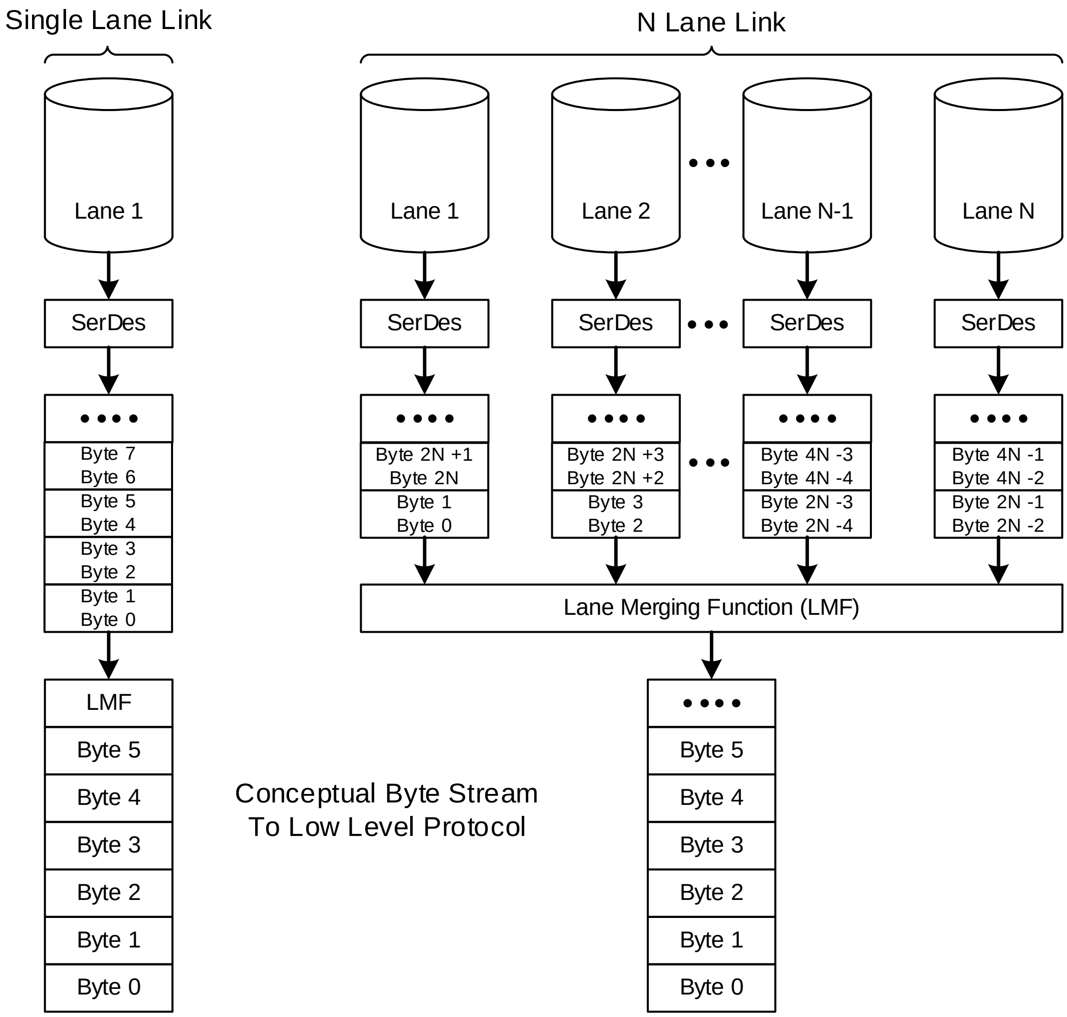


Figure 38 Conceptual Overview of the Lane Merging Function for C-PHY

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

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

- 2-Lane system (***Figure 39***): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to Lane 1, byte 3 goes to Lane 2, byte 4 goes to Lane 1, and so on.
- 3-Lane system (***Figure 40***): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to Lane 3, byte 3 goes to Lane 1, byte 4 goes to Lane 2, and so on.
- N-Lane system (***Figure 41***): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte N-1 goes to Lane N, byte N goes to Lane 1, byte N+1 goes to Lane 2, and so on.
- N-lane system (***Figure 42***) with N>4 short packet (4 bytes) transmission: byte 0 of the packet goes 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 do not receive bytes and stay in LPS state.

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

Each D-PHY data Lane operates autonomously.

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

The N PHYs on the receiving end of the link collect bytes 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 layer.

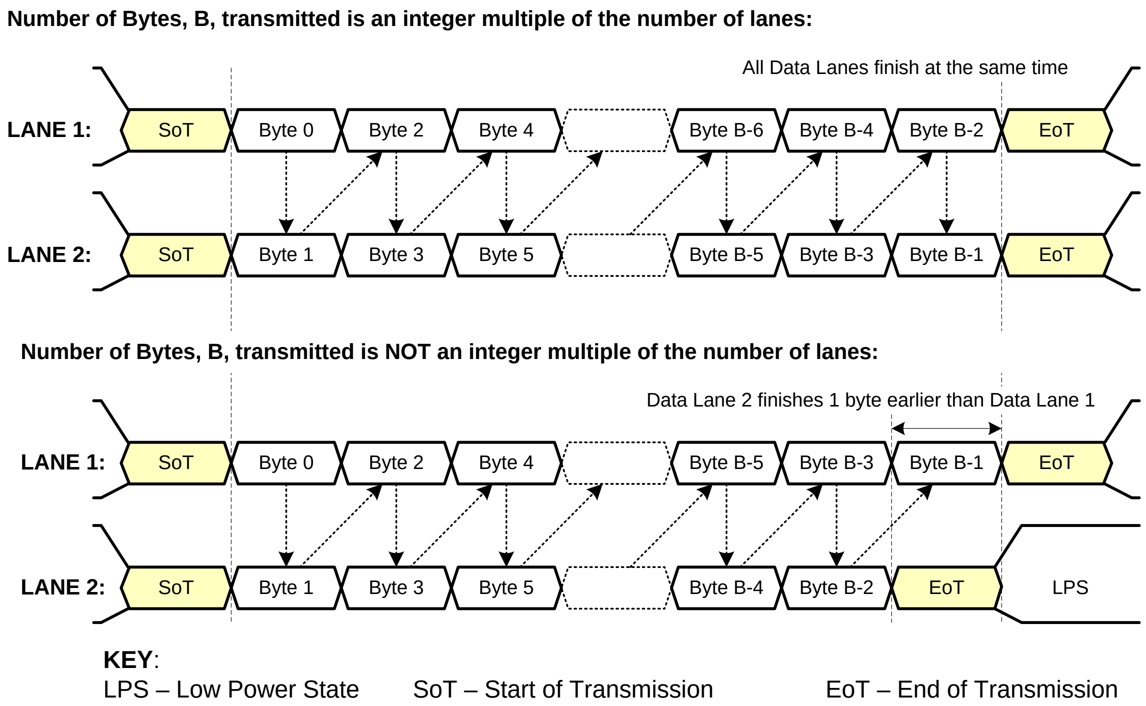
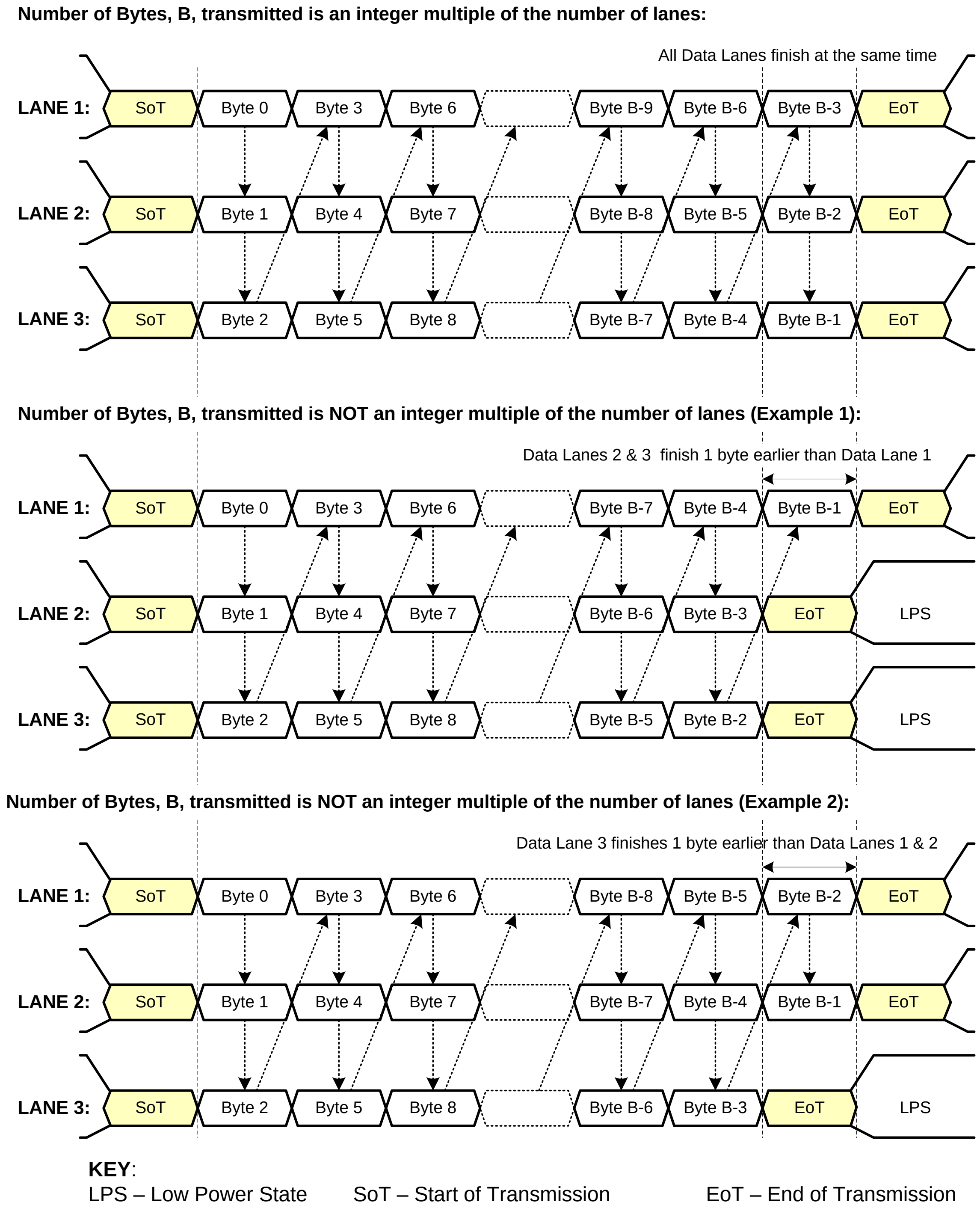

Number of Bytes, B, transmitted is an integer multiple of the number of lanes:
All Data Lanes finish at the same time
LANE 1:
SoT
Byte 0
Byte 2
Byte 4
Byte B-6
Byte B-4
Byte B-2
EoT
LANE 2:
SoT
Byte 1
Byte 3
Byte 5
Byte B-5
Byte B-3
Byte B-1
EoT
Number of Bytes, B, transmitted is NOT an integer multiple 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

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


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

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

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

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
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
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

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

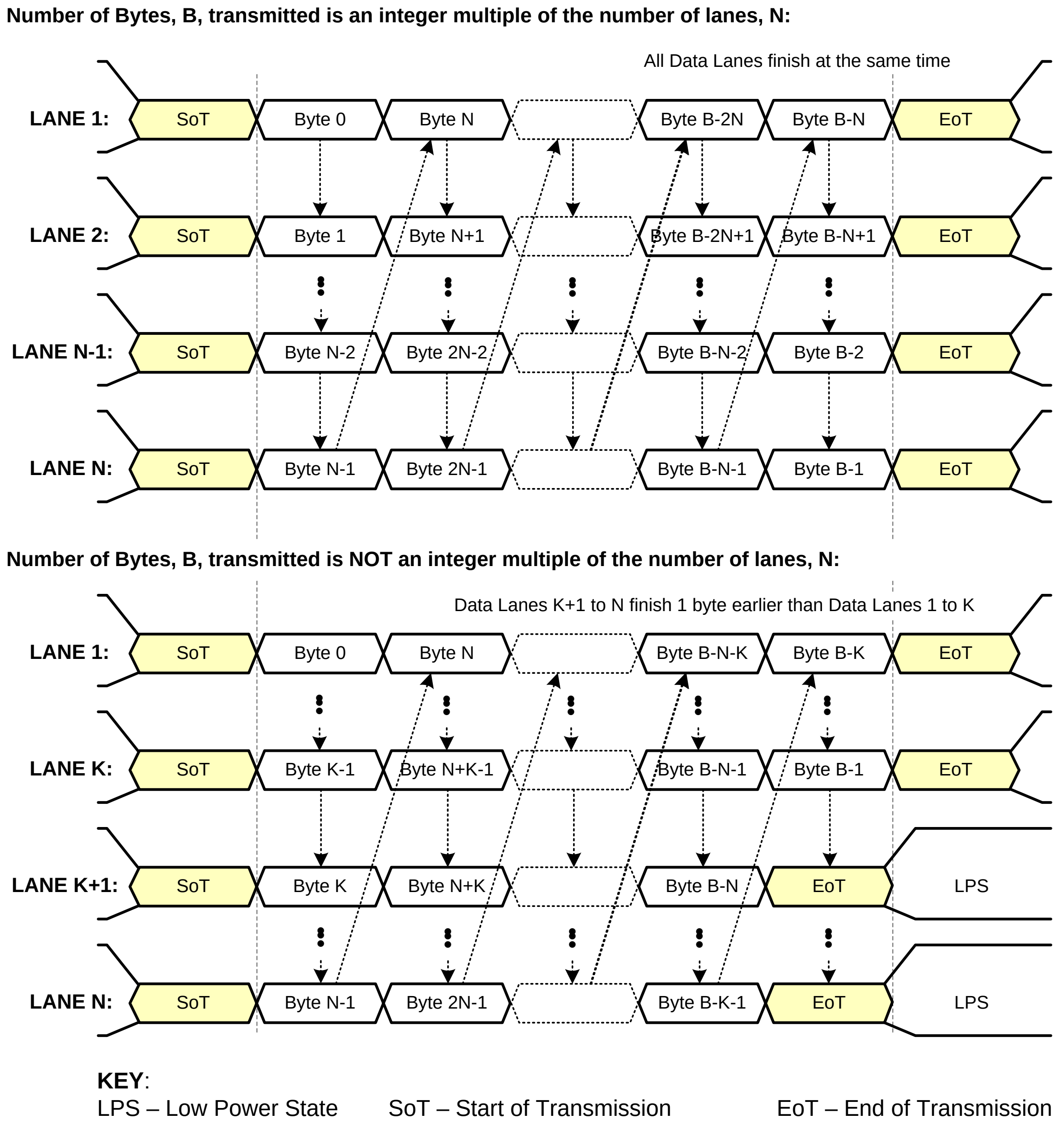


Figure 41 N-Lane Multi-Lane Example for D-PHY

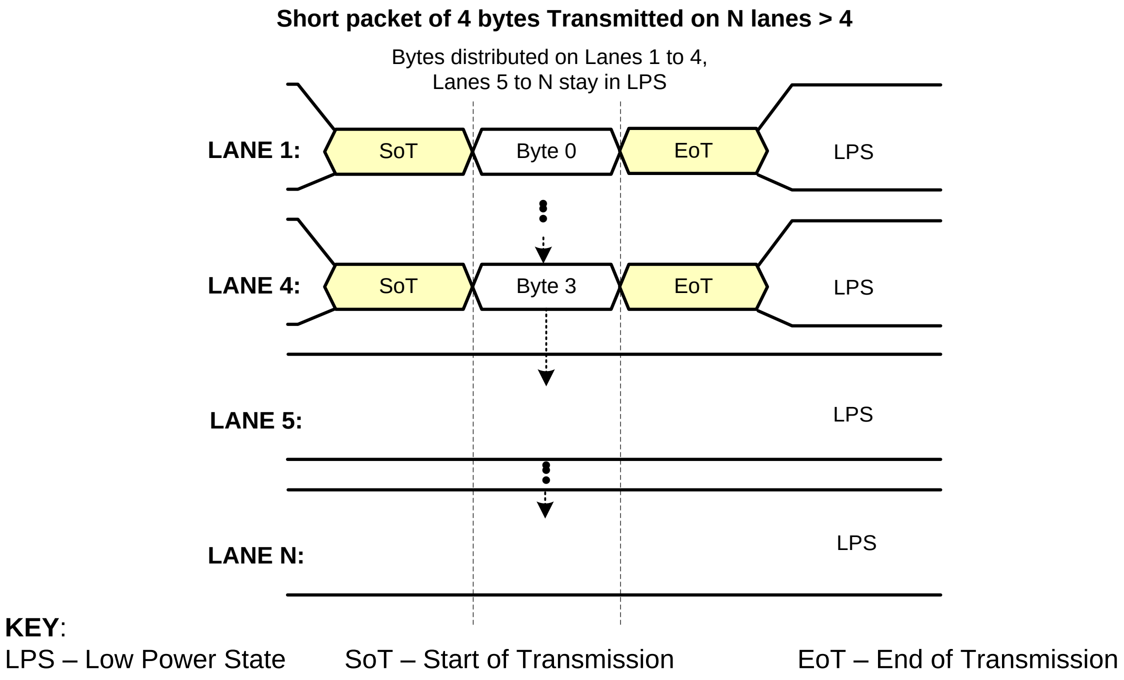

Short packet of 4 bytes Transmitted on N lanes > 4
Bytes distributed on Lanes 1 to 4,
Lanes 5 to N stay in LPS
LANE 1:
SoT
Byte 0
EoT
LPS
LANE 4:
SoT
Byte 3
EoT
LPS
LANE 5:
LPS
LANE N:
LPS
KEY:
LPS – Low Power State
SoT – Start of Transmission
EoT – End of Transmission

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 ≥ 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 ≤ n ≤ 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.

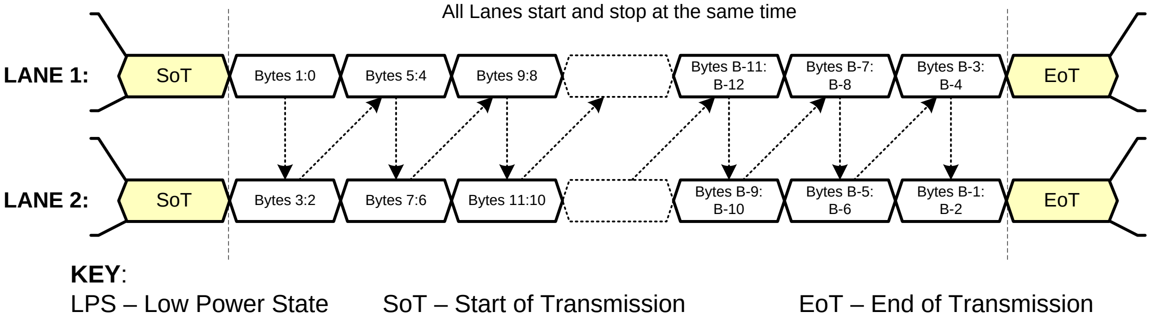


Figure 43 Two Lane Multi-Lane Example for C-PHY

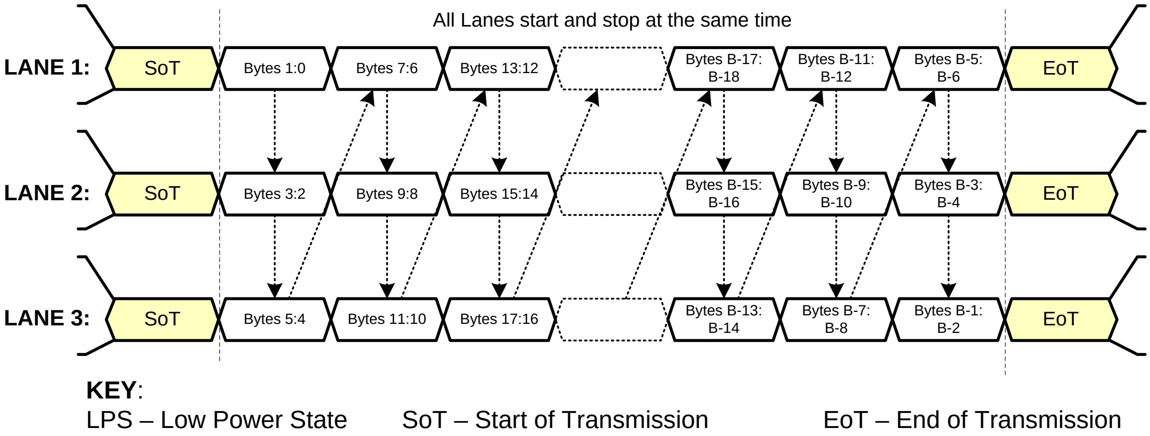


Figure 44 Three Lane Multi-Lane Example for C-PHY

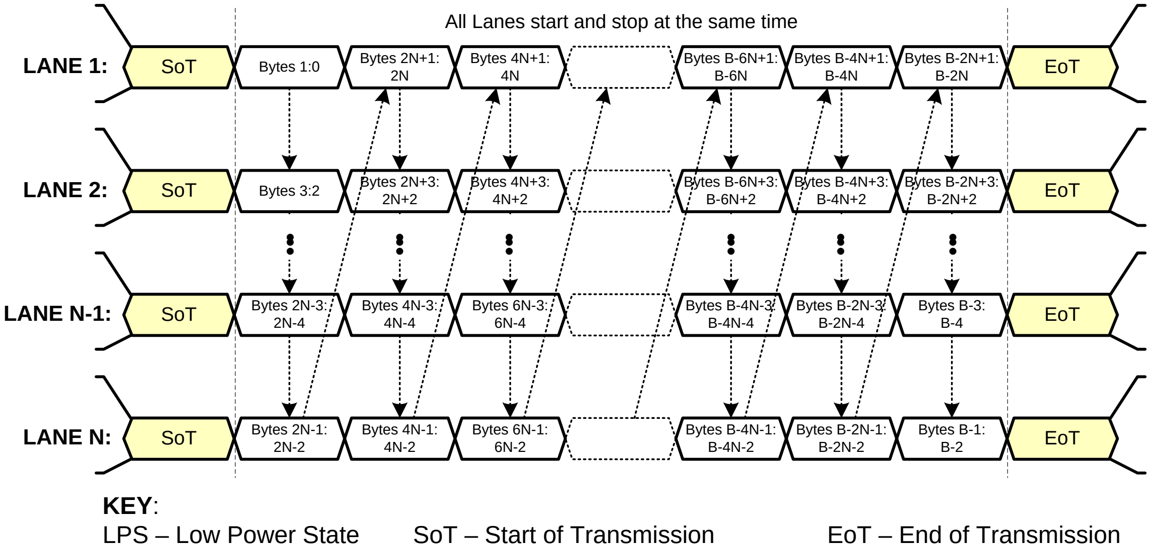


Figure 45 General N-Lane Multi-Lane Distribution for C-PHY

8.3 Multi-Lane Interoperability

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

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

Two cases:

- If M<=N then there is no loss of performance – the receiver has sufficient data Lanes to match the transmitter (***Figure 46*** and ***Figure 47***).
- If M> N then there may be a loss of performance (e.g. frame rate) as the receiver has fewer data Lanes than the transmitter (***Figure 48*** and ***Figure 49***).
- Note that while the examples shown are for the D-PHY physical layer option, the C-PHY physical layer option is handled similarly, except there is no clock Lane.

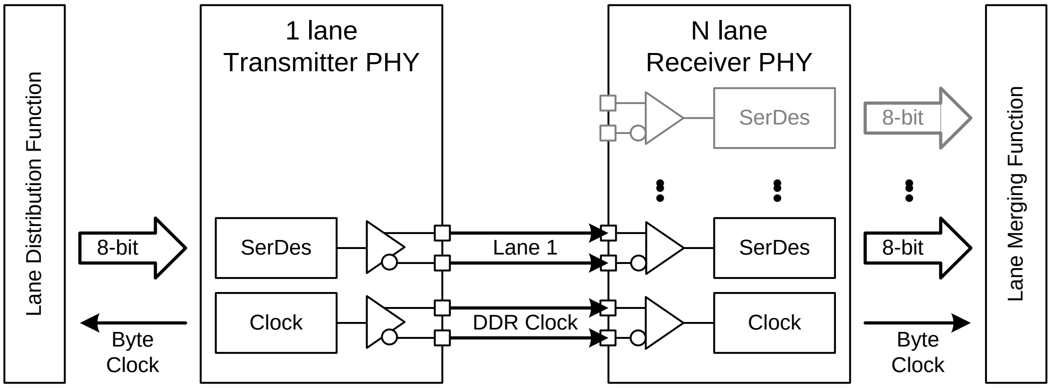


Figure 46 One Lane Transmitter and N-Lane Receiver Example for D-PHY

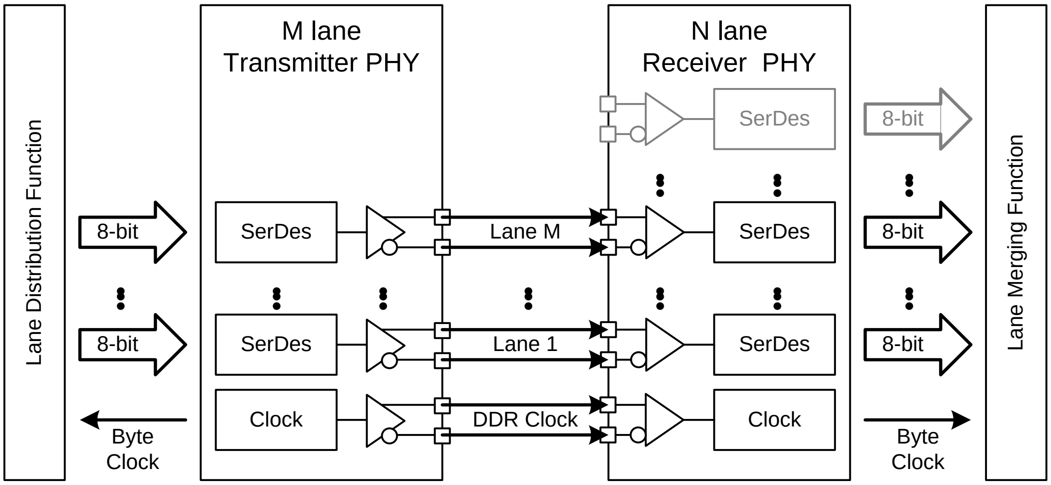


Figure 47 M-Lane Transmitter and N-Lane Receiver Example (M<N) for D-PHY

848

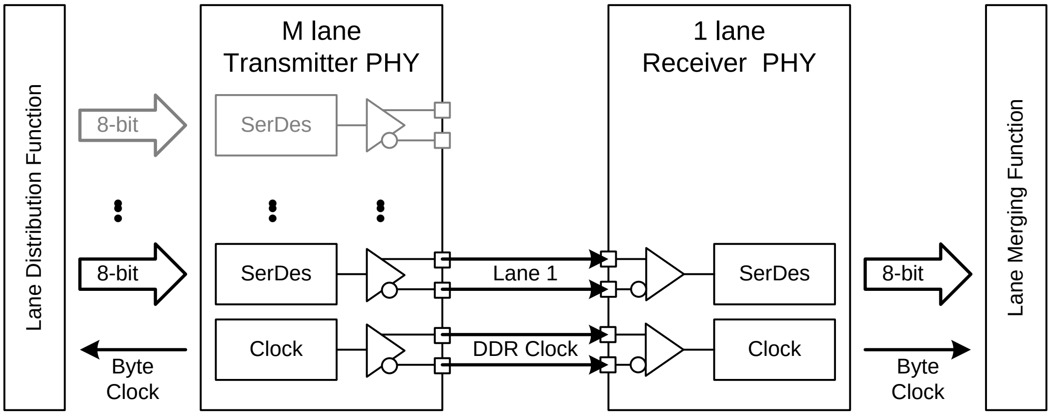


Figure 48 M-Lane Transmitter and One Lane Receiver Example for D-PHY

849

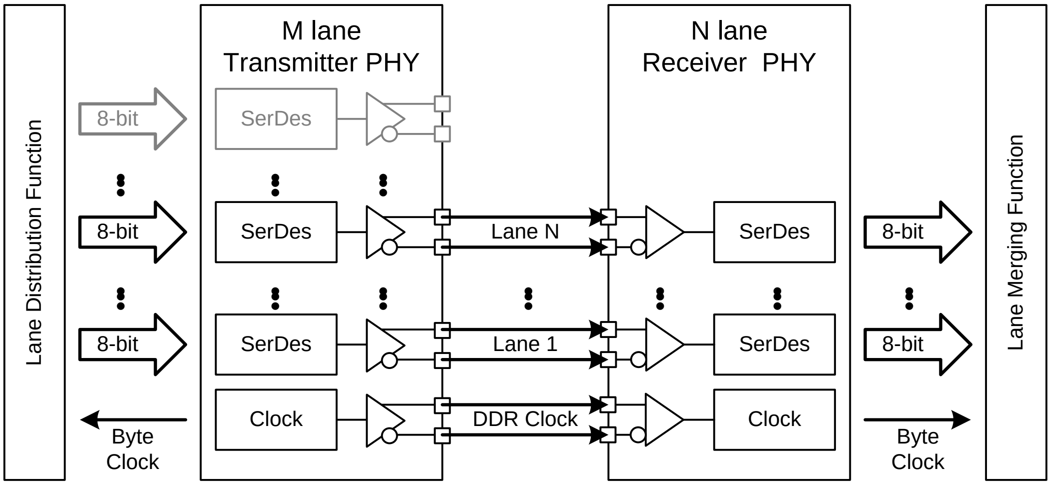


Figure 49 M-Lane Transmitter and N-Lane Receiver Example (N<M) for D-PHY

8.3.1 C-PHY Lane De-Skew

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

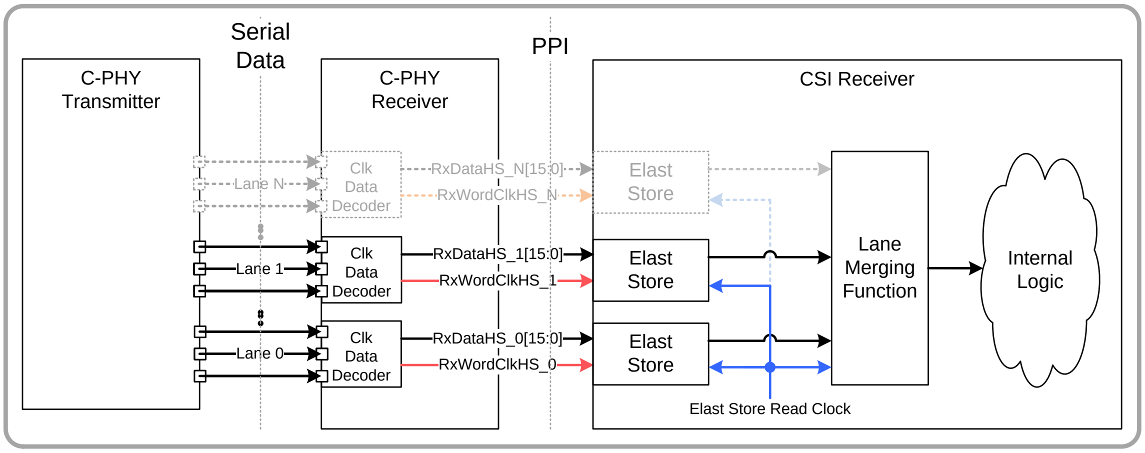


Figure 50 Example of Digital Logic to Align All RxDataHS

9 Low Level Protocol

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

Low Level Protocol Features:

- Transport of arbitrary data (Payload independent)
- 8-bit word size
- Support for up to sixteen interleaved virtual channels on the same D-PHY Link, or up to 32 interleaved virtual channels on the same C-PHY Link
- Special packets for frame start, frame end, line start and line end information
- Descriptor for the type, pixel depth and format of the Application Specific Payload data
- 16-bit Checksum Code for error detection.
- 6-bit Error Correction Code for error detection and correction (D-PHY physical layer only)

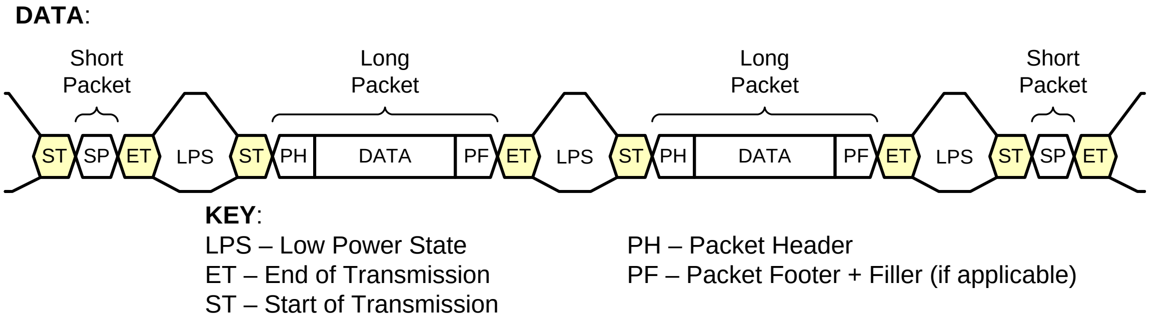


Figure 51 Low Level Protocol Packet Overview

9.1 Low Level Protocol Packet Format

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

9.1.1 Low Level Protocol Long Packet Format

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

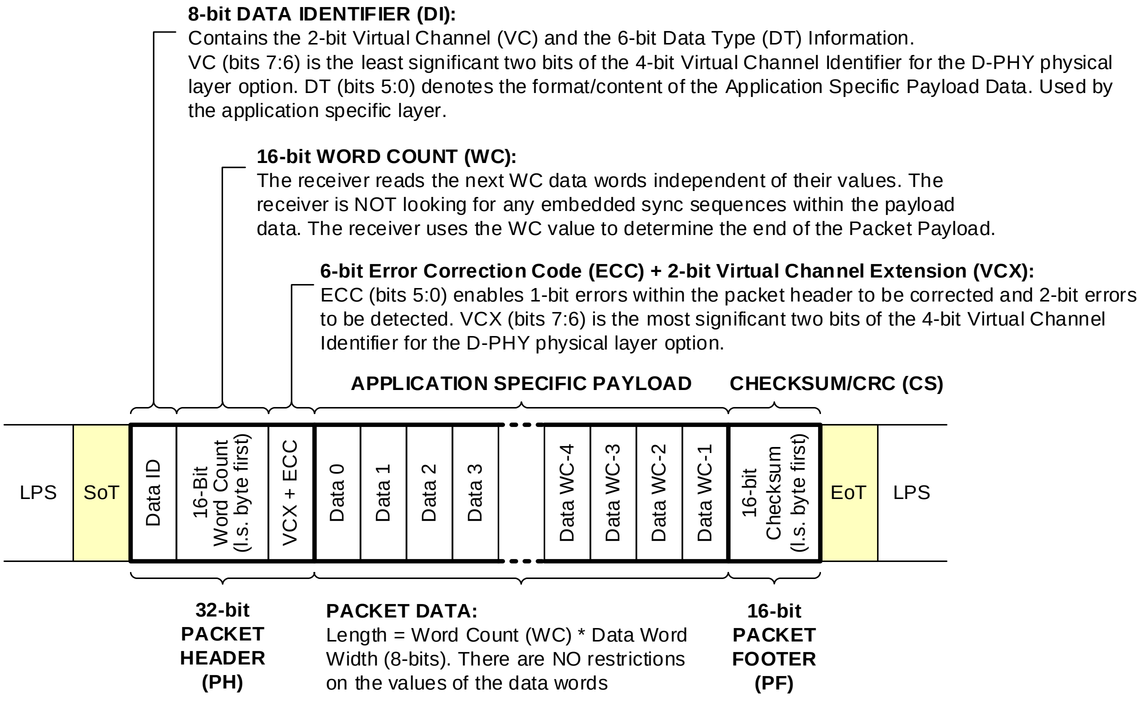

8-bit DATA IDENTIFIER (DI):
Contains the 2-bit Virtual Channel (VC) and the 6-bit Data Type (DT) Information.
VC (bits 7:6) is the least significant two bits of the 4-bit Virtual Channel Identifier for the D-PHY physical layer option. DT (bits 5:0) denotes the format/content of the Application Specific Payload Data. Used by the application specific layer.
16-bit WORD COUNT (WC):
The receiver reads the next WC data words independent of their values. The receiver is NOT looking for any embedded sync sequences within the payload data. The receiver uses the WC value to determine the end of the Packet Payload.
6-bit Error Correction Code (ECC) + 2-bit Virtual Channel Extension (VCX):
ECC (bits 5:0) enables 1-bit errors within the packet header to be corrected and 2-bit errors to be detected. VCX (bits 7:6) is the most significant two bits of the 4-bit Virtual Channel Identifier for the D-PHY physical layer option.
APPLICATION SPECIFIC PAYLOAD
CHECKSUM/CRC (CS)
LPS
SoT
Data ID
16-Bit Word Count (l.s. byte first)
VCX + ECC
Data 0
Data 1
Data 2
Data 3
Data WC-4
Data WC-3
Data WC-2
Data WC-1
16-bit Checksum (l.s. byte first)
EoT
LPS
32-bit PACKET HEADER (PH)
PACKET DATA:
Length = Word Count (WC) * Data Word Width (8-bits). There are NO restrictions on the values of the data words
16-bit PACKET FOOTER (PF)

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

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

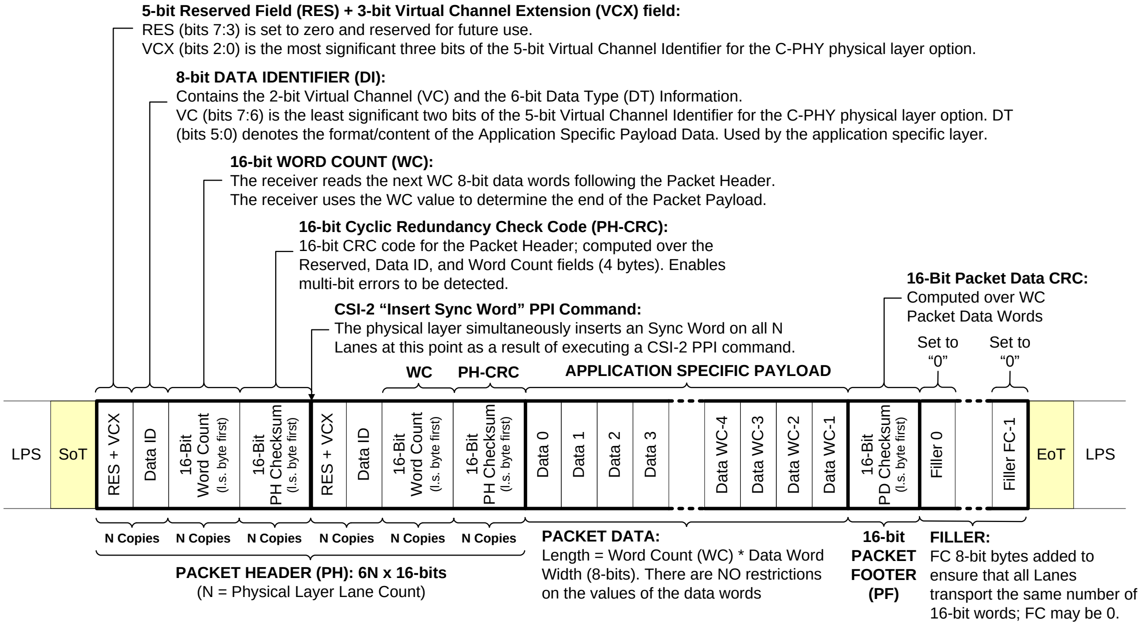

5-bit Reserved Field (RES) + 3-bit Virtual Channel Extension (VCX) field:
RES (bits 7:3) is set to zero and reserved for future use.
VCX (bits 2:0) is the most significant three bits of the 5-bit Virtual Channel Identifier for the C-PHY physical layer option.
8-bit DATA IDENTIFIER (DI):
Contains the 2-bit Virtual Channel (VC) and the 6-bit Data Type (DT) Information.
VC (bits 7:6) is the least significant two bits of the 5-bit Virtual Channel Identifier for the C-PHY physical layer option. DT (bits 5:0) denotes the format/content of the Application Specific Payload Data. Used by the application specific layer.
16-bit WORD COUNT (WC):
The receiver reads the next WC 8-bit data words following the Packet Header.
The receiver uses the WC value to determine the end of the Packet Payload.
16-bit Cyclic Redundancy Check Code (PH-CRC):
16-bit CRC code for the Packet Header; computed over the Reserved, Data ID, and Word Count fields (4 bytes). Enables multi-bit errors to be detected.
CSI-2 "Insert Sync Word" PPI Command:
The physical layer simultaneously inserts an Sync Word on all N Lanes at this point as a result of executing a CSI-2 PPI command.
16-Bit Packet Data CRC:
Computed over WC Packet Data Words
Set to "0"
Set to "0"
WC
PH-CRC
APPLICATION SPECIFIC PAYLOAD
LPS
SoT
RES + VCX
Data ID
16-Bit Word Count (l.s. byte first)
16-Bit PH Checksum (l.s. byte first)
RES + VCX
Data ID
16-Bit Word Count (l.s. byte first)
16-Bit PH Checksum (l.s. byte first)
Data 0
Data 1
Data 2
Data 3
Data WC-4
Data WC-3
Data WC-2
Data WC-1
16-Bit PD Checksum (l.s. byte first)
Filler 0
Filler FC-1
EoT
LPS
N Copies
N Copies
N Copies
N Copies
N Copies
N Copies
PACKET HEADER (PH): 6N x 16-bits
(N = Physical Layer Lane Count)
PACKET DATA:
Length = Word Count (WC) * Data Word Width (8-bits). There are NO restrictions on the values of the data words
16-bit PACKET FOOTER (PF)
FILLER:
FC 8-bit bytes added to ensure that all Lanes transport the same number of 16-bit words; FC may be 0.

Figure 53 Long Packet Structure for C-PHY Physical Layer Option

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

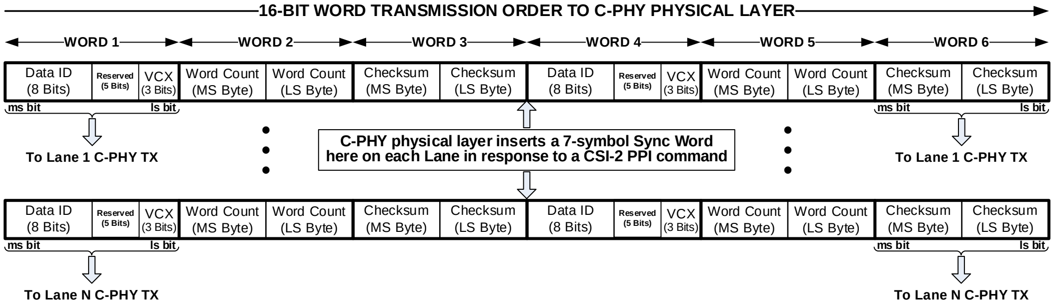


Figure 54 Packet Header Lane Distribution for C-PHY Physical Layer Option

For both physical layer options, the 8-bit Data Identifier field defines the 2-bit Virtual Channel (VC) and the Data Type for the application specific payload data. The Virtual Channel Extension (VCX) field is also 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 VCX fields comprise the 4- or 5-bit Virtual Channel Identifier field which determines the Virtual Channel number associated with the packet (see ***Section 9.3***).

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

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

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

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

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

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

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

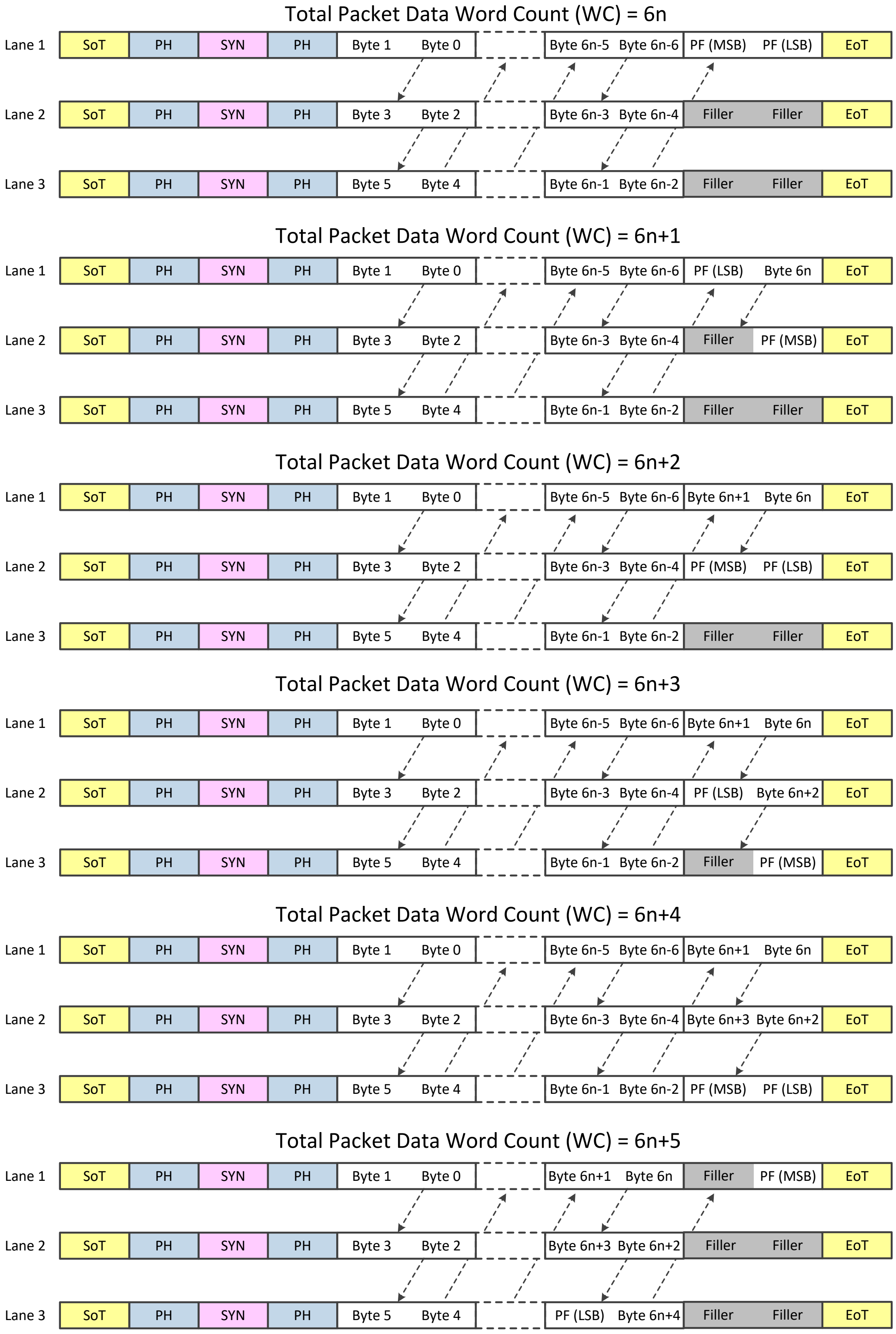
Figure 55 illustrates the Lane distribution of the minimal number of Filler bytes required for packets of various lengths transmitted over three C-PHY Lanes. The total number of Filler bytes required per packet ranges from 0 to 5, depending on the value of the Packet Data Word Count (WC). In general,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

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

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

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

After the EoT sequence the receiver begins looking for the next SoT sequence.


Total Packet Data Word Count (WC) = 6n
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 6n-5 Byte 6n-6 PF (MSB) PF (LSB) EoT
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 6n-3 Byte 6n-4 Filler Filler EoT
Lane 3 SoT PH SYN PH Byte 5 Byte 4 Byte 6n-1 Byte 6n-2 Filler Filler EoT
Total Packet Data Word Count (WC) = 6n+1
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 6n-5 Byte 6n-6 PF (LSB) Byte 6n EoT
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 6n-3 Byte 6n-4 Filler PF (MSB) EoT
Lane 3 SoT PH SYN PH Byte 5 Byte 4 Byte 6n-1 Byte 6n-2 Filler Filler EoT
Total Packet Data Word Count (WC) = 6n+2
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 6n-5 Byte 6n-6 Byte 6n+1 Byte 6n EoT
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 6n-3 Byte 6n-4 PF (MSB) PF (LSB) EoT
Lane 3 SoT PH SYN PH Byte 5 Byte 4 Byte 6n-1 Byte 6n-2 Filler Filler EoT
Total Packet Data Word Count (WC) = 6n+3
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 6n-5 Byte 6n-6 Byte 6n+1 Byte 6n EoT
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 6n-3 Byte 6n-4 PF (LSB) Byte 6n+2 EoT
Lane 3 SoT PH SYN PH Byte 5 Byte 4 Byte 6n-1 Byte 6n-2 Filler PF (MSB) EoT
Total Packet Data Word Count (WC) = 6n+4
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 6n-5 Byte 6n-6 Byte 6n+1 Byte 6n EoT
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 6n-3 Byte 6n-4 Byte 6n+3 Byte 6n+2 EoT
Lane 3 SoT PH SYN PH Byte 5 Byte 4 Byte 6n-1 Byte 6n-2 PF (MSB) PF (LSB) EoT
Total Packet Data Word Count (WC) = 6n+5
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 6n+1 Byte 6n Filler PF (MSB) EoT
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 6n+3 Byte 6n+2 Filler Filler EoT
Lane 3 SoT PH SYN PH Byte 5 Byte 4 PF (LSB) Byte 6n+4 Filler Filler EoT

KEY:

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

Figure 55 Minimal Filler Byte Insertion Requirements for Three Lane C-PHY

9.1.2 Low Level Protocol Short Packet Format

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

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

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

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

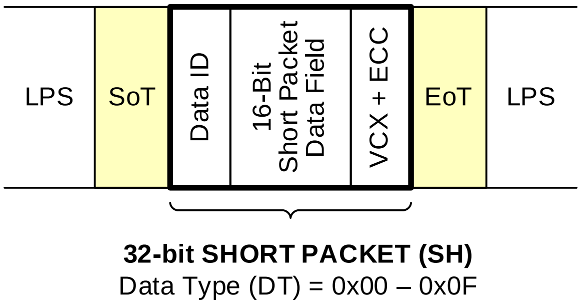
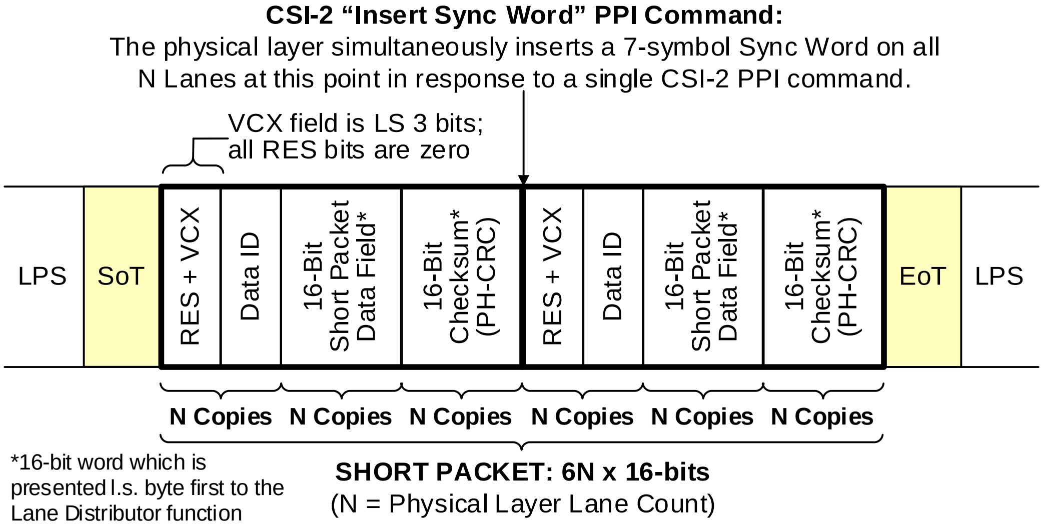


Figure 56 Short Packet Structure for D-PHY Physical Layer Option


LPS
SoT
RES + VCX
Data ID
16-Bit Short Packet Data Field*
16-Bit Checksum* (PH-CRC)
RES + VCX
Data ID
16-Bit Short Packet Data Field*
16-Bit Checksum* (PH-CRC)
EoT
LPS
N Copies
N Copies
N Copies
N Copies
N Copies
N Copies

CSI-2 "Insert Sync Word" PPI Command:
The physical layer simultaneously inserts a 7-symbol Sync Word on all N Lanes at this point in response to a single CSI-2 PPI command.

VCX field is LS 3 bits;
all RES bits are zero

*16-bit word which is presented l.s. byte first to the Lane Distributor function

SHORT PACKET: 6N x 16-bits
(N = Physical Layer Lane Count)

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

9.2 Data Identifier (DI)

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

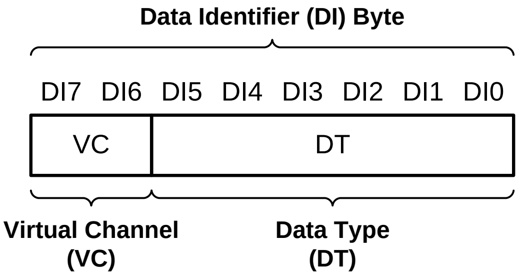


Figure 58 Data Identifier Byte

9.3 Virtual Channel Identifier

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

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

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

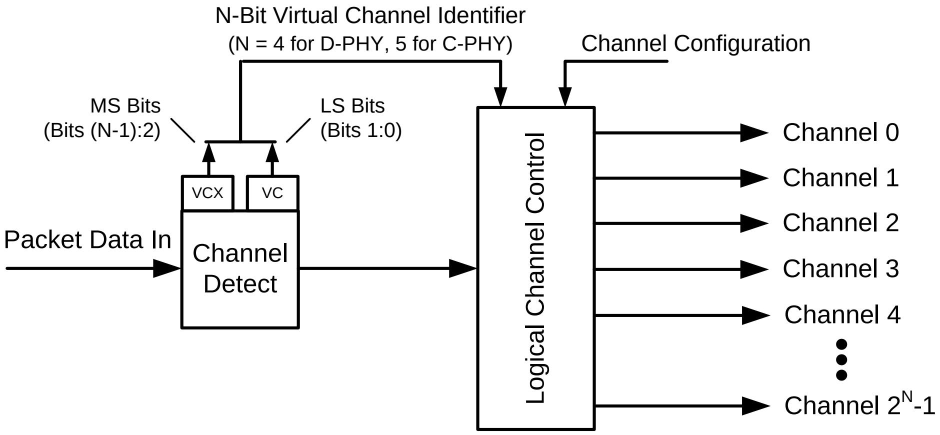


Figure 59 Logical Channel Block Diagram (Receiver)

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

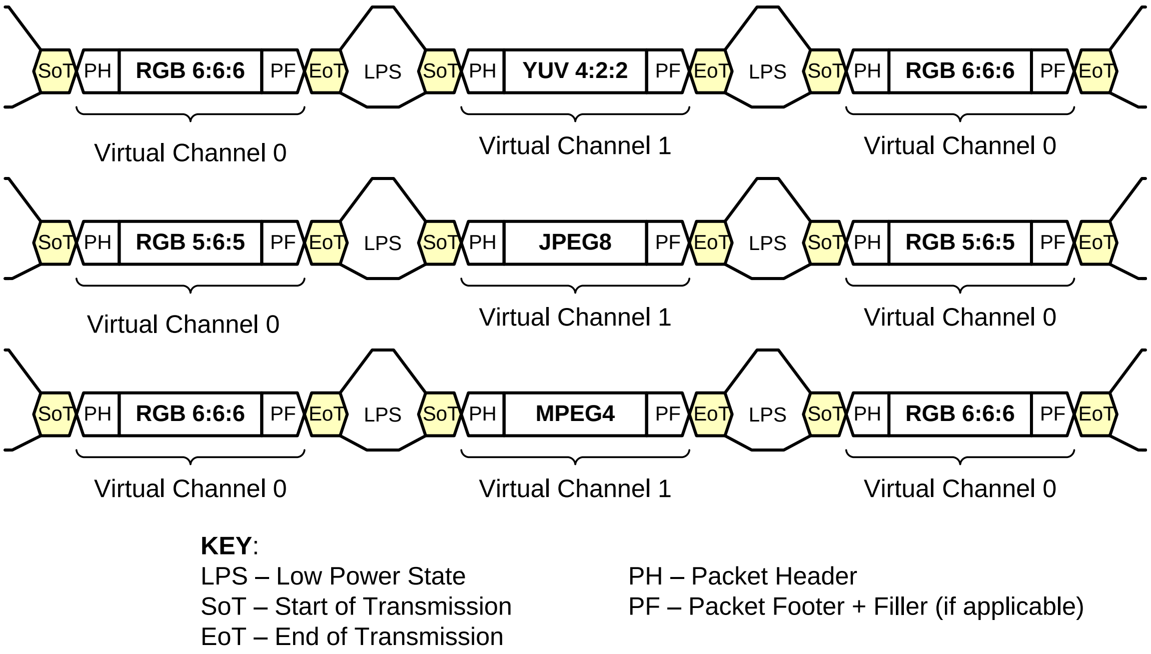


Figure 60 Interleaved Video Data Streams Examples

9.4 Data Type (DT)

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

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

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

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

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

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

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]) to D[15:8], the Word Count MS Byte (WC[15:8]) to D[23:16], and the Virtual Channel Extension (VCX) field to D[25:24]. This mapping is shown in ***Figure 61***, which also serves as an ECC calculation example.

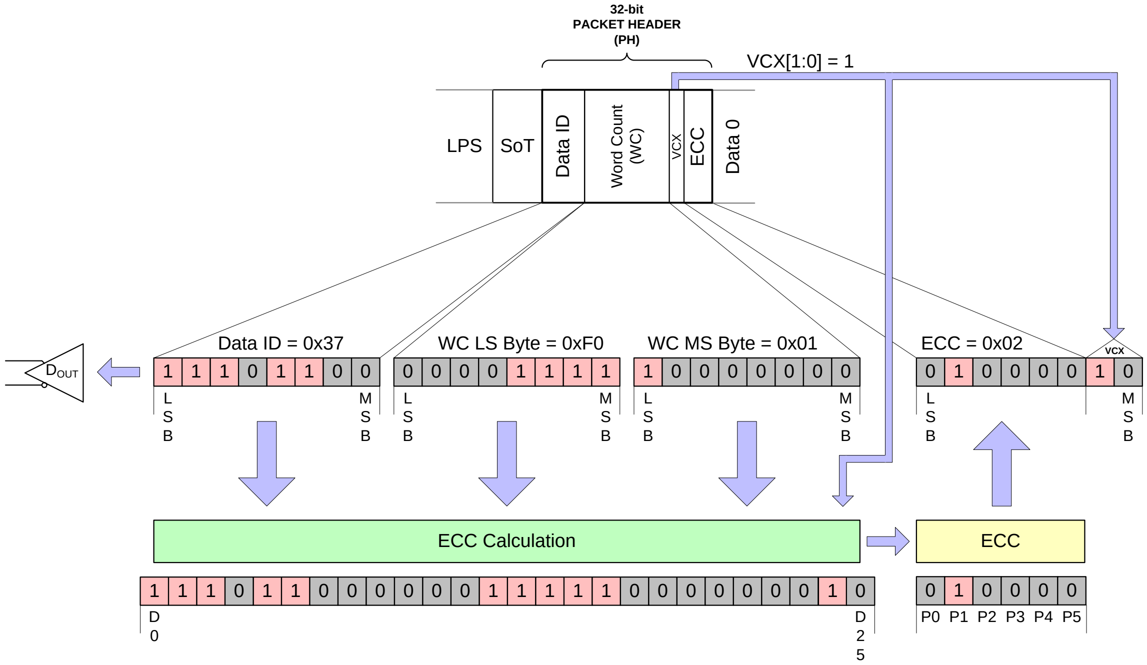


Figure 61 26-bit ECC Generation Example

9.5.1 General Hamming Code Applied to Packet Header

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

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

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

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

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

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

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

If all elements of s are zero, the code word was received correctly. If s contains non-zero elements, then at least one error is present. If a single bit error is encountered then the syndrome s is one of the elements of **H** 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 syndrome will be one of the elements on **I**, else it will be the data bit identified by the position of the syndrome in $\mathbf{A}^\mathbf{T}$.

9.5.2 Hamming-Modified Code

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

Table 11 ECC Syndrome Association Matrix

	d2d1d0							
d5d4d3	**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

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

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

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

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

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

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

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

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

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

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

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

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

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

This is an informative section.

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

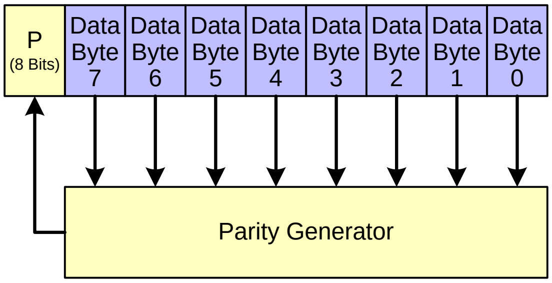


Figure 62 64-bit ECC Generation on TX Side

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

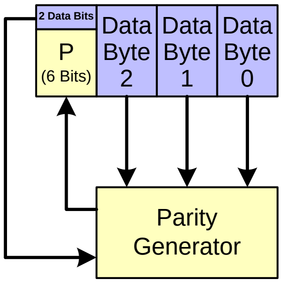


Figure 63 26-bit ECC Generation on TX Side

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

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

For backwards-compatibility, transmitters conforming to this CSI-2 Specification version should always set Packet Header bits 24 and 25 (the VCX field) to zero in any packets sent to receivers conforming to 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.

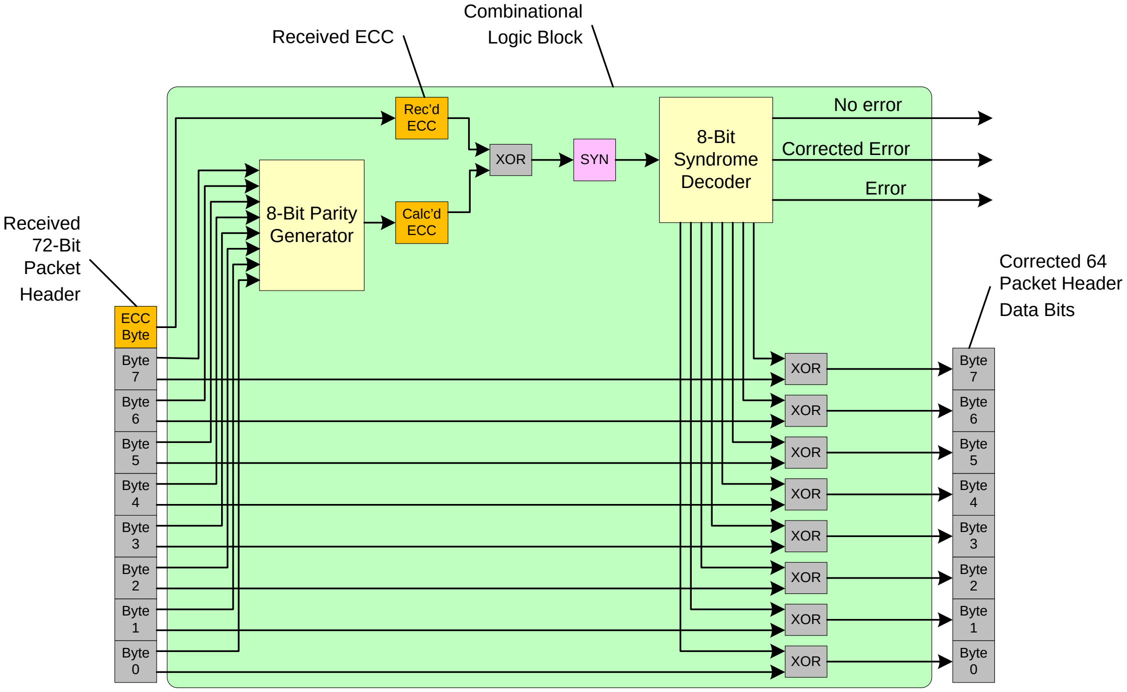


Figure 64 64-bit ECC on RX Side Including Error Correction

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.

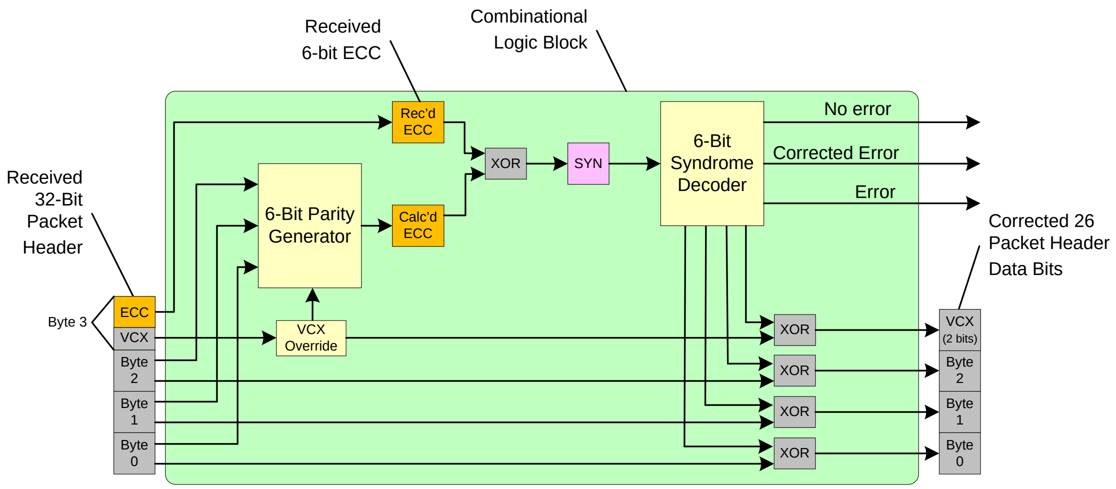


Figure 65 26-bit ECC on RX Side Including Error Correction

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***.

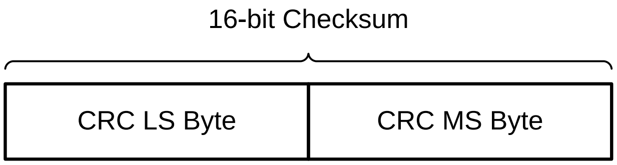


Figure 66 Checksum Transmission Byte Order

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.

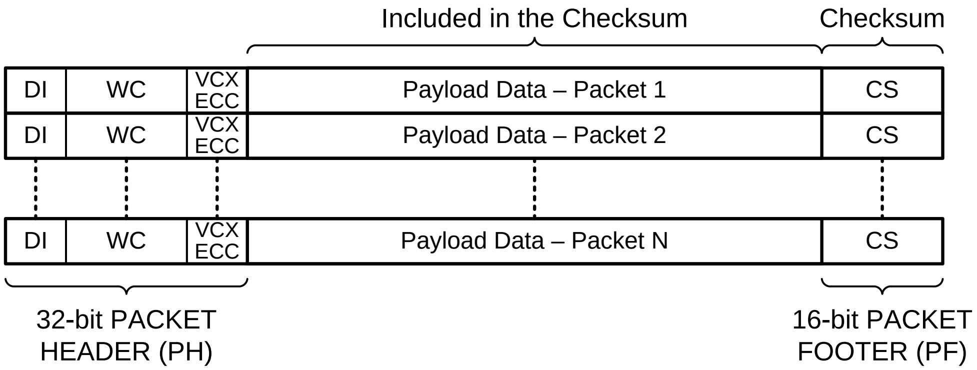
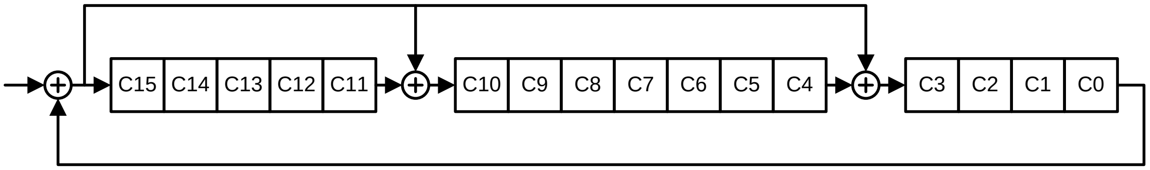


Figure 67 Checksum Generation for Long Packet Payload Data

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);
}
```

Figure 69 16-bit CRC Software Implementation Example

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

CRC computation examples:

```
Input Data Bytes:
FF 00 00 02 B9 DC F3 72 BB D4 B8 5A C8 75 C2 7C 81 F8 05 DF FF 00 00 01
Checksum LS byte and MS byte:
F0 00

Input Data Bytes:
FF 00 00 00 1E F0 1E C7 4F 82 78 C5 82 E0 8C 70 D2 3C 78 E9 FF 00 00 01
Checksum LS byte and MS byte:
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.

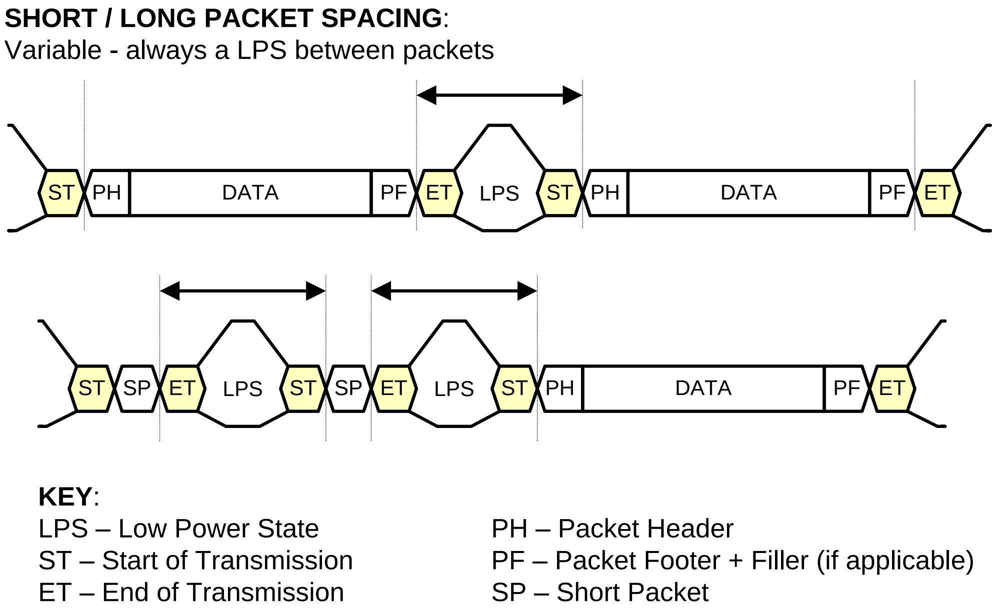


Figure 70 Packet Spacing

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.

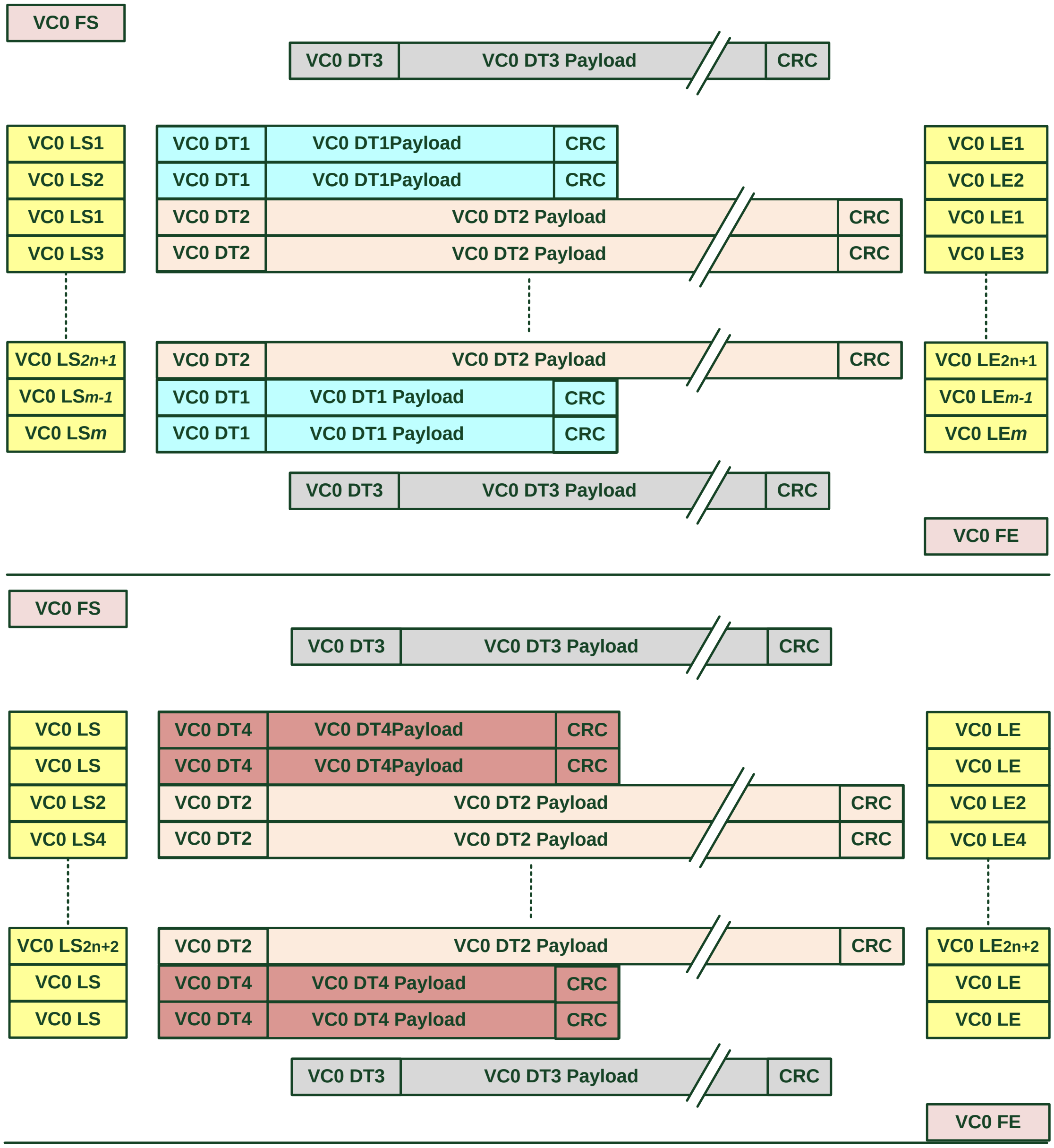
Or:

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

Table 14 lists the Generic Short Packet Data Types.

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

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

9.10 Packet Spacing Examples Using the Low Power State

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

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

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

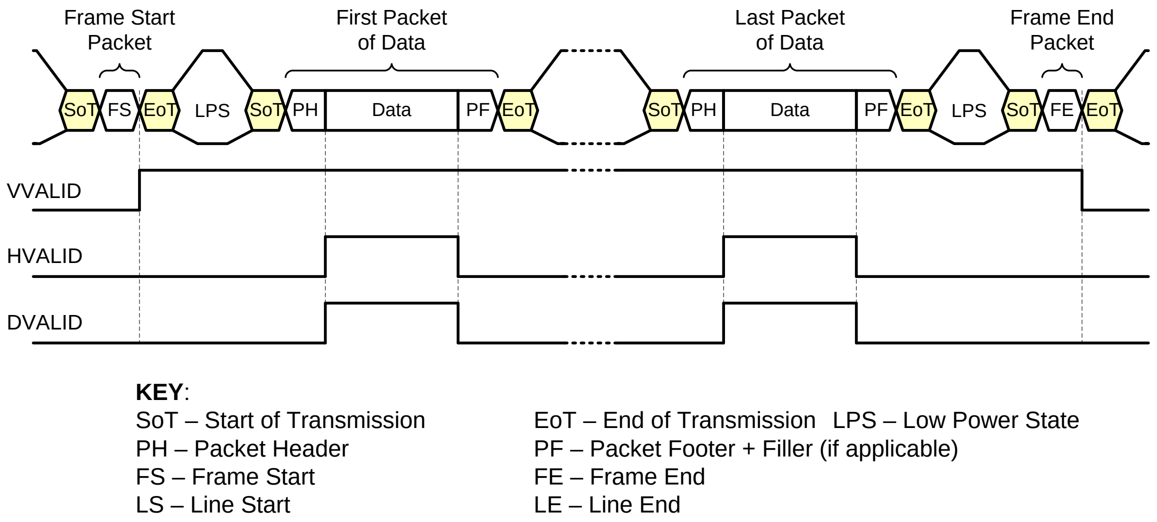
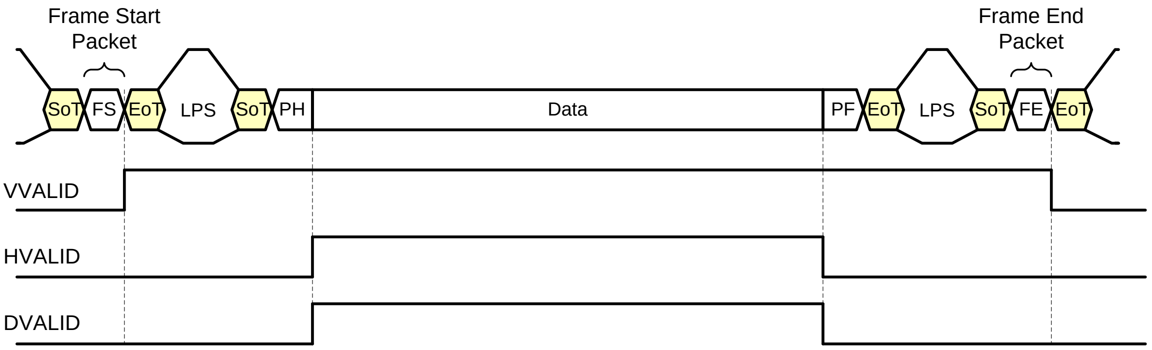
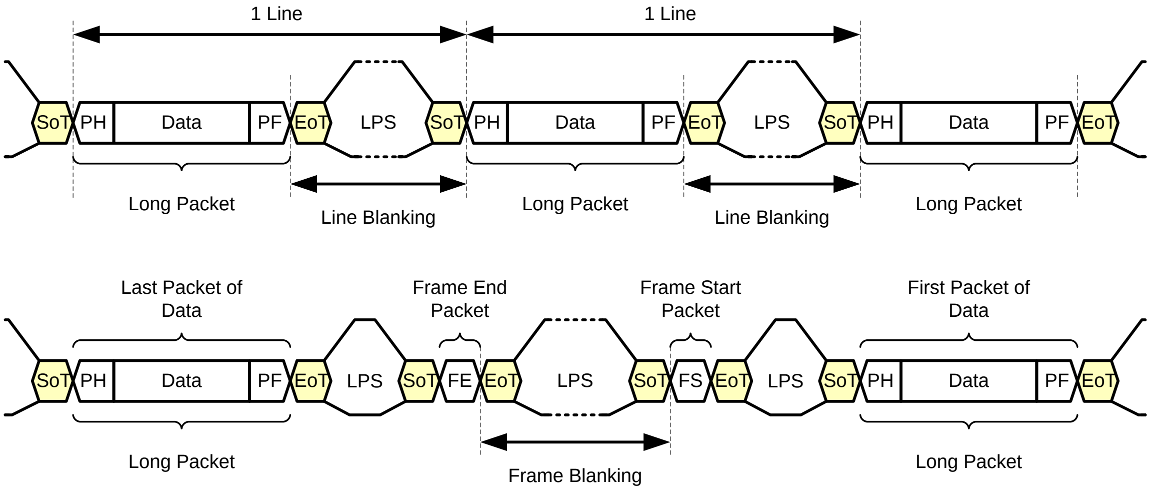


Figure 72 Multiple Packet Example



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.

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

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

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

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

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

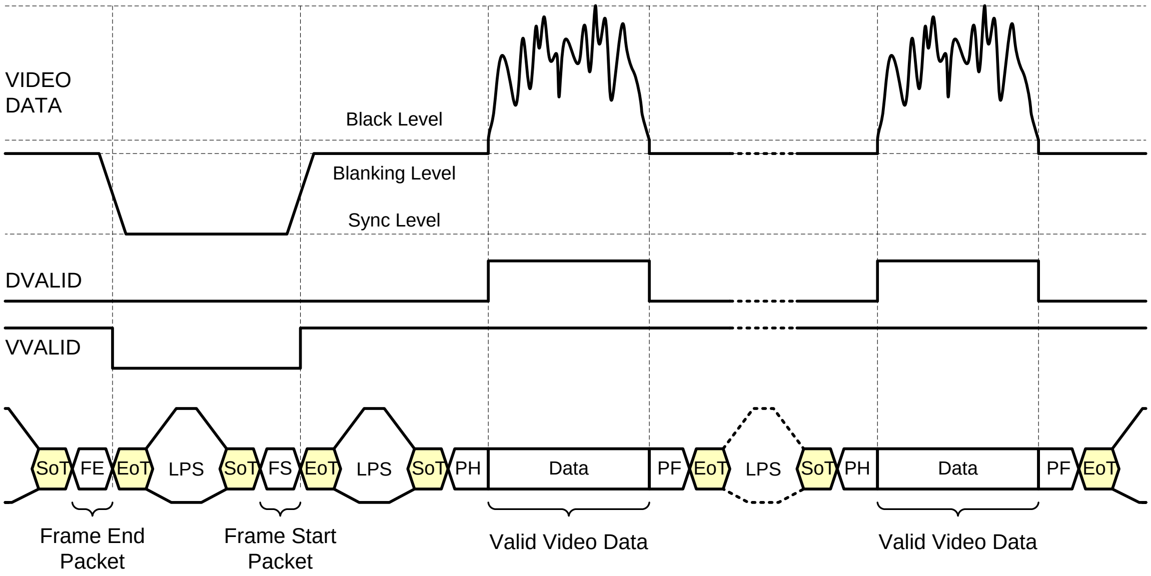
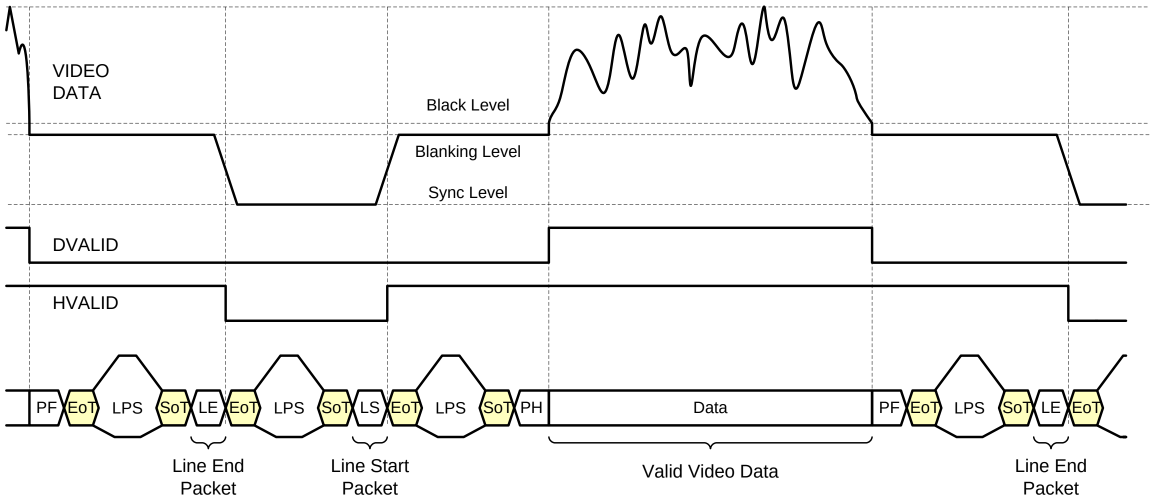


Figure 75 Vertical Sync Example



1238

Figure 76 Horizontal Sync Example

9.11 Latency Reduction and Transport Efficiency (LRTE)

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

LRTE has two parts, further detailed in this Section:

- Interpacket Latency Reduction (ILR)
- Enhanced Transport Efficiency

9.11.1 Interpacket Latency Reduction (ILR)

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

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

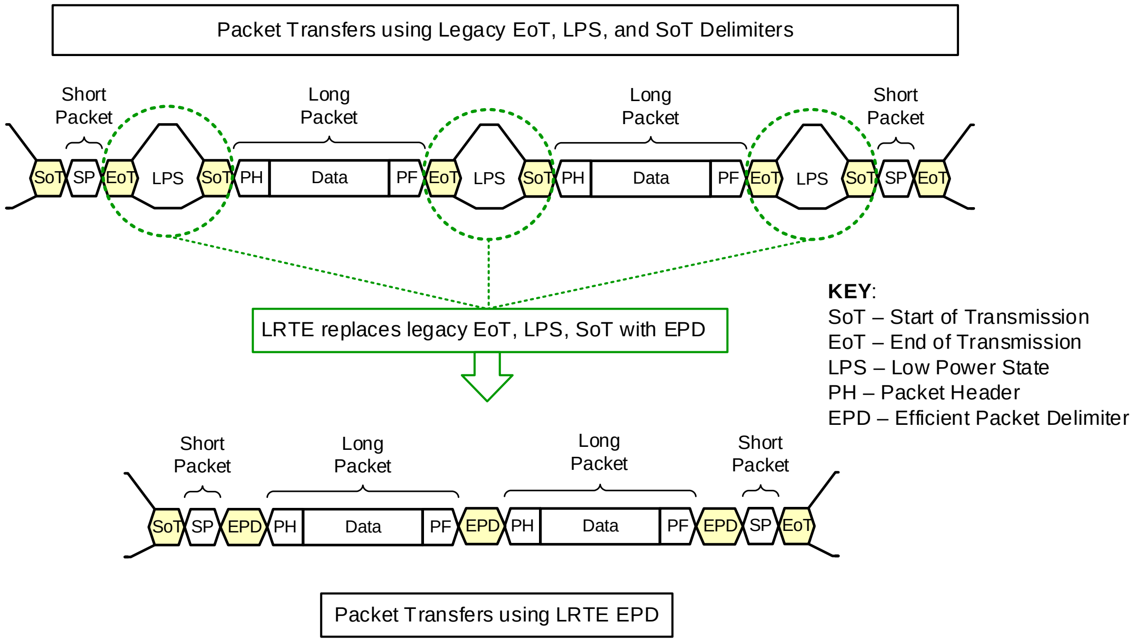


Figure 77 Interpacket Latency Reduction Using LRTE EPD

9.11.1.1 EPD for C-PHY Physical Layer Option

The EPD for the C-PHY physical layer option uses one or more instances of the PHY-generated and PHY-consumed 7-UI Sync Word for the Packet Delimiter Quick (PDQ) signaling. 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. An image sensor should reuse “TxSendSyncHS” at the PPI in order to generate the PDQ control code by the C-PHY transmitter. Upon reception of the PDQ control code by the C-PHY receiver, an application processor should reuse “RxSyncHS” at the PPI in order to notify the CSI-2 protocol layer. The duration of the 7-UI PDQ control code is directly proportional to the C-PHY Symbol rate.

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

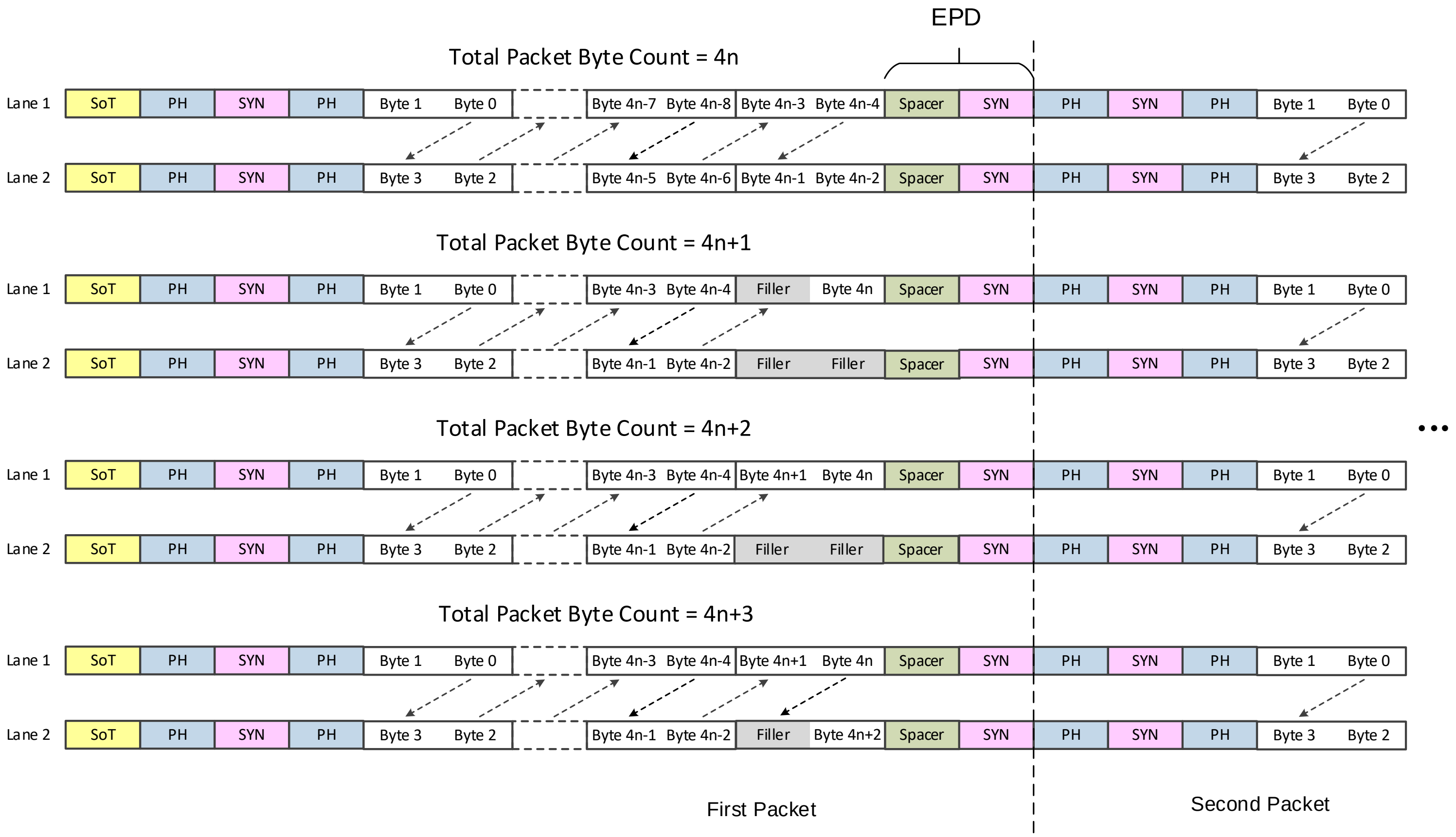
The image sensor (transmitter) shall include the following two 16-bit registers, in order to facilitate the optimal interpacket latency for imaging applications:

1. **TX_REG_CSI_EPD_EN_SSP** (EPD Enable and Short Packet Spacer) **Register**
   - The MS bit of this register shall be used to enable EPD with 7-UI PDQ (Sync Word) insertion between two CSI-2 packets and optional Spacer insertions for Short Packets and Long Packets.
     - 1’b0: C-PHY legacy EoT, LPS, SoT Packet Delimiter
     - 1’b1: C-PHY EPD (Efficient Packet Delimiter)
   - The remaining 15 bits of this register (bits [14:0]) shall be used to generate up to 32,767 Spacer insertions per Lane following CSI-2 Short Packets.
2. **TX_REG_CSI_EPD_OP_SLP** (Long Packet Spacer) **Register**
   - The MS bit of this register is reserved for future use.
   - The remaining 15 bits of this register (bits [14:0]) shall be used to generate up to 32,767 Spacer insertions per Lane following CSI-2 Long Packets.

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

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

The transmitter shall support at least one non-zero value of the Spacer insertion count field in each of the **TX_REG_CSI_EPD_EN_SSP** and **TX_REG_CSI_EPD_OP_SLP** registers.


EPD
Total Packet Byte Count = 4n
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 4n-7 Byte 4n-8 Byte 4n-3 Byte 4n-4 Spacer SYN PH SYN PH Byte 1 Byte 0
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 4n-5 Byte 4n-6 Byte 4n-1 Byte 4n-2 Spacer SYN PH SYN PH Byte 3 Byte 2
Total Packet Byte Count = 4n+1
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 4n-3 Byte 4n-4 Filler Byte 4n Spacer SYN PH SYN PH Byte 1 Byte 0
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 4n-1 Byte 4n-2 Filler Filler Spacer SYN PH SYN PH Byte 3 Byte 2
Total Packet Byte Count = 4n+2
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 4n-3 Byte 4n-4 Byte 4n+1 Byte 4n Spacer SYN PH SYN PH Byte 1 Byte 0
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 4n-1 Byte 4n-2 Filler Filler Spacer SYN PH SYN PH Byte 3 Byte 2
Total Packet Byte Count = 4n+3
Lane 1 SoT PH SYN PH Byte 1 Byte 0 Byte 4n-3 Byte 4n-4 Byte 4n+1 Byte 4n Spacer SYN PH SYN PH Byte 1 Byte 0
Lane 2 SoT PH SYN PH Byte 3 Byte 2 Byte 4n-1 Byte 4n-2 Filler Byte 4n+2 Spacer SYN PH SYN PH Byte 3 Byte 2
•••
First Packet
Second Packet

KEY:

SoT – Start of Transmission

EoT – End of Transmission

EPD – Efficient Packet Delimiter contains optional Spacer word insertion followed by a mandatory PDQ (Sync Word). An EPD consisting of single Spacer followed by one PDQ shown for illustration.

SYN – Sync Word

PH – Packet Header

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.

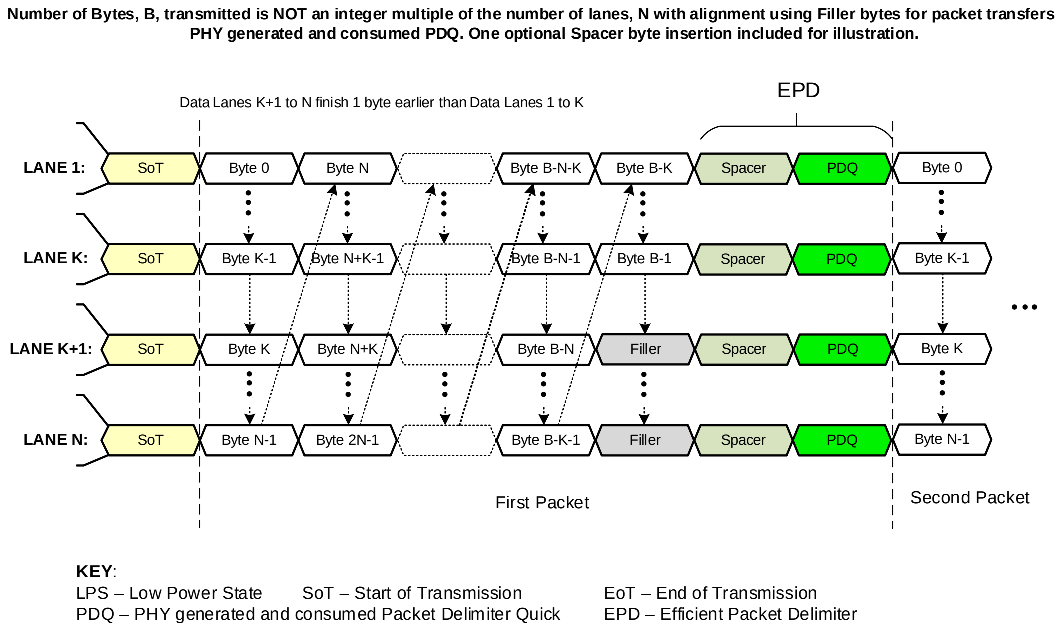

Number of Bytes, B, transmitted is NOT an integer multiple of the number of lanes, N with alignment using Filler bytes for packet transfers using PHY generated and consumed PDQ. One optional Spacer byte insertion included for illustration.
Data Lanes K+1 to N finish 1 byte earlier than Data Lanes 1 to K
EPD
LANE 1:
SoT
Byte 0
Byte N
Byte B-N-K
Byte B-K
Spacer
PDQ
Byte 0
LANE K:
SoT
Byte K-1
Byte N+K-1
Byte B-N-1
Byte B-1
Spacer
PDQ
Byte K-1
LANE K+1:
SoT
Byte K
Byte N+K
Byte B-N
Filler
Spacer
PDQ
Byte K
LANE N:
SoT
Byte N-1
Byte 2N-1
Byte B-K-1
Filler
Spacer
PDQ
Byte N-1
First Packet
Second Packet
KEY:
LPS – Low Power State
SoT – Start of Transmission
EoT – End of Transmission
PDQ – PHY generated and consumed Packet Delimiter Quick
EPD – Efficient Packet Delimiter

Figure 79 Example of LRTE EPD for CSI-2 Over D-PHY – Option 1

9.11.1.2.2 D-PHY EPD Option 2

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

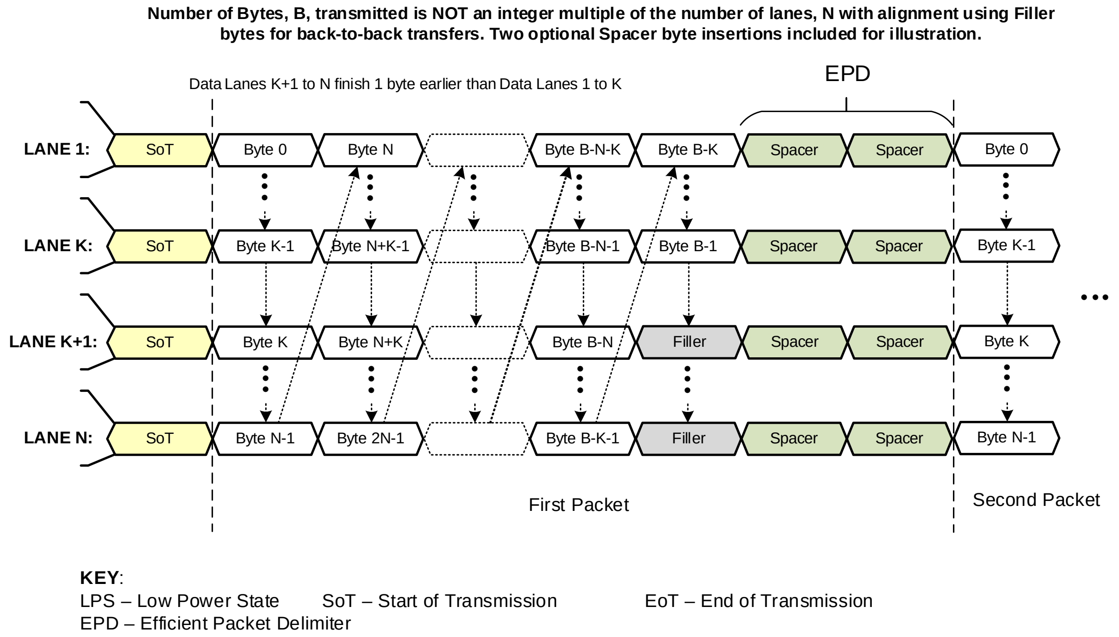

Number of Bytes, B, transmitted is NOT an integer multiple of the number of lanes, N with alignment using Filler bytes for back-to-back transfers. Two optional Spacer byte insertions included for illustration.
Data Lanes K+1 to N finish 1 byte earlier than Data Lanes 1 to K
EPD
LANE 1:
SoT
Byte 0
Byte N
Byte B-N-K
Byte B-K
Spacer
Spacer
Byte 0
LANE K:
SoT
Byte K-1
Byte N+K-1
Byte B-N-1
Byte B-1
Spacer
Spacer
Byte K-1
LANE K+1:
SoT
Byte K
Byte N+K
Byte B-N
Filler
Spacer
Spacer
Byte K
LANE N:
SoT
Byte N-1
Byte 2N-1
Byte B-K-1
Filler
Spacer
Spacer
Byte N-1
First Packet
Second Packet
KEY:
LPS – Low Power State
SoT – Start of Transmission
EoT – End of Transmission
EPD – Efficient Packet Delimiter

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)

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

1. **TX_REG_CSI_EPD_EN_SSP** (EPD Enable and Short Packet Spacer) **Register**
   - The MS bit of this register shall be used to enable EPD insertion between two CSI-2 packets.
     - 1'b1: Enable D-PHY EPD (Efficient Packet Delimiter)
   - If D-PHY EPD is enabled, then the remaining fifteen bits of this register (bits [14:0]) shall be used to generate up to 32,767 Spacer insertions per Lane following CSI-2 Short Packets. These Spacer insertions for CSI-2 Short Packets apply to both D-PHY EPD options.
2. **TX_REG_CSI_EPD_OP_SLP** (EPD Option and Long Packet Spacer) **Register**
   - The MS bit of this register shall be used to select the D-PHY EPD option.
     - 1'b0: D-PHY EPD Option 1
     - 1'b1: D-PHY EPD Option 2
   - If D-PHY EPD is enabled, then the remaining fifteen bits of this register (bits [14:0]) shall be used to generate up to 32,767 optional Spacer insertions per Lane following CSI-2 Long Packets. These Spacer insertions for CSI-2 Long Packets apply to both D-PHY EPD options.

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

- A register value of 15’d0 produces no Spacer generation (zero Spacers inserted).
- A register value of 15’d5 generates 5 Spacers.
- The maximum register value of 15’d32,767 generates 32,767 Spacers.

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

9.11.2 Using ILR and Enhanced Transport Efficiency Together

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

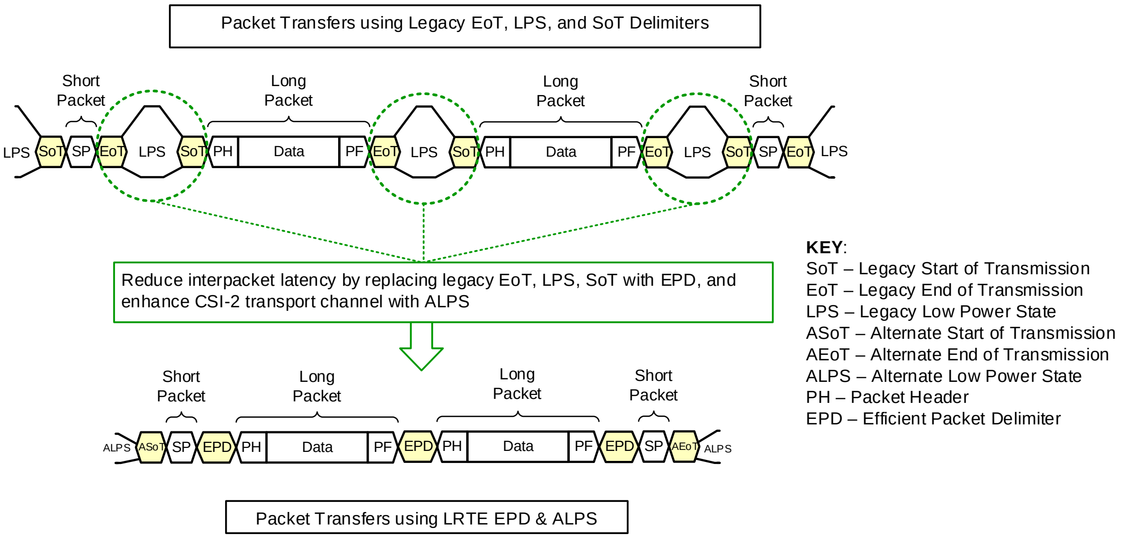

Packet Transfers using Legacy EoT, LPS, and SoT Delimiters
Short Packet
Long Packet
Long Packet
Short Packet
LPS
SoT
SP
EoT
LPS
SoT
PH
Data
PF
EoT
LPS
SoT
PH
Data
PF
EoT
LPS
SoT
SP
EoT
LPS
Reduce interpacket latency by replacing legacy EoT, LPS, SoT with EPD, and enhance CSI-2 transport channel with ALPS
Short Packet
Long Packet
Long Packet
Short Packet
ALPS
ASoT
SP
EPD
PH
Data
PF
EPD
PH
Data
PF
EPD
SP
AEoT
ALPS
Packet Transfers using LRTE EPD & ALPS
KEY:
SoT – Legacy Start of Transmission
EoT – Legacy End of Transmission
LPS – Legacy Low Power State
ASoT – Alternate Start of Transmission
AEoT – Alternate End of Transmission
ALPS – Alternate Low Power State
PH – Packet Header
EPD – Efficient Packet Delimiter

Figure 81 Using EPD and ALPS Together

9.11.3 LRTE Register Tables

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

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.

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

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

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

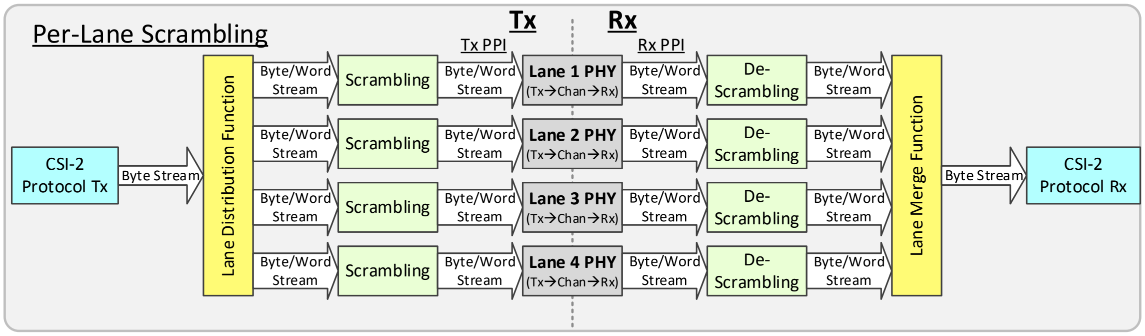


Figure 82 System Diagram Showing Per-Lane Scrambling

9.12.1 CSI-2 Scrambling for D-PHY

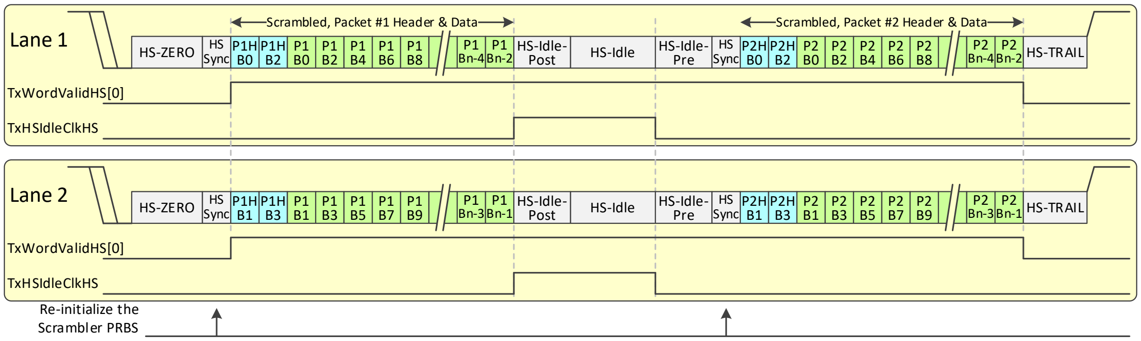
Figure 83 shows the format of a burst transmission of two packets over two Lanes when the D-PHY physical layer is used. After the Start of Transmission, HS-ZERO and HS-SYNC are transmitted, the Packet Header and data payload are distributed across the two Lanes.

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

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

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

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

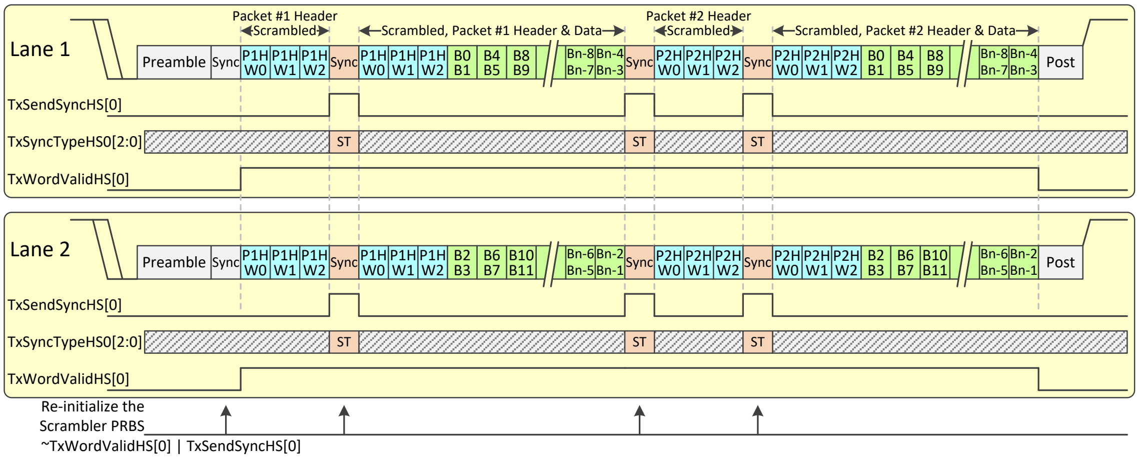


Note: The Packet Footer at the end of every packet is scrambled.

Figure 83 Example of Data Bursts in Two Lanes Using the D-PHY Physical Layer

9.12.2 CSI-2 Scrambling for C-PHY

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



Note: The Packet Footer at the end of every packet is scrambled.

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

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

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

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

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

Note:

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

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

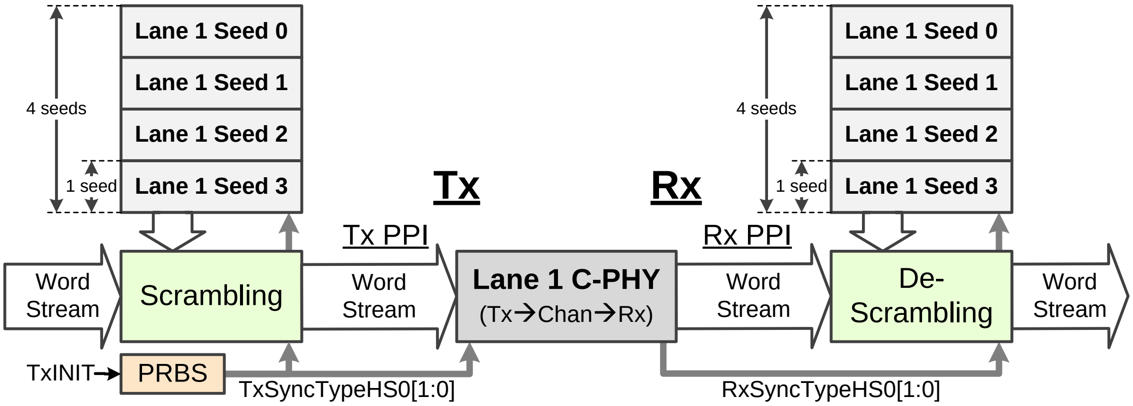


Figure 85 Generating Tx Sync Type as Seed Index (Single Lane View)

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

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

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

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

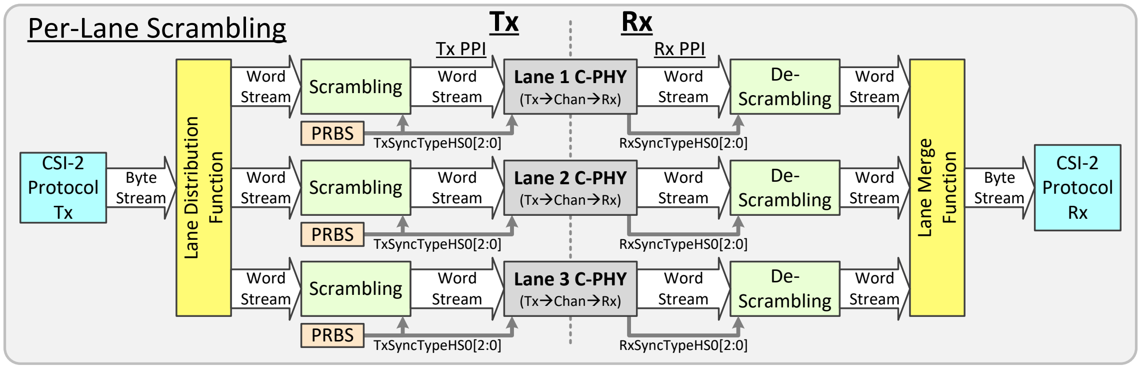


Figure 86 Generating Tx Sync Type Using the C-PHY Physical Layer

9.12.3 Scrambling Details

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

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

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

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

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

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`

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

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`

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

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

Scrambling shall be achieved by modulo-2 bit-wise addition (X-OR) of a sequence of eight bits G(x) with 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] ⊕ Q'[8];
- TxD[6] = PktD[6] ⊕ Q'[9];
- TxD[5] = PktD[5] ⊕ Q'[10];
- TxD[4] = PktD[4] ⊕ Q'[11];
- TxD[3] = PktD[3] ⊕ Q'[12];
- TxD[2] = PktD[2] ⊕ Q'[13];
- TxD[1] = PktD[1] ⊕ Q'[14];
- TxD[0] = PktD[0] ⊕ Q'[15];

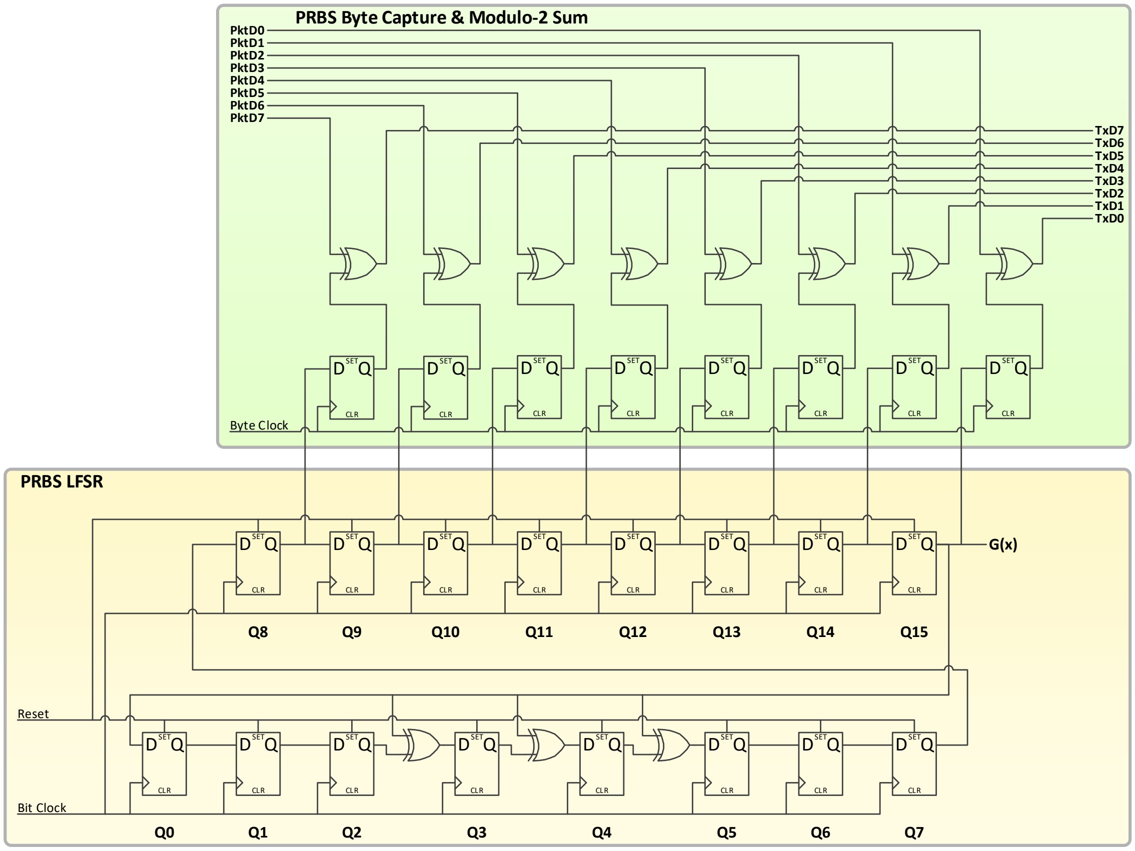


Figure 87 PRBS LFSR Serial Implementation Example

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

Table 22 shows the first ten PRBS Byte Outputs produced by the PRBS LFSR in Lane 2 when the D-PHY 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	0xxxxx	0xxxxx
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.

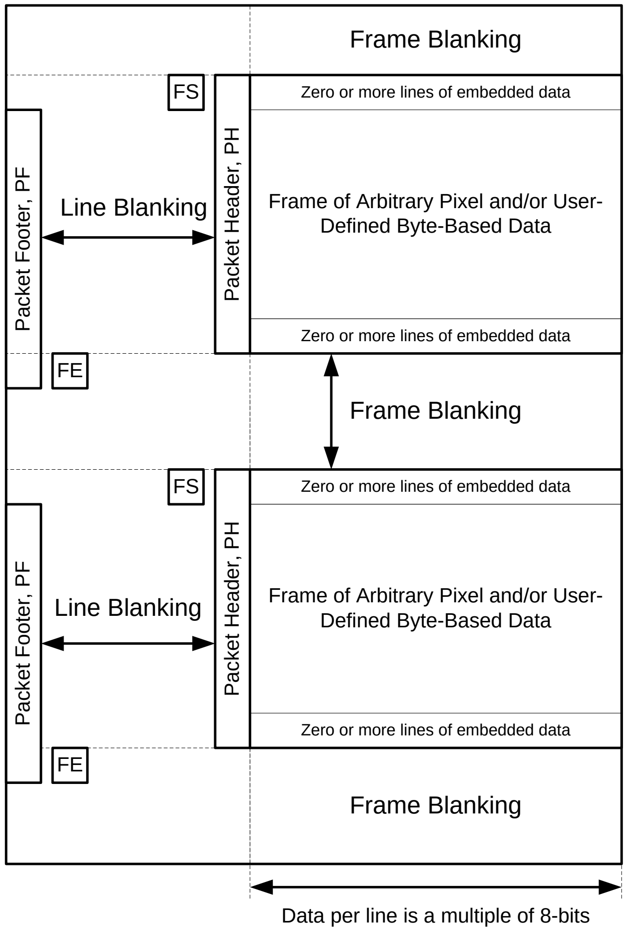
The total size of payload data within a long packet for all data types shall be a multiple of eight bits. However, it is also possible that a data type's payload data transmission format, as defined elsewhere in this Specification, imposes additional constraints on payload size. In order to meet these 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***


Frame Blanking
FS
Zero or more lines of embedded data
Packet Footer, PF
Line Blanking
Packet Header, PH
Frame of Arbitrary Pixel and/or User-Defined Byte-Based Data
Zero or more lines of embedded data
FE
Frame Blanking
FS
Zero or more lines of embedded data
Packet Footer, PF
Line Blanking
Packet Header, PH
Frame of Arbitrary Pixel and/or User-Defined Byte-Based Data
Zero or more lines of embedded data
FE
Frame Blanking
Data per line is a multiple of 8-bits

KEY:

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

Figure 88 General Frame Format Example

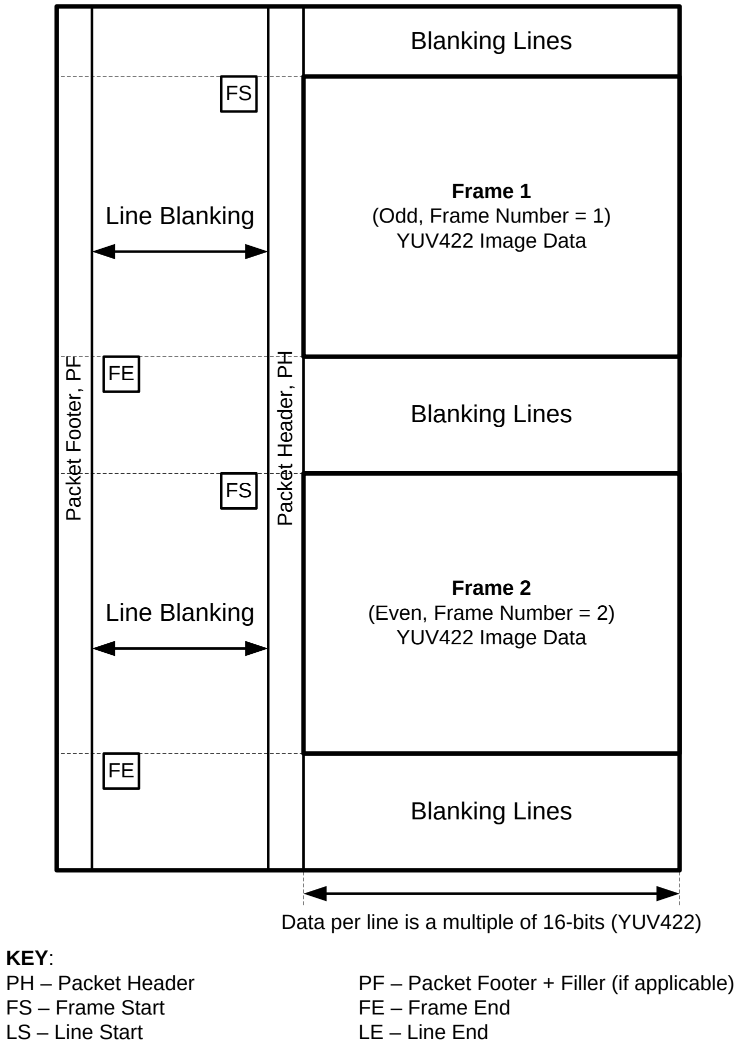
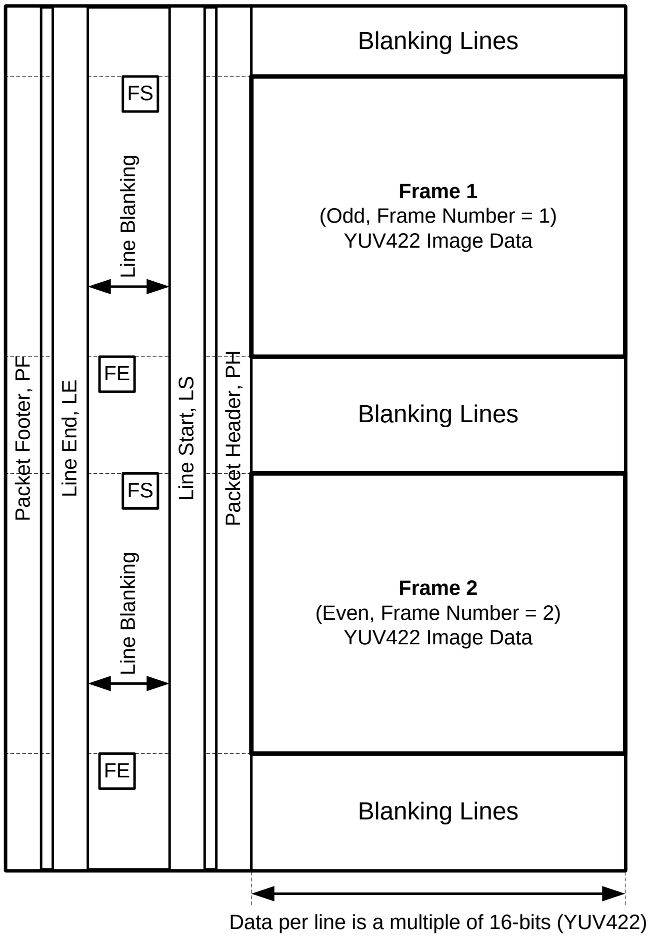


Figure 89 Digital Interlaced Video Example



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

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

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

- Data Type
- Virtual Channel Identifier

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

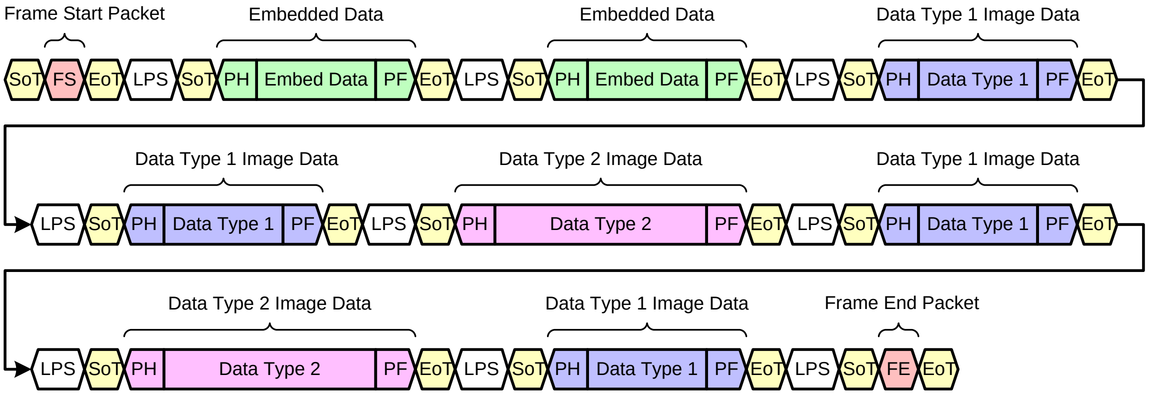
9.15.1 Data Type Interleaving

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

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

- For defined image data types – any non-reserved codes in the range 0x18 to 0x3F – only the single corresponding MIPI-defined packet payload data format shall be considered correct
- Reserved image data types – any reserved codes in the range 0x18 to 0x3F – shall not be used. No packet payload data format shall be considered correct for reserved image data types
- For generic long packet data types (codes 0x10 thru 0x17) and user-defined, byte-based (codes 0x30 – 0x37), any packet payload data format shall be considered correct
- Generic long packet data types (codes 0x10 thru 0x17) and user-defined, byte-based (codes 0x30 – 0x37), should not be used with packet payloads that meet any MIPI image data format definition
- Synchronization short packet data types (codes 0x00 thru 0x07) shall consist of only the header and shall not include payload data bytes
- Generic short packet data types (codes 0x08 thru 0x0F) shall consist of only the header and shall not include payload data bytes

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


Frame Start Packet
Embedded Data
Embedded Data
Data Type 1 Image Data
SoT FS EoT LPS SoT PH Embed Data PF EoT LPS SoT PH Embed Data PF EoT LPS SoT PH Data Type 1 PF EoT
Data Type 1 Image Data
Data Type 2 Image Data
Data Type 1 Image Data
LPS SoT PH Data Type 1 PF EoT LPS SoT PH Data Type 2 PF EoT LPS SoT PH Data Type 1 PF EoT
Data Type 2 Image Data
Data Type 1 Image Data
Frame End Packet
LPS SoT PH Data Type 2 PF EoT LPS SoT PH Data Type 1 PF EoT LPS SoT FE EoT

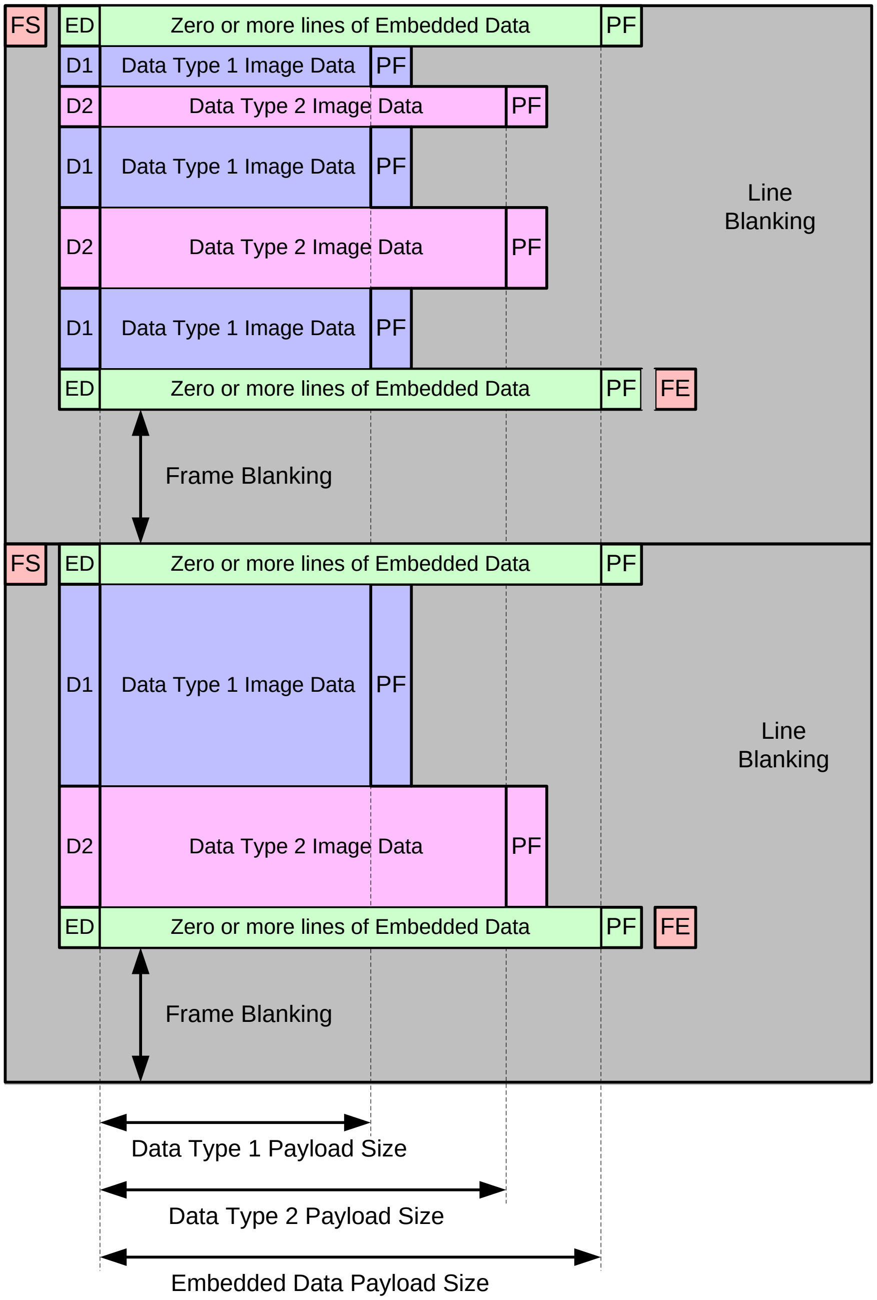
KEY:

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

Figure 91 Interleaved Data Transmission using Data Type Value

All of the packets within the same virtual channel, independent of the Data Type value, share the same frame start/end and line start/end synchronization information. By definition, all of the packets, independent of data type, between a Frame Start and a Frame End packet within the same virtual channel belong to the same frame.

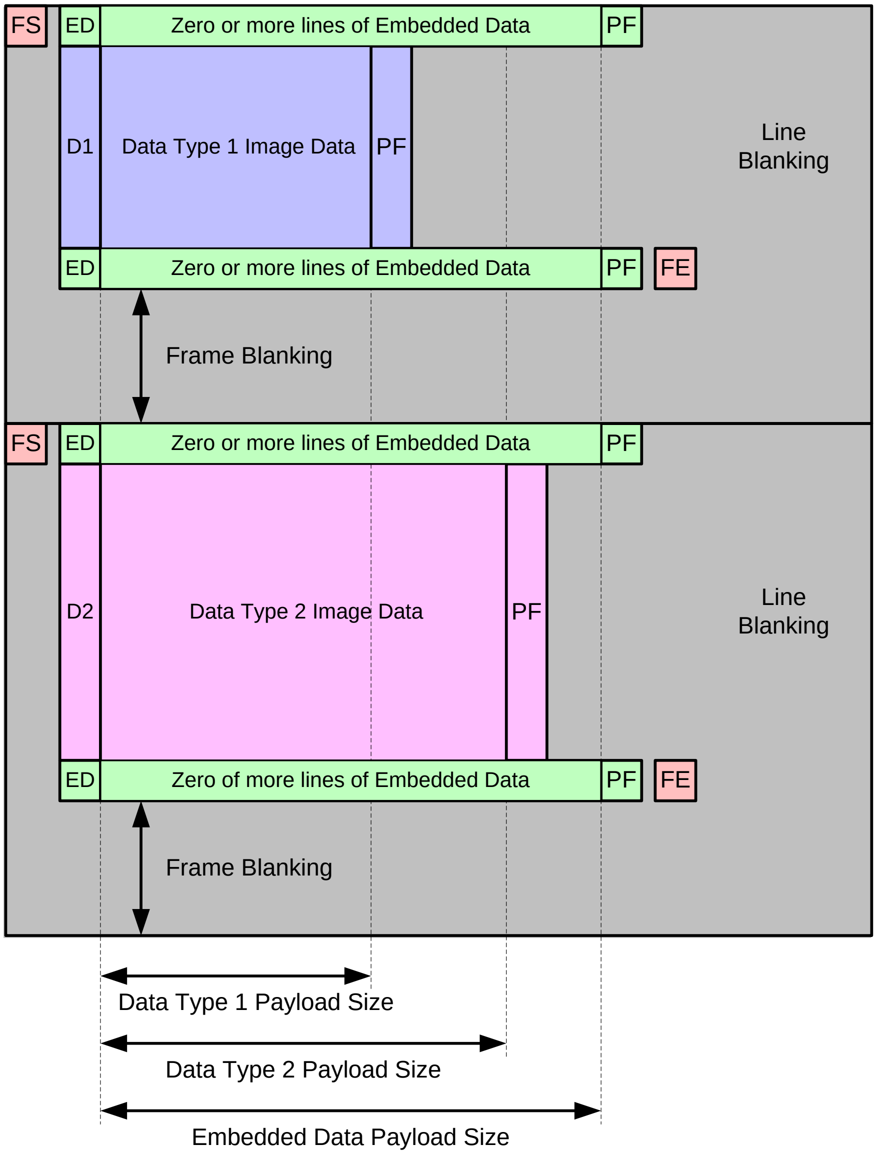
Packets of different data types may be interleaved at either the packet level as illustrated in ***Figure 92*** or the frame level as illustrated in ***Figure 93***. Data formats are defined in ***Section 11***.



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)

Figure 92 Packet 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)

Figure 93 Frame Level Interleaved Data Transmission

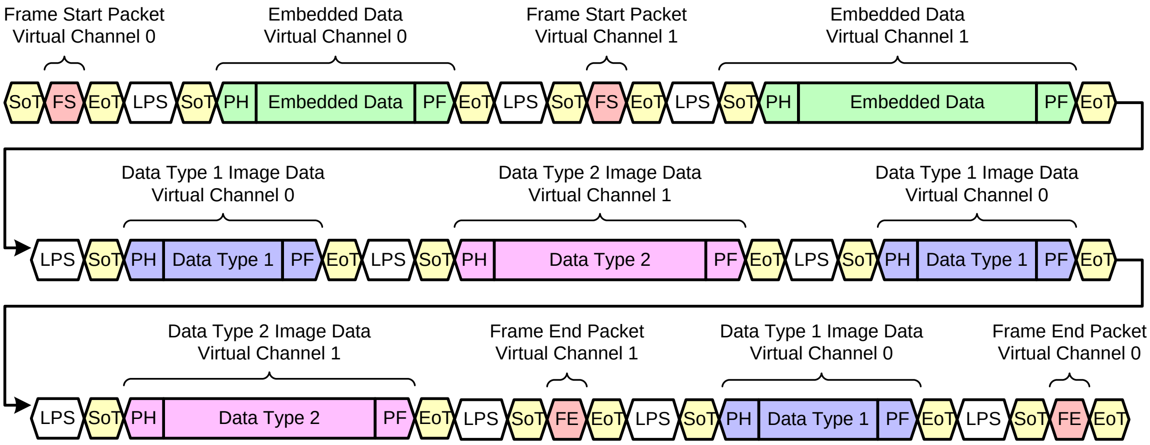
9.15.2 Virtual Channel Identifier Interleaving

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

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

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

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



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

FS – Frame Start Packet
FE – Frame End Packet

PH – Packet Header
PF – Packet Footer + Filler (if applicable)

Figure 94 Interleaved Data Transmission using Virtual Channels

This page intentionally left blank

10 Color Spaces

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

10.1 RGB Color Space Definition

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

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

10.2 YUV Color Space Definition

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

This page intentionally left blank.

11 Data Formats

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

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

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

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*

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

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

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

For the D-PHY physical layer option, each byte in the sequence is serially transmitted LSB-first, whereas for the C-PHY physical layer option, successive byte pairs in the sequence are encoded and then serially transmitted LSS-first. ***Figure 95*** illustrates these options for a single-Lane system.

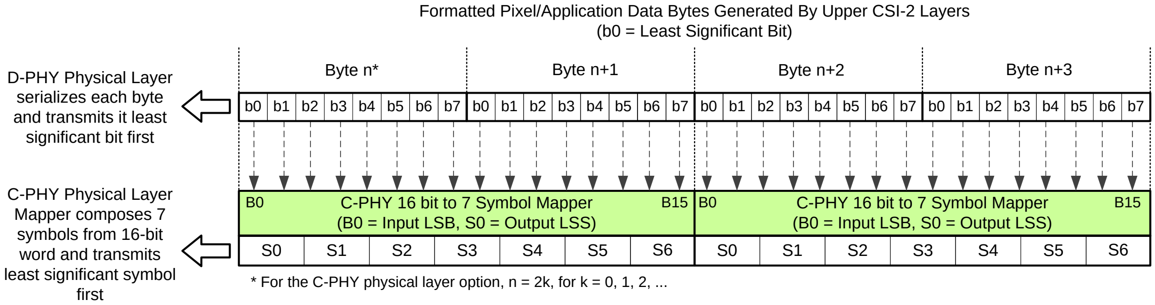

Formatted Pixel/Application Data Bytes Generated By Upper CSI-2 Layers
(b0 = Least Significant Bit)
Byte n*
Byte n+1
Byte n+2
Byte n+3
b0 b1 b2 b3 b4 b5 b6 b7
D-PHY Physical Layer serializes each byte and transmits it least significant bit first
C-PHY Physical Layer Mapper composes 7 symbols from 16-bit word and transmits least significant symbol first
B0
C-PHY 16 bit to 7 Symbol Mapper
(B0 = Input LSB, S0 = Output LSS)
B15
S0 S1 S2 S3 S4 S5 S6
* For the C-PHY physical layer option, n = 2k, for k = 0, 1, 2, ...

Figure 95 Byte Packing Pixel Data to C-PHY Symbol Illustration

11.1 Generic 8-bit Long Packet Data Types

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

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

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

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

11.1.2 Embedded Information

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

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

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

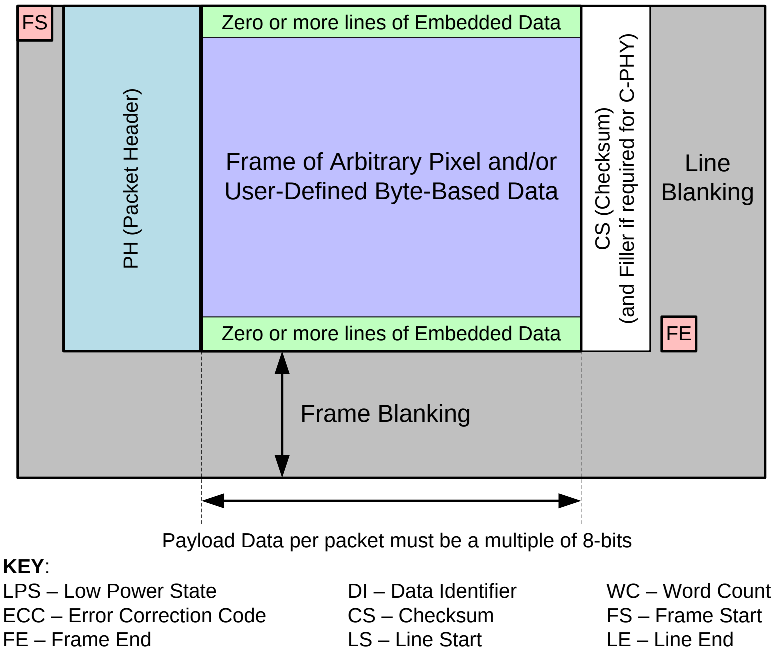

FS
PH (Packet Header)
Zero or more lines of Embedded Data
Frame of Arbitrary Pixel and/or User-Defined Byte-Based Data
Zero or more lines of Embedded Data
CS (Checksum) (and Filler if required for C-PHY)
Line Blanking
FE
Frame Blanking
Payload Data per packet must be a multiple of 8-bits
KEY:
LPS – Low Power State
ECC – Error Correction Code
FE – Frame End
DI – Data Identifier
CS – Checksum
LS – Line Start
WC – Word Count
FS – Frame 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

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

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

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

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

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

Table 27 Legacy YUV420 8-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

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

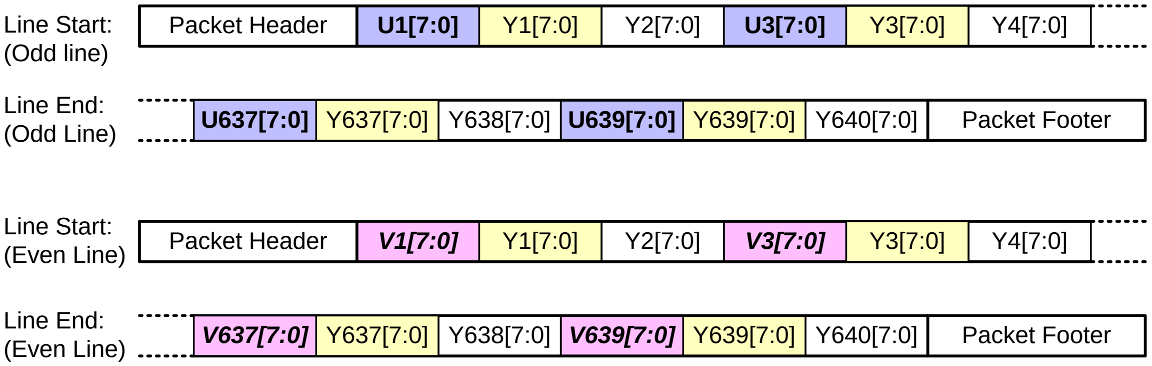


Figure 97 Legacy YUV420 8-bit Transmission

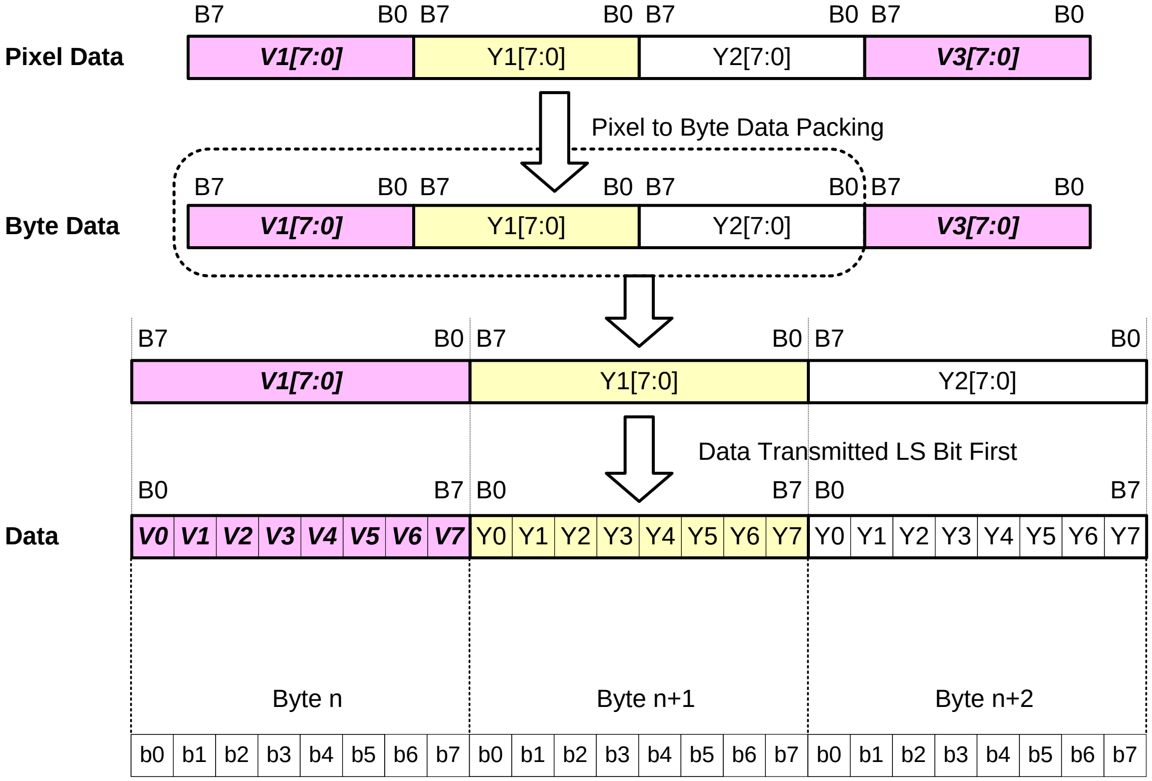


Figure 98 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

There is one spatial sampling option

- H.261, H.263 and MPEG1 Spatial Sampling (***Figure 99***).

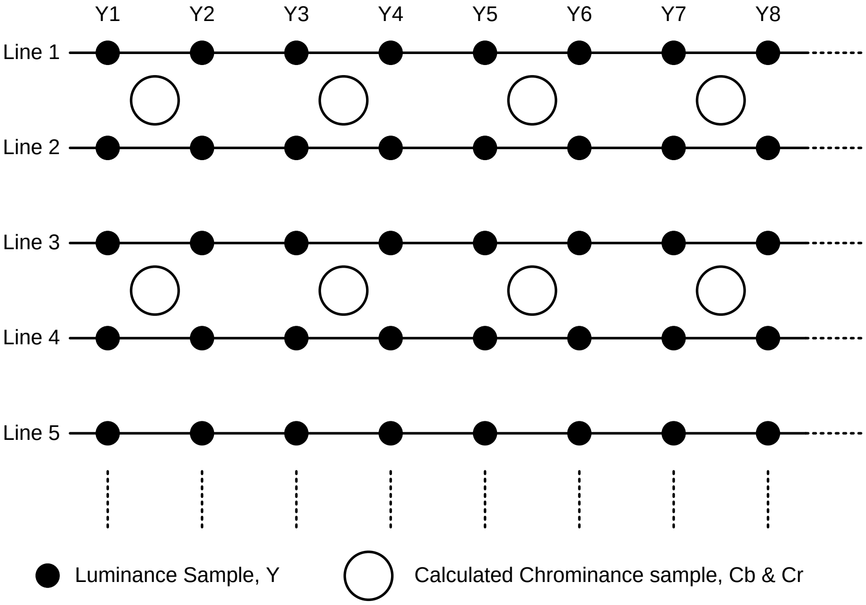
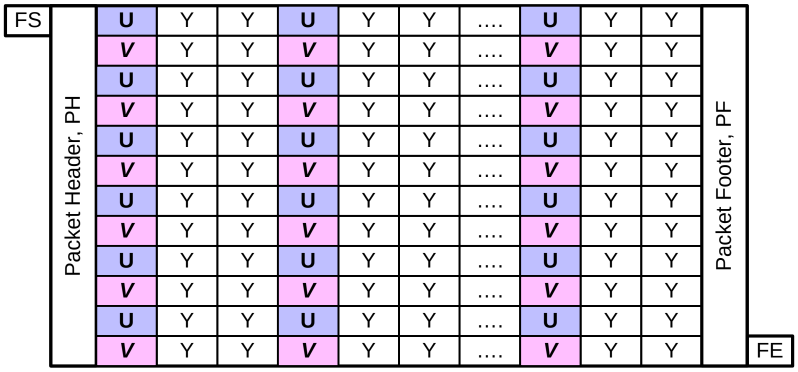


Figure 99 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1



U	Y	Y	**U**	Y	Y	….	**U**	Y	Y
V	Y	Y	***V***	Y	Y	….	***V***	Y	Y
U	Y	Y	**U**	Y	Y	….	**U**	Y	Y
V	Y	Y	***V***	Y	Y	….	***V***	Y	Y
U	Y	Y	**U**	Y	Y	….	**U**	Y	Y
V	Y	Y	***V***	Y	Y	….	***V***	Y	Y
U	Y	Y	**U**	Y	Y	….	**U**	Y	Y
V	Y	Y	***V***	Y	Y	….	***V***	Y	Y
U	Y	Y	**U**	Y	Y	….	**U**	Y	Y
V	Y	Y	***V***	Y	Y	….	***V***	Y	Y
U	Y	Y	**U**	Y	Y	….	**U**	Y	Y
V	Y	Y	***V***	Y	Y	….	***V***	Y	Y

Figure 100 Legacy YUV420 8-bit Frame Format

11.2.2 YUV420 8-bit

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

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

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

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

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

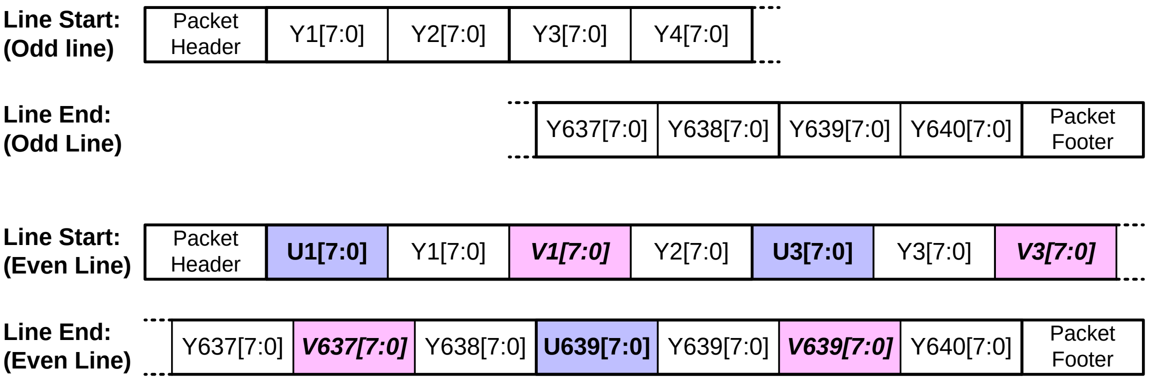


Figure 101 YUV420 8-bit Data Transmission Sequence

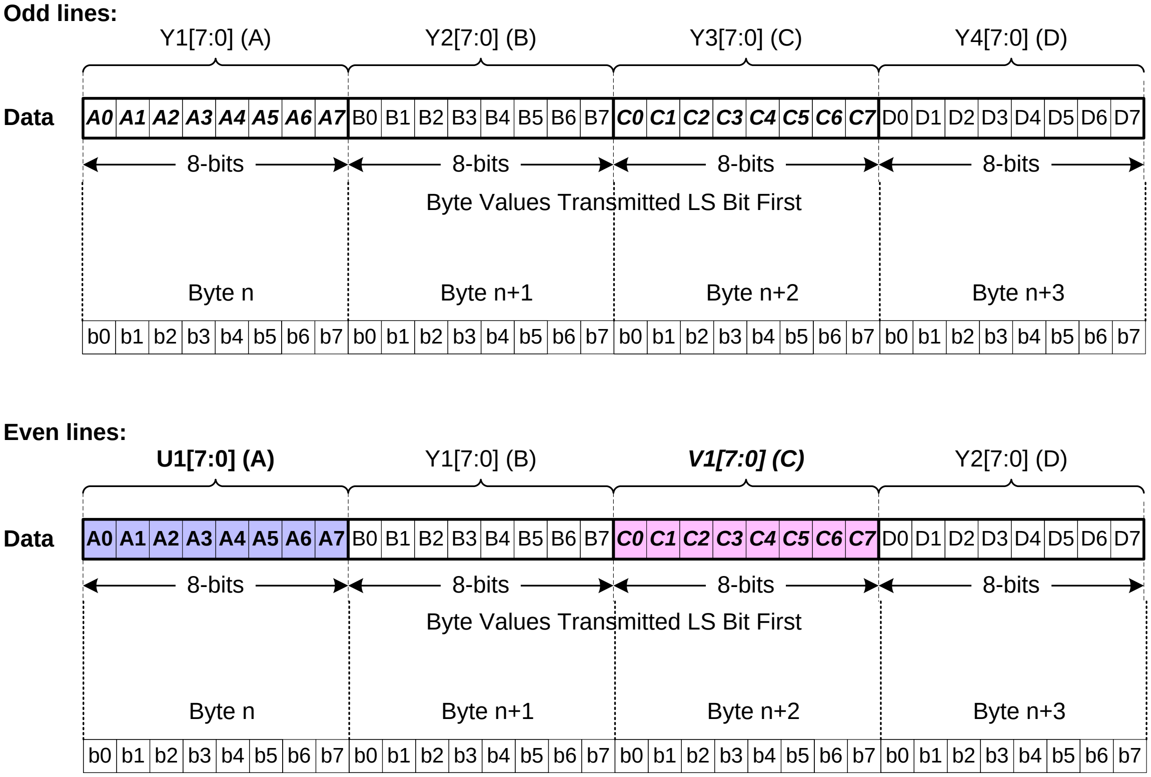
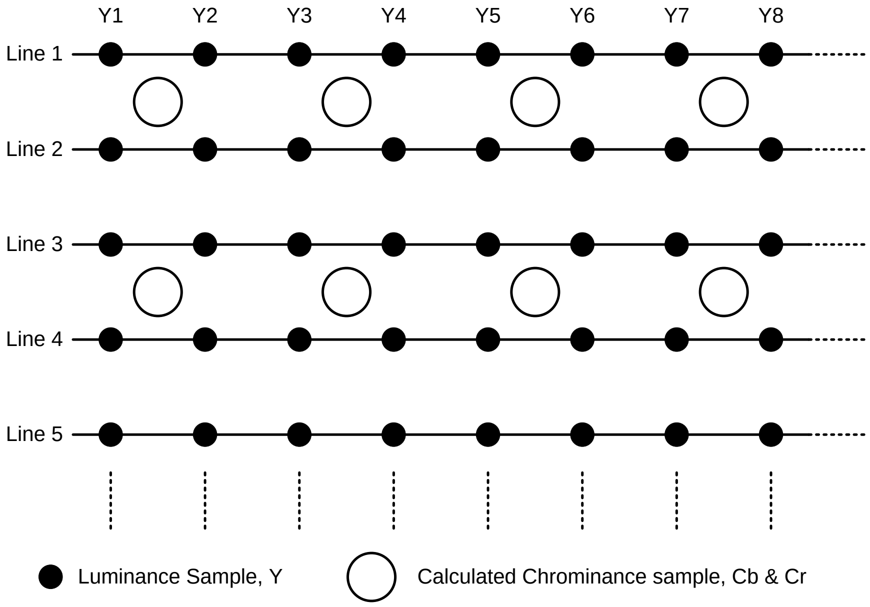

Odd lines:
Y1[7:0] (A)
Y2[7:0] (B)
Y3[7:0] (C)
Y4[7:0] (D)
Data
A0 A1 A2 A3 A4 A5 A6 A7 B0 B1 B2 B3 B4 B5 B6 B7 C0 C1 C2 C3 C4 C5 C6 C7 D0 D1 D2 D3 D4 D5 D6 D7
8-bits
8-bits
8-bits
8-bits
Byte Values Transmitted LS Bit First
Byte n
Byte n+1
Byte n+2
Byte n+3
b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7
Even lines:
U1[7:0] (A)
Y1[7:0] (B)
V1[7:0] (C)
Y2[7:0] (D)
Data
A0 A1 A2 A3 A4 A5 A6 A7 B0 B1 B2 B3 B4 B5 B6 B7 C0 C1 C2 C3 C4 C5 C6 C7 D0 D1 D2 D3 D4 D5 D6 D7
8-bits
8-bits
8-bits
8-bits
Byte Values Transmitted LS Bit First
Byte n
Byte n+1
Byte n+2
Byte n+3
b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7

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

There are two spatial sampling options

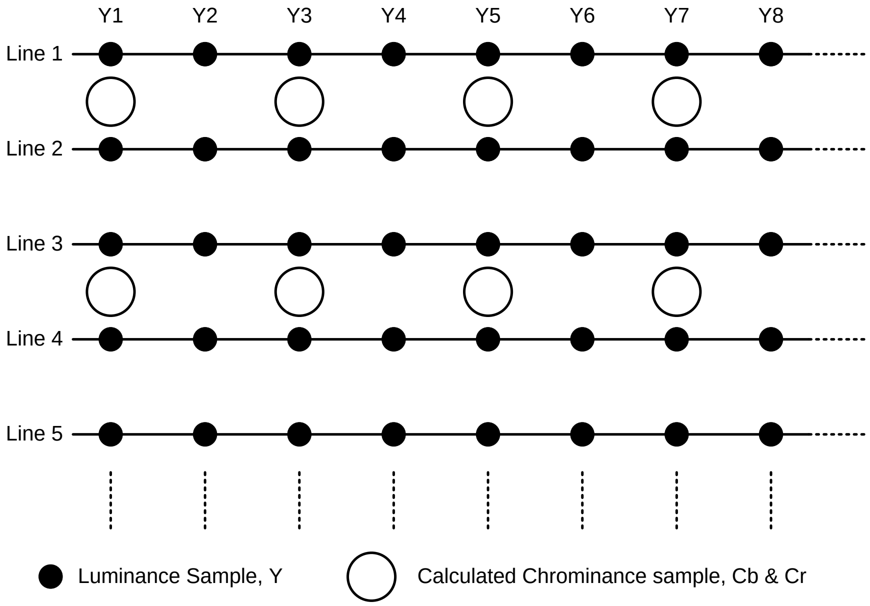
- H.261, H.263 and MPEG1 Spatial Sampling (***Figure 103***).
- Chroma Shifted Pixel Sampling (CSPS) for MPEG2, MPEG4 (***Figure 104***).

Figure 105 shows the YUV420 frame format.



1617

Figure 103 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1



1618

Figure 104 YUV420 Spatial Sampling for MPEG 2 and MPEG 4

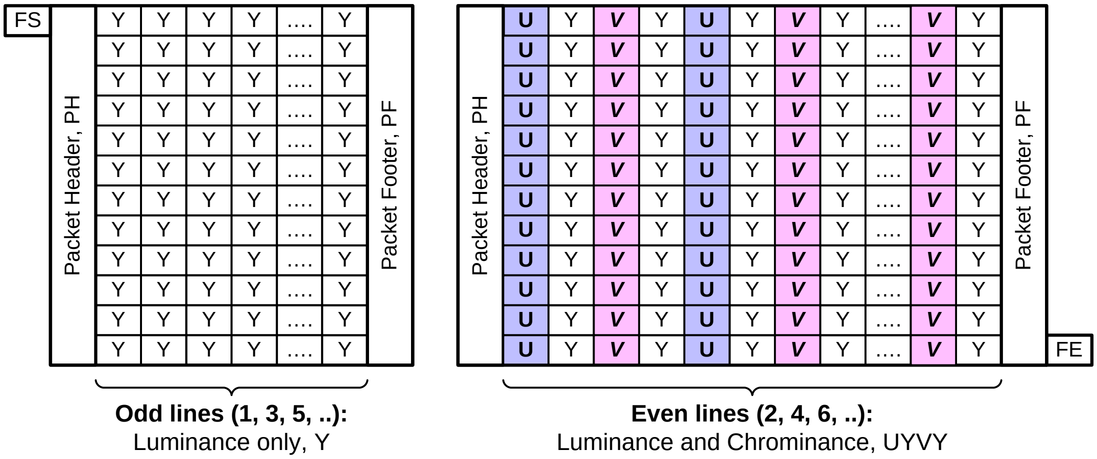


Figure 105 YUV420 8-bit Frame Format

11.2.3 YUV420 10-bit

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

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

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

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

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

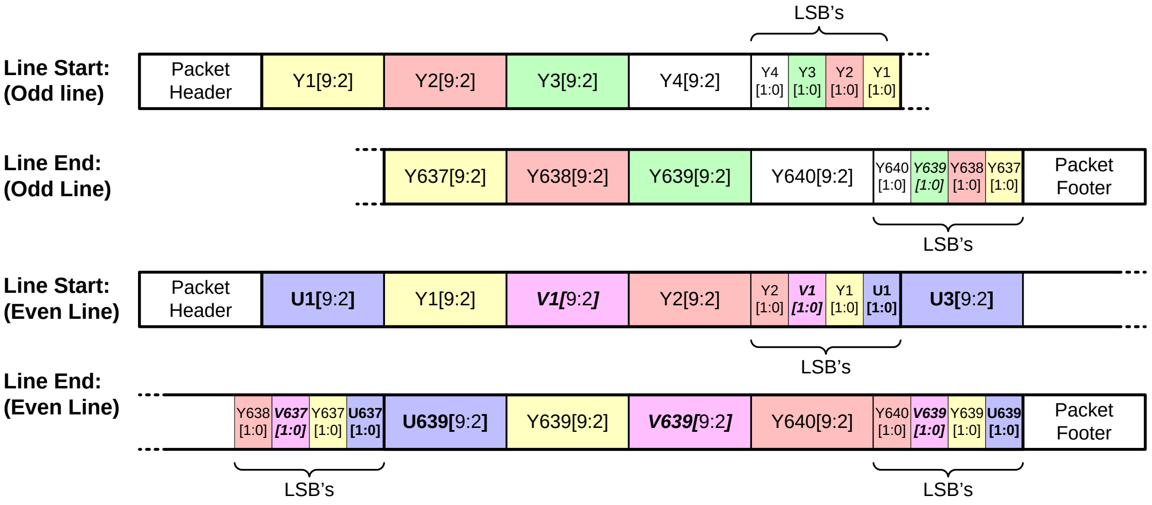


Figure 106 YUV420 10-bit Transmission

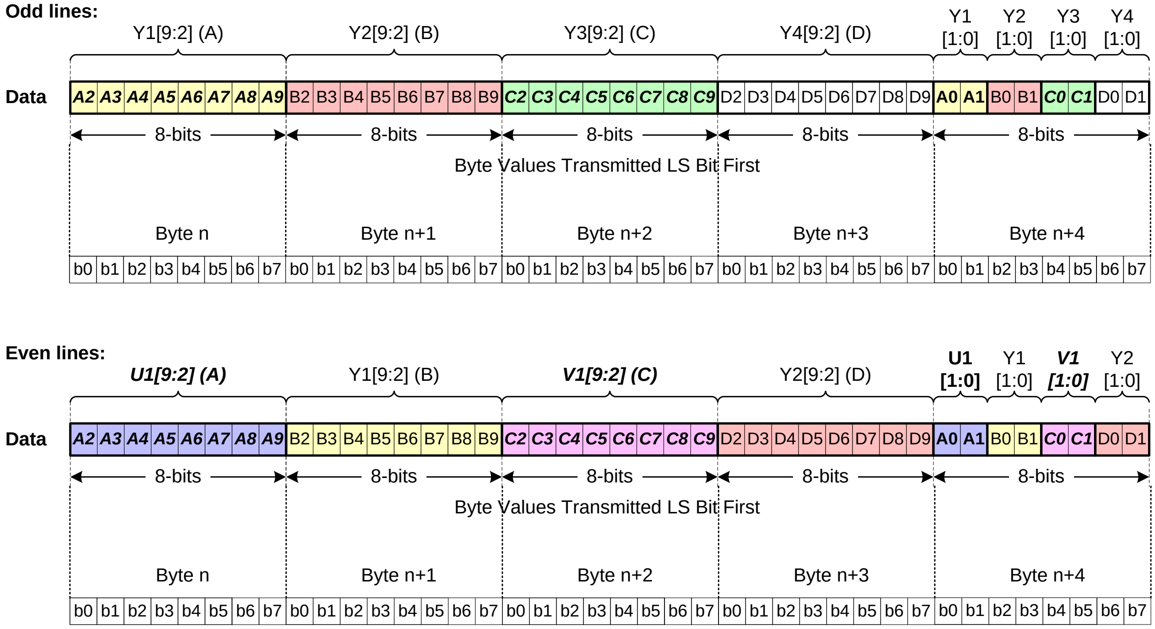


Figure 107 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration

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

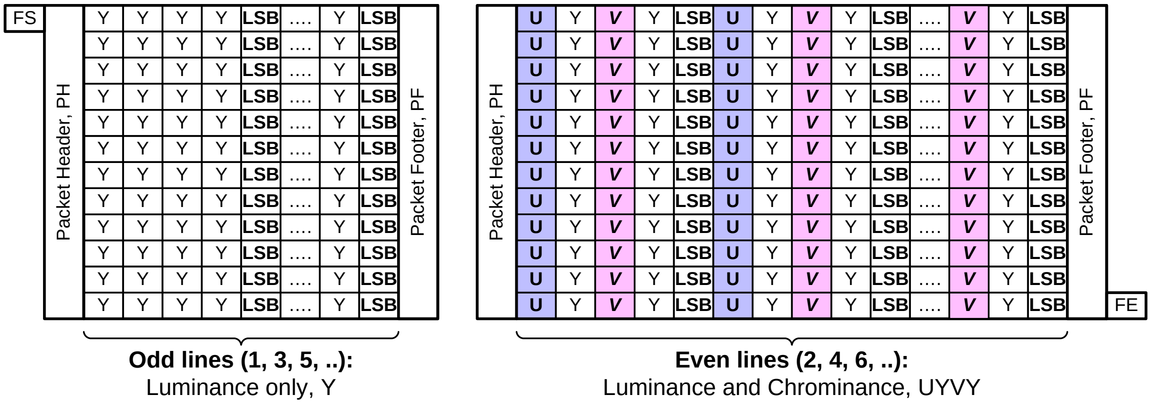


Figure 108 YUV420 10-bit Frame Format

11.2.4 YUV422 8-bit

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

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

Table 30 YUV422 8-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	4	32

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

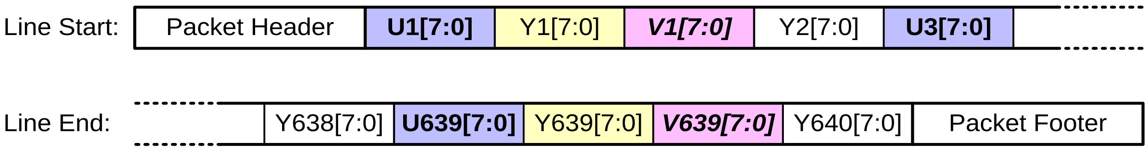


Figure 109 YUV422 8-bit Transmission

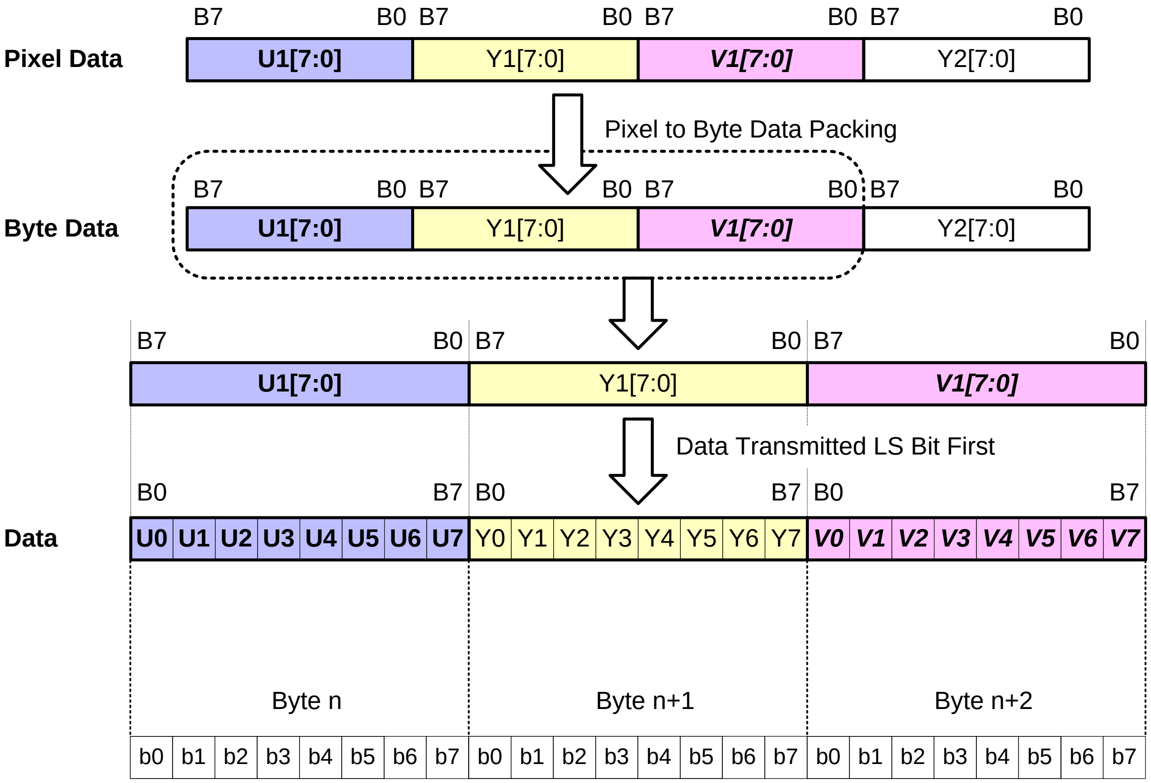


Figure 110 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration

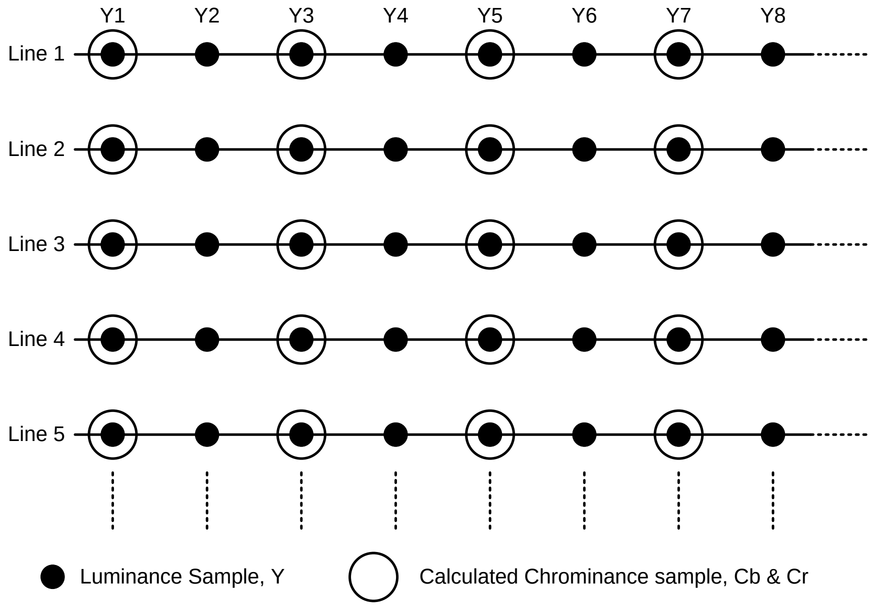
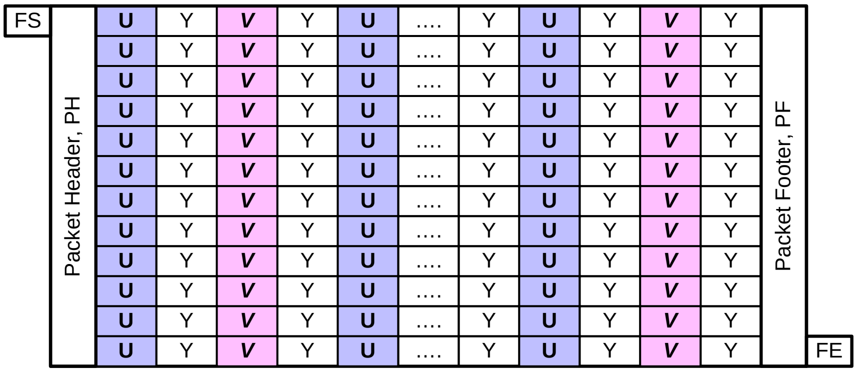


Figure 111 YUV422 Co-sited Spatial Sampling

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



U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y
U	Y	***V***	Y	**U**	....	Y	**U**	Y	***V***	Y

Figure 112 YUV422 8-bit Frame Format

11.2.5 YUV422 10-bit

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

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.

Table 31 YUV422 10-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	5	40

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

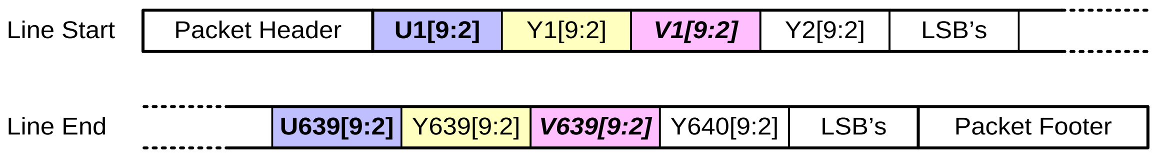


Figure 113 YUV422 10-bit Transmitted Bytes

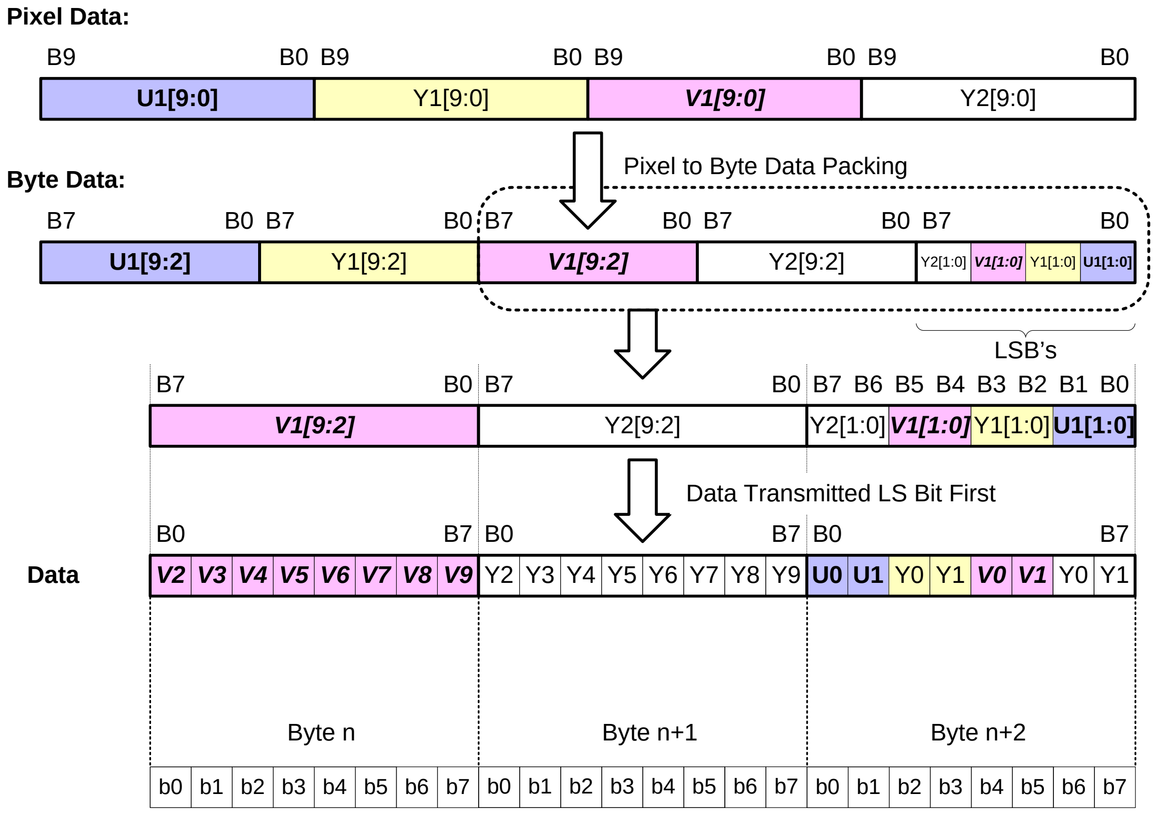


Figure 114 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration

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***.

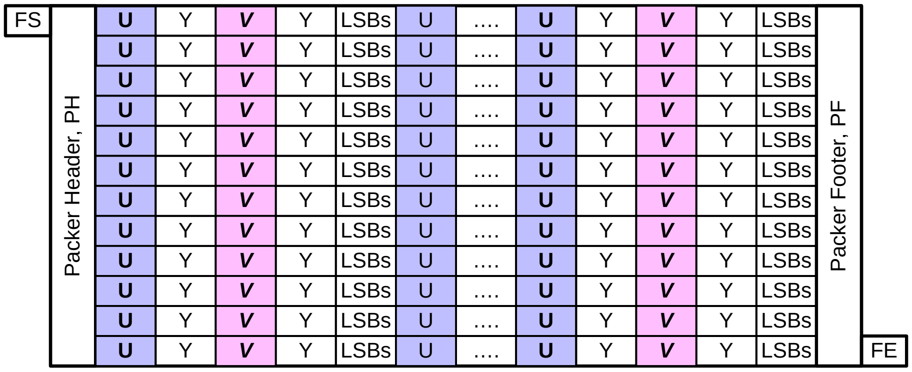


Figure 115 YUV422 10-bit Frame Format

11.3 RGB Image Data

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

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

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

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

Table 33 RGB888 Packet Data Size Constraints

Pixels	Bytes	Bits
1	3	24

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in ***Figure 117***.

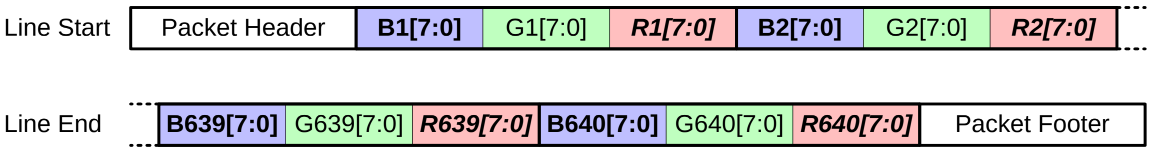


Figure 116 RGB888 Transmission

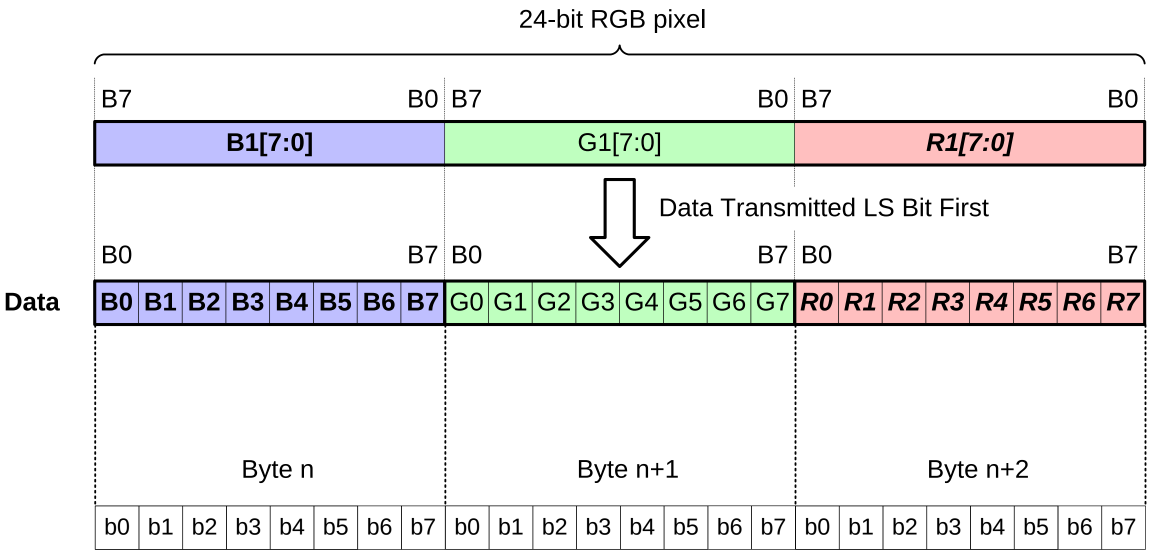


Figure 117 RGB888 Transmission in CSI-2 Bus Bitwise Illustration

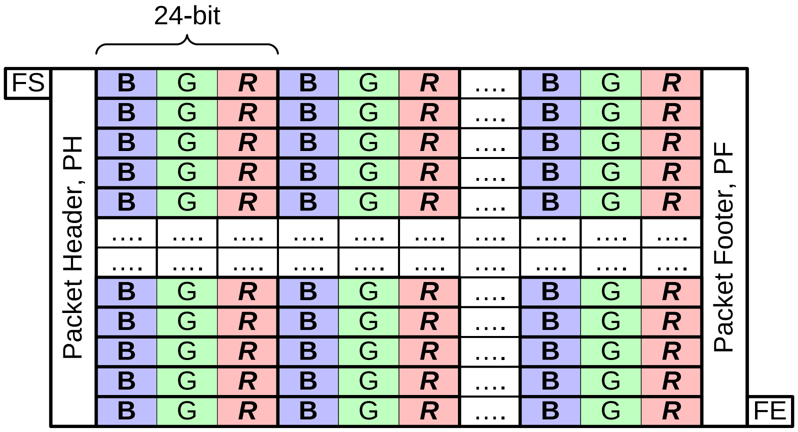


Figure 118 RGB888 Frame Format

11.3.2 RGB666

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

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

Table 34 RGB666 Packet Data Size Constraints

Pixels	Bytes	Bits
4	9	72

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

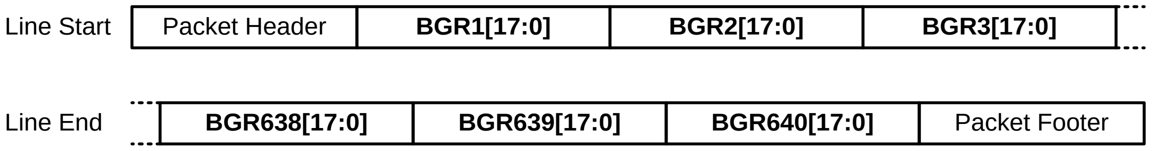


Figure 119 RGB666 Transmission with 18-bit BGR Words

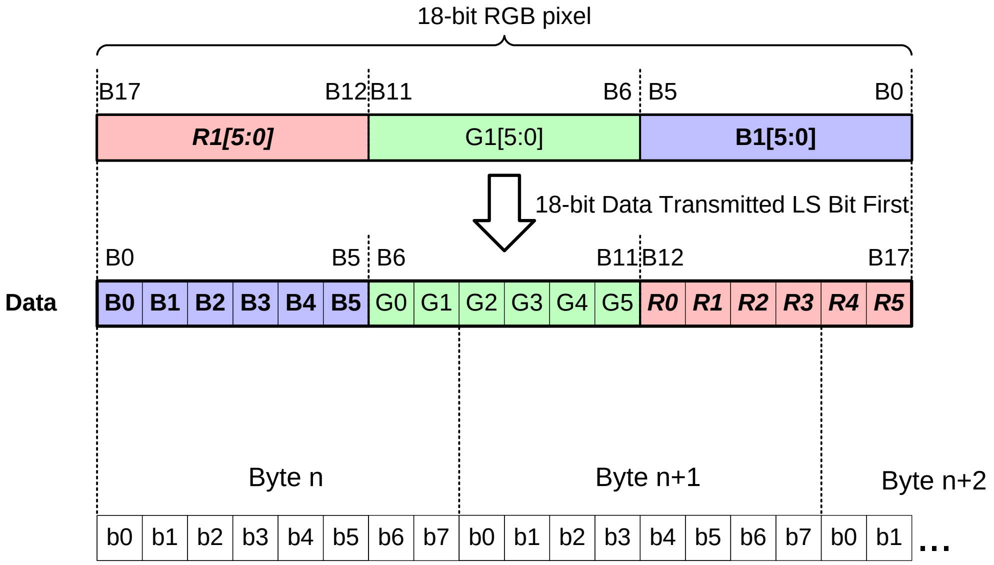


Figure 120 RGB666 Transmission on CSI-2 Bus Bitwise Illustration

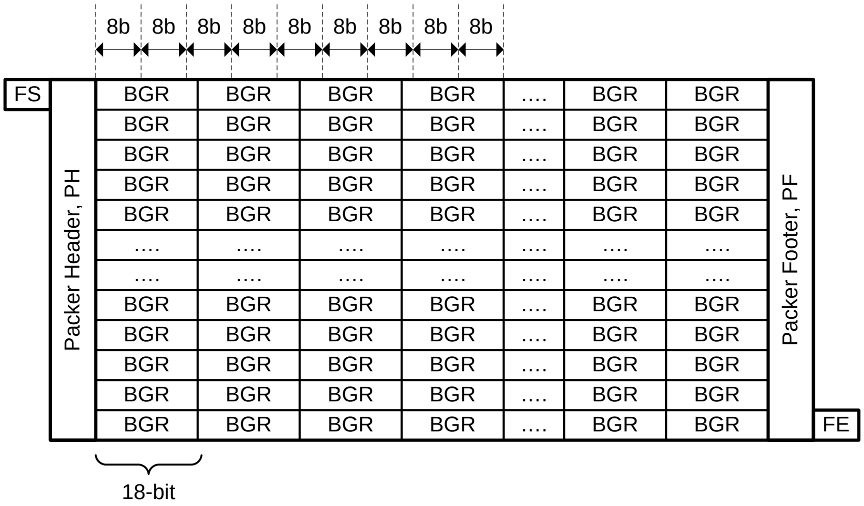


Figure 121 RGB666 Frame Format

11.3.3 RGB565

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

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

Table 35 RGB565 Packet Data Size Constraints

Pixels	Bytes	Bits
1	2	16

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

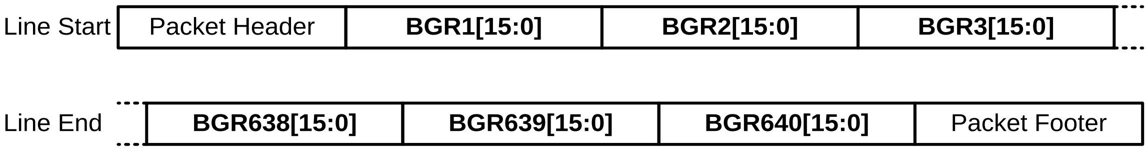


Figure 122 RGB565 Transmission with 16-bit BGR Words

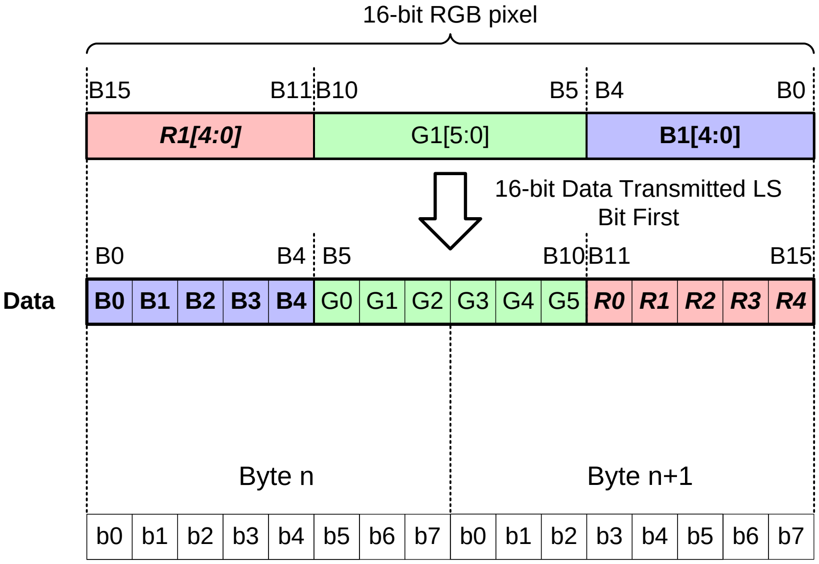


Figure 123 RGB565 Transmission on CSI-2 Bus Bitwise Illustration

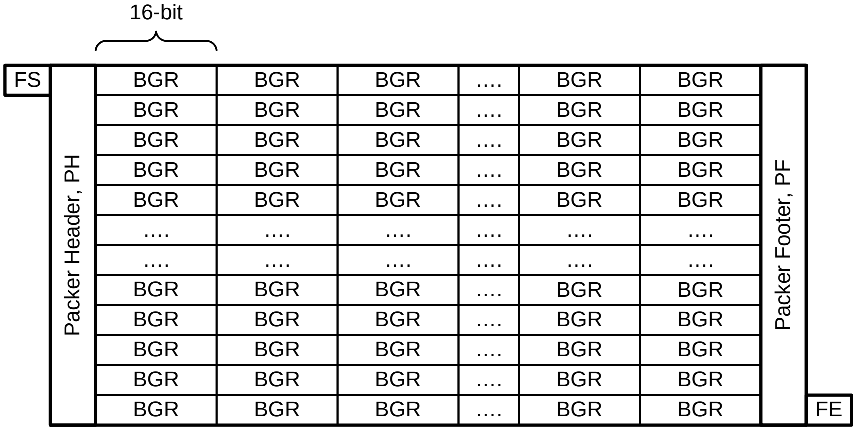


Figure 124 RGB565 Frame Format

11.3.4 RGB555

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

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

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

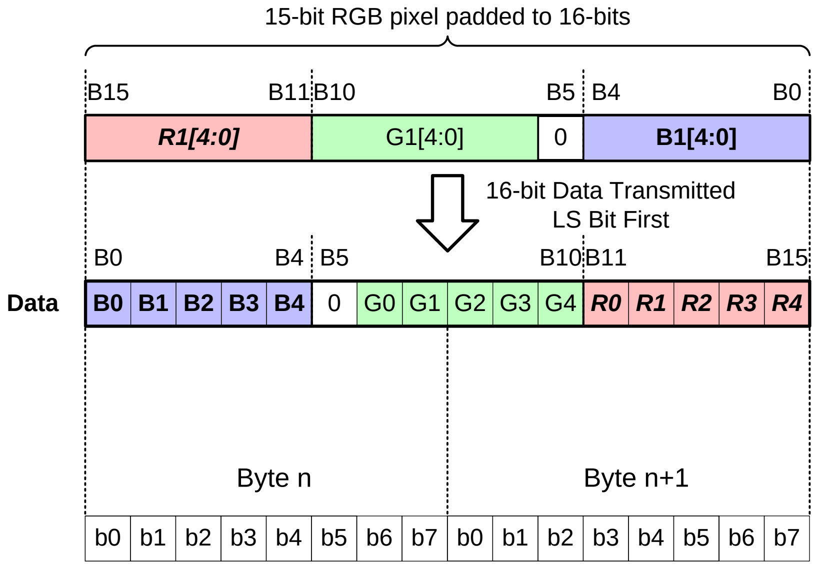


Figure 125 RGB555 Transmission on CSI-2 Bus Bitwise Illustration

11.3.5 RGB444

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

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

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

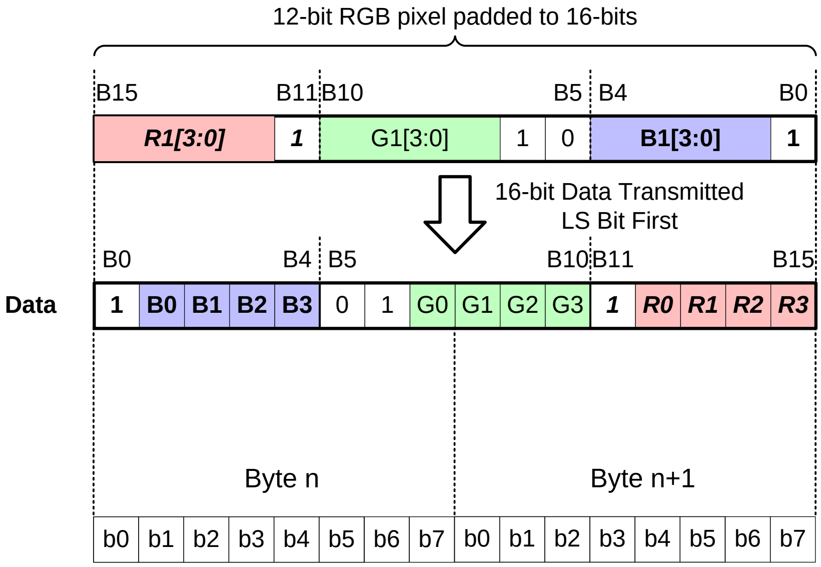


Figure 126 RGB444 Transmission on CSI-2 Bus Bitwise Illustration

11.4 RAW Image Data

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

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

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

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

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

See Errata 01, Item 1

11.4.1 RAW6

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

Table 37 RAW6 Packet Data Size Constraints

Pixels	Bytes	Bits
4	3	24

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

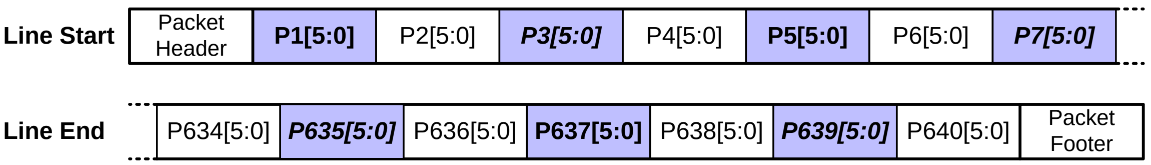


Figure 127 RAW6 Transmission

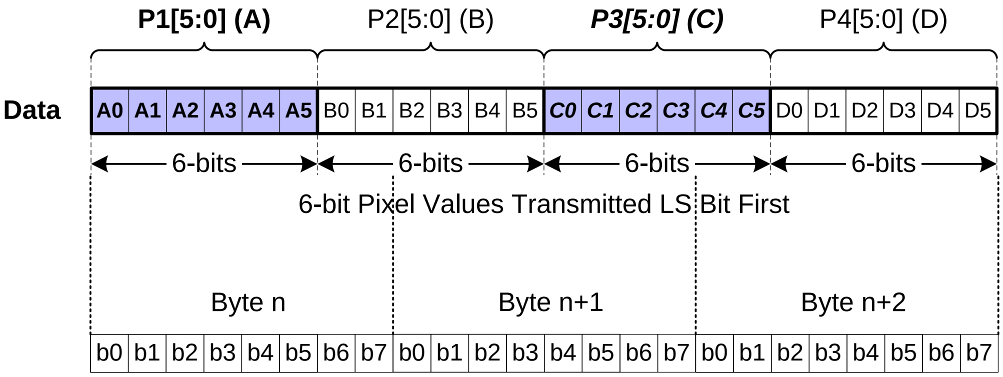

P1[5:0] (A)
P2[5:0] (B)
P3[5:0] (C)
P4[5:0] (D)
Data
A0 A1 A2 A3 A4 A5 B0 B1 B2 B3 B4 B5 C0 C1 C2 C3 C4 C5 D0 D1 D2 D3 D4 D5
6-bits
6-bits
6-bits
6-bits
6-bit Pixel Values Transmitted LS Bit First
Byte n
Byte n+1
Byte n+2
b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7

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

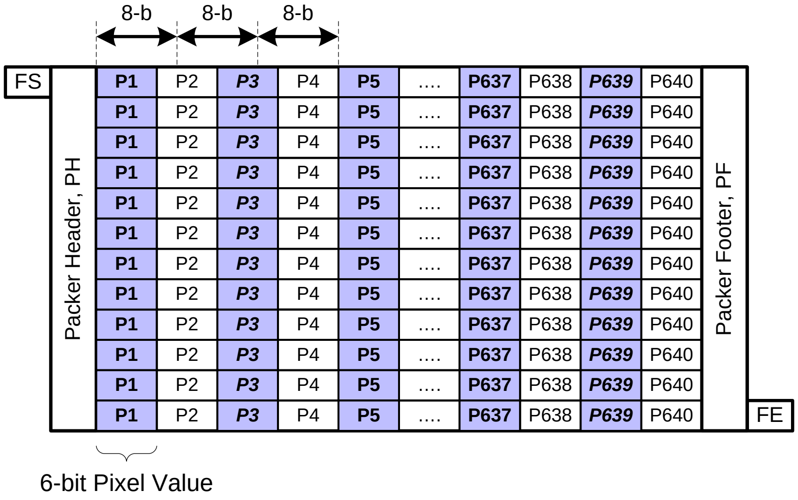


Figure 129 RAW6 Frame Format

11.4.2 RAW7

See Errata 01, Item 2

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

Table 38 RAW7 Packet Data Size Constraints

Pixels	Bytes	Bits
8	7	56

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

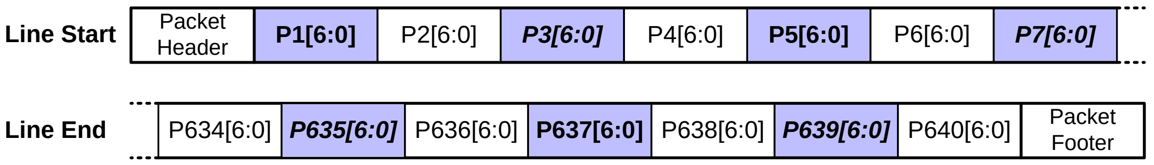


Figure 130 RAW7 Transmission

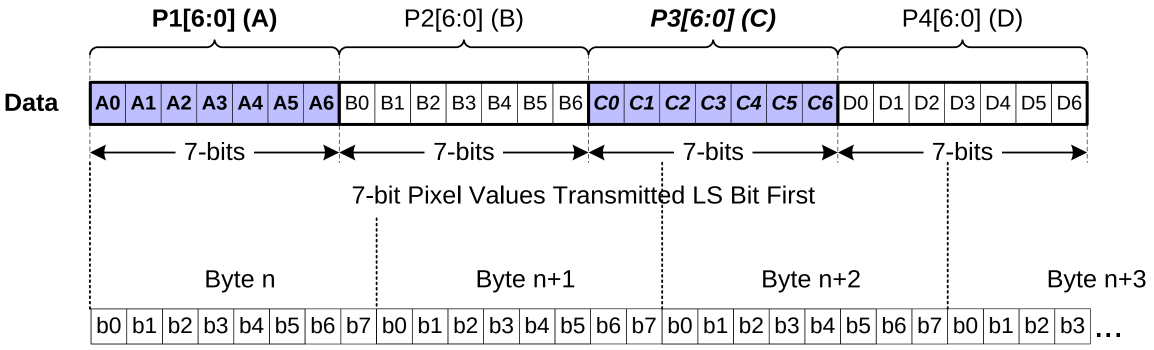


Figure 131 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration

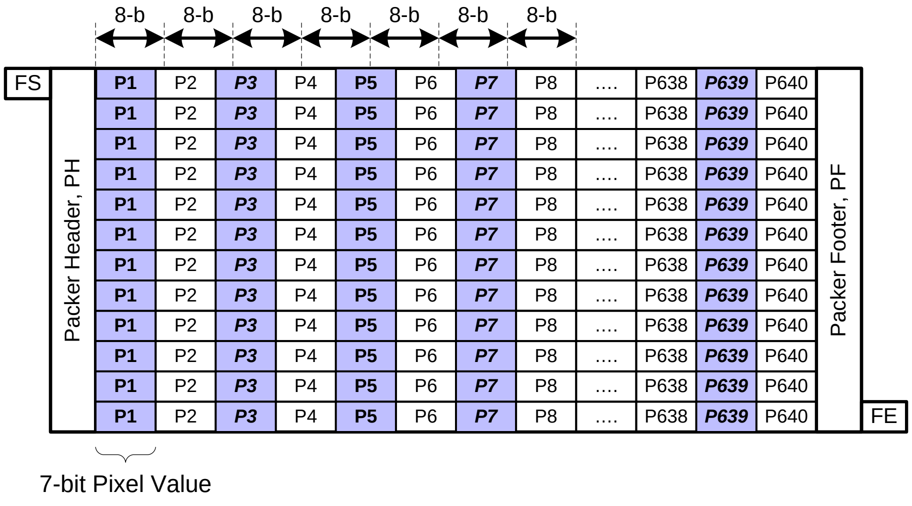


Figure 132 RAW7 Frame Format

11.4.3 RAW8

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

Table 39 RAW8 Packet Data Size Constraints

Pixels	Bytes	Bits
1	1	8

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

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

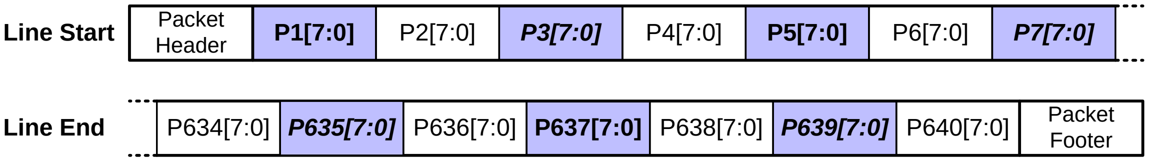


Figure 133 RAW8 Transmission

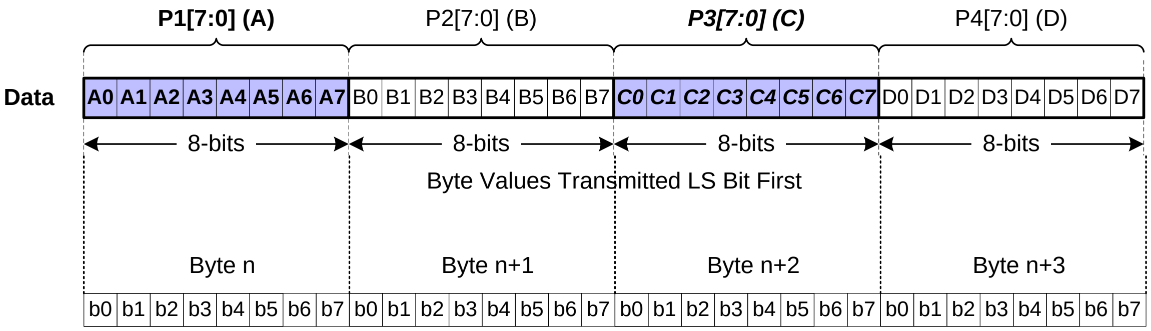
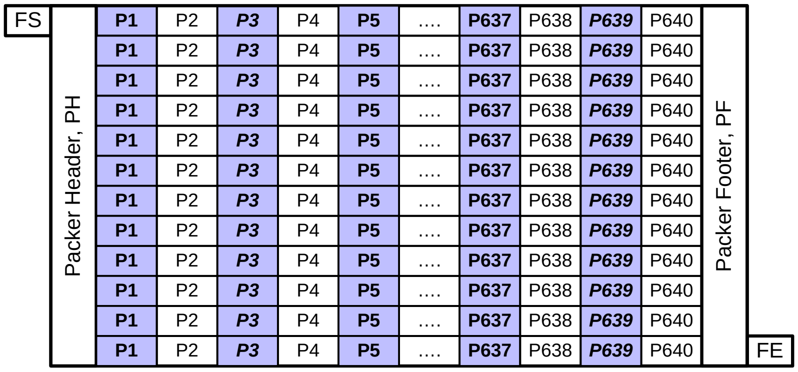

P1[7:0] (A)
P2[7:0] (B)
P3[7:0] (C)
P4[7:0] (D)
Data
A0 A1 A2 A3 A4 A5 A6 A7 B0 B1 B2 B3 B4 B5 B6 B7 C0 C1 C2 C3 C4 C5 C6 C7 D0 D1 D2 D3 D4 D5 D6 D7
8-bits
8-bits
8-bits
8-bits
Byte Values Transmitted LS Bit First
Byte n
Byte n+1
Byte n+2
Byte n+3
b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7

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



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

Figure 135 RAW8 Frame Format

11.4.4 RAW10

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

Table 40 RAW10 Packet Data Size Constraints

Pixels	Bytes	Bits
4	5	40

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

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

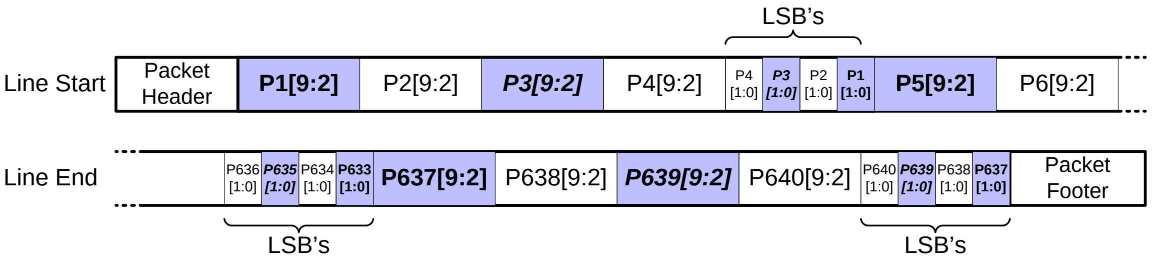


Figure 136 RAW10 Transmission

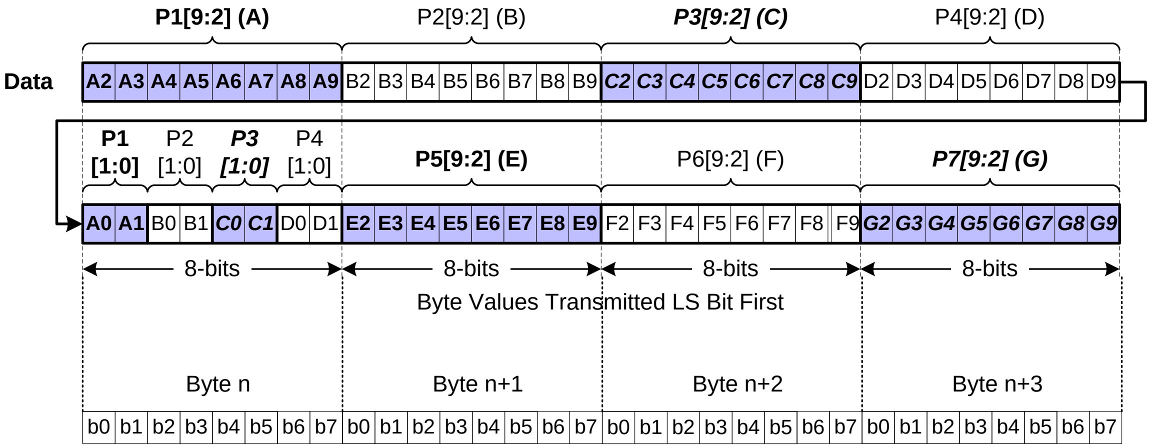
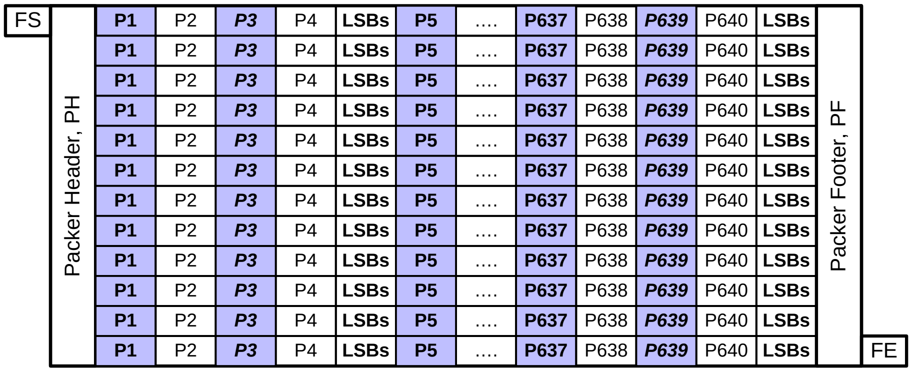

P1[9:2] (A)
P2[9:2] (B)
P3[9:2] (C)
P4[9:2] (D)
Data
A2 A3 A4 A5 A6 A7 A8 A9 B2 B3 B4 B5 B6 B7 B8 B9 C2 C3 C4 C5 C6 C7 C8 C9 D2 D3 D4 D5 D6 D7 D8 D9
P1 [1:0]
P2 [1:0]
P3 [1:0]
P4 [1:0]
P5[9:2] (E)
P6[9:2] (F)
P7[9:2] (G)
A0 A1 B0 B1 C0 C1 D0 D1 E2 E3 E4 E5 E6 E7 E8 E9 F2 F3 F4 F5 F6 F7 F8 F9 G2 G3 G4 G5 G6 G7 G8 G9
8-bits
8-bits
8-bits
8-bits
Byte Values Transmitted LS Bit First
Byte n
Byte n+1
Byte n+2
Byte n+3
b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7

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**

Figure 138 RAW10 Frame Format

11.4.5 RAW12

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

Table 41 RAW12 Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

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

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

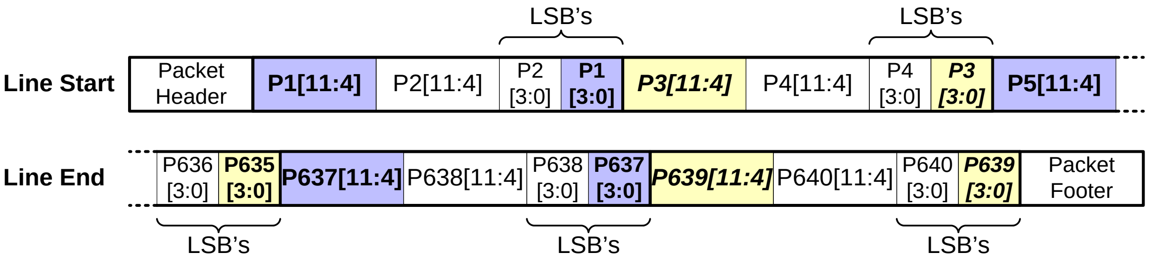


Figure 139 RAW12 Transmission

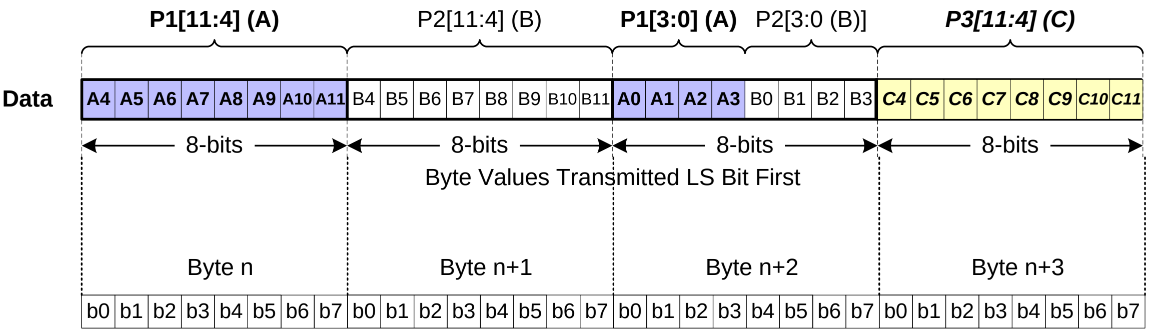


Figure 140 RAW12 Transmission on CSI-2 Bus Bitwise Illustration

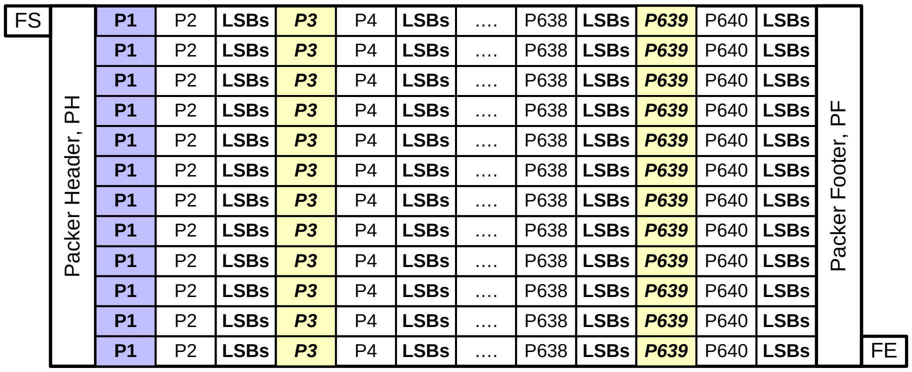


Figure 141 RAW12 Frame Format

11.4.6 RAW14

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

Table 42 RAW14 Packet Data Size Constraints

Pixels	Bytes	Bits
4	7	56

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

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

Note:

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

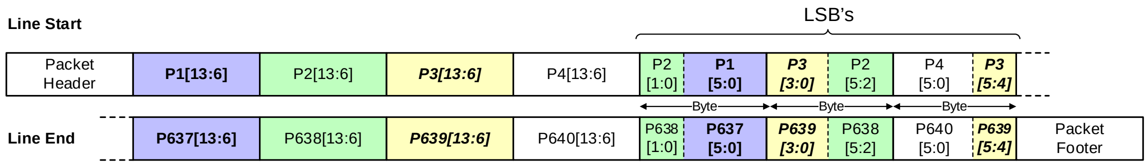


Figure 142 RAW14 Transmission

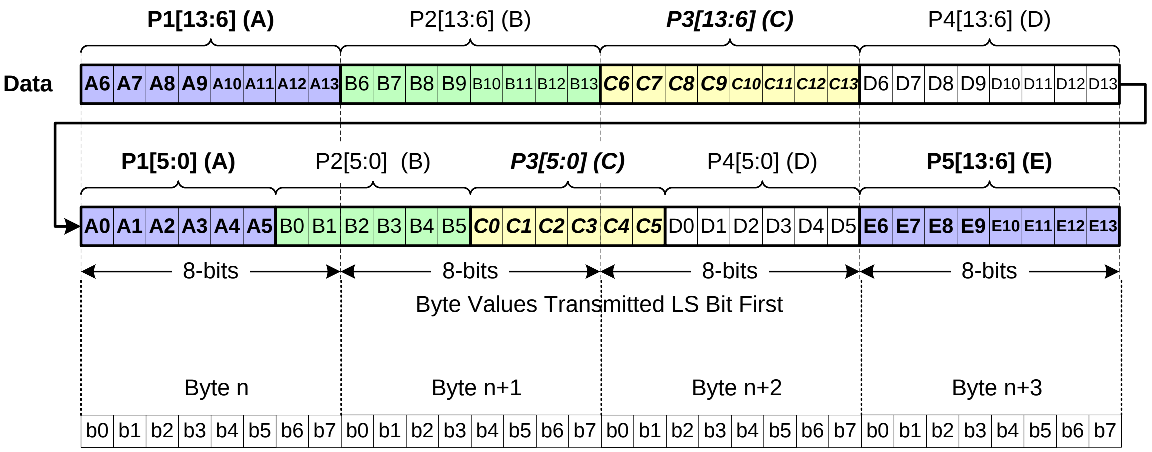


Figure 143 RAW14 Transmission on CSI-2 Bus Bitwise Illustration

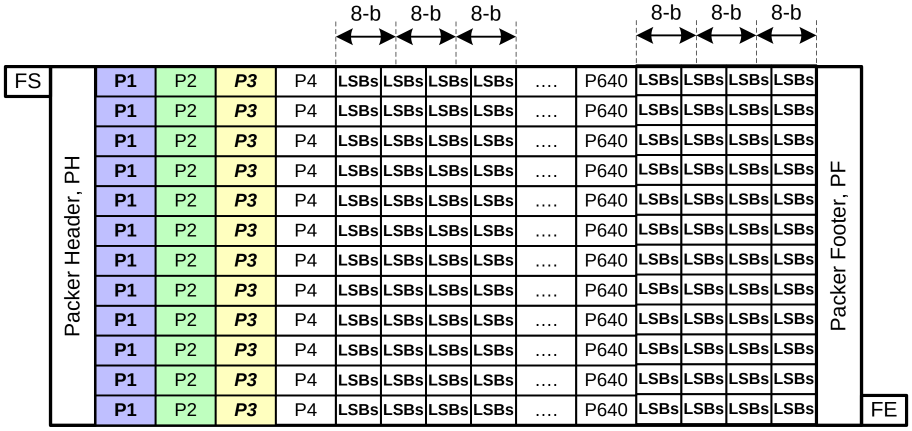


Figure 144 RAW14 Frame Format

11.4.7 RAW16

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

Table 43 RAW16 Packet Data Size Constraints

Pixels	Bytes	Bits
1	2	16

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

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

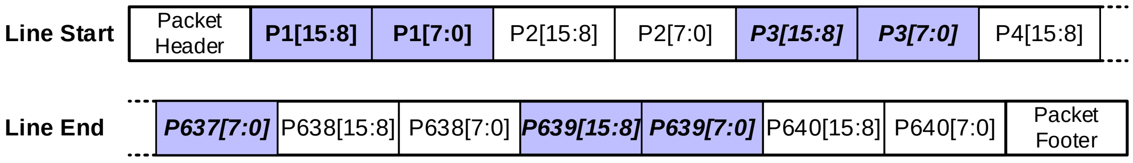


Figure 145 RAW16 Transmission

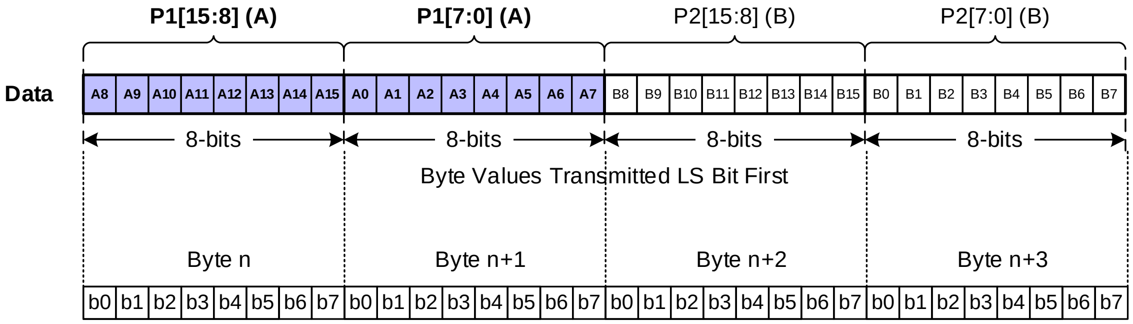

P1[15:8] (A)
P1[7:0] (A)
P2[15:8] (B)
P2[7:0] (B)
Data
A8 A9 A10 A11 A12 A13 A14 A15 A0 A1 A2 A3 A4 A5 A6 A7 B8 B9 B10 B11 B12 B13 B14 B15 B0 B1 B2 B3 B4 B5 B6 B7
8-bits
8-bits
8-bits
8-bits
Byte Values Transmitted LS Bit First
Byte n
Byte n+1
Byte n+2
Byte n+3
b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7

Figure 146 RAW16 Transmission on CSI-2 Bus Bitwise Illustration

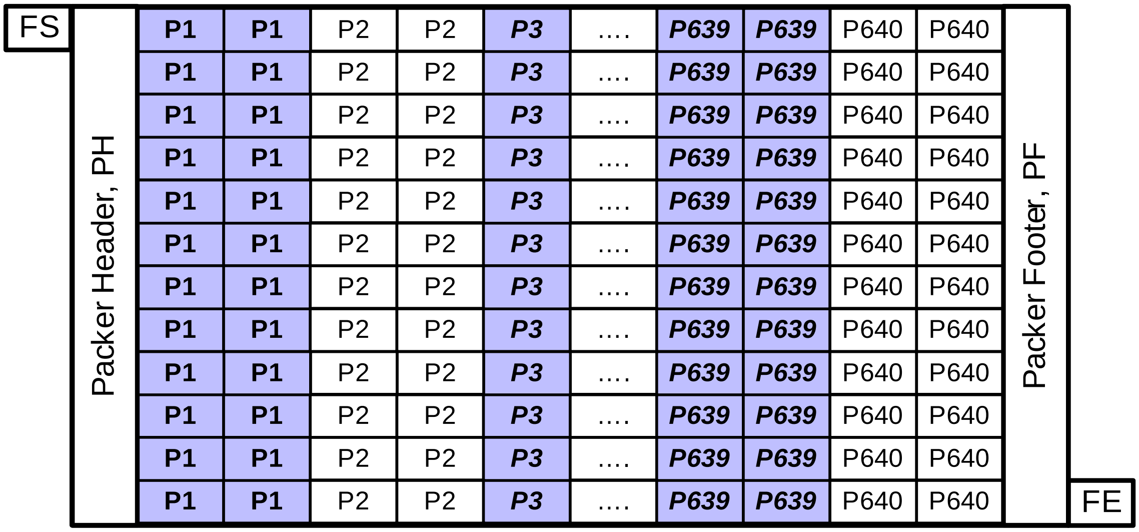


Figure 147 RAW16 Frame Format

11.4.8 RAW20

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

Table 44 RAW20 Packet Data Size Constraints

Pixels	Bytes	Bits
2	5	40

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

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

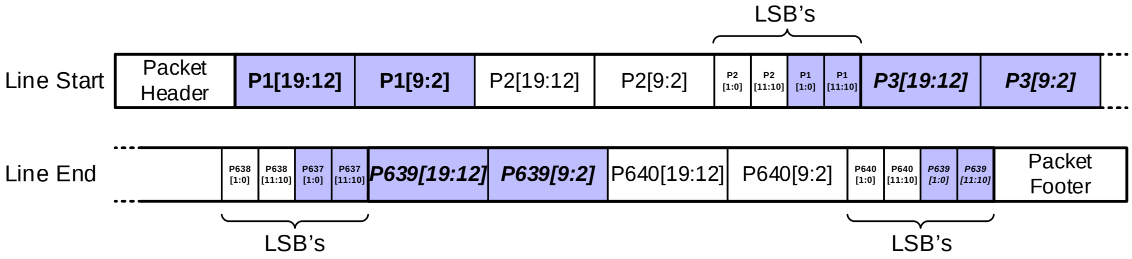


Figure 148 RAW20 Transmission

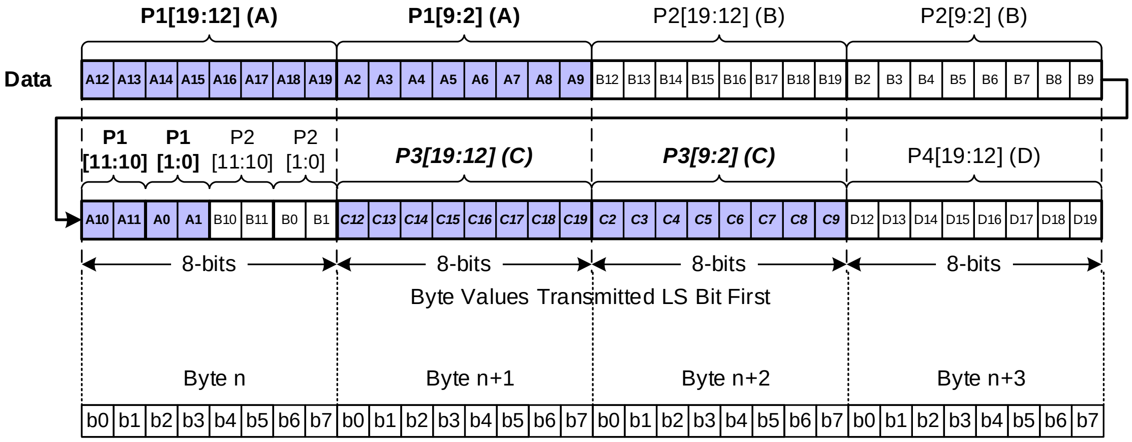
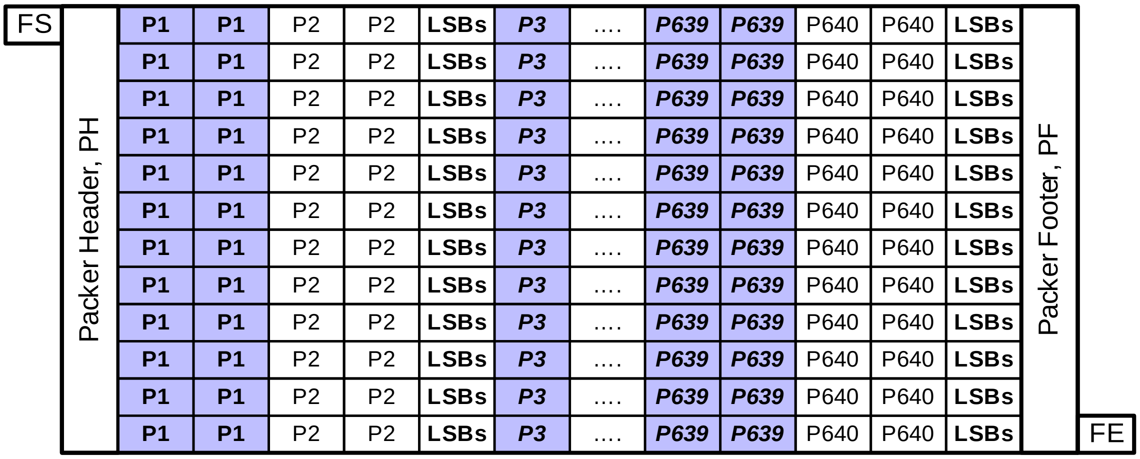

P1[19:12] (A)
P1[9:2] (A)
P2[19:12] (B)
P2[9:2] (B)
Data
A12 A13 A14 A15 A16 A17 A18 A19 A2 A3 A4 A5 A6 A7 A8 A9 B12 B13 B14 B15 B16 B17 B18 B19 B2 B3 B4 B5 B6 B7 B8 B9
P1 [11:10]
P1 [1:0]
P2 [11:10]
P2 [1:0]
P3[19:12] (C)
P3[9:2] (C)
P4[19:12] (D)
A10 A11 A0 A1 B10 B11 B0 B1 C12 C13 C14 C15 C16 C17 C18 C19 C2 C3 C4 C5 C6 C7 C8 C9 D12 D13 D14 D15 D16 D17 D18 D19
8-bits
8-bits
8-bits
8-bits
Byte Values Transmitted LS Bit First
Byte n
Byte n+1
Byte n+2
Byte n+3
b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7 b0 b1 b2 b3 b4 b5 b6 b7

Figure 149 RAW20 Transmission on CSI-2 Bus Bitwise Illustration



P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**
P1	**P1**	P2	P2	**LSBs**	***P3***	….	***P639***	***P639***	P640	P640	**LSBs**

Figure 150 RAW20 Frame Format

11.5 User Defined Data Formats

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

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

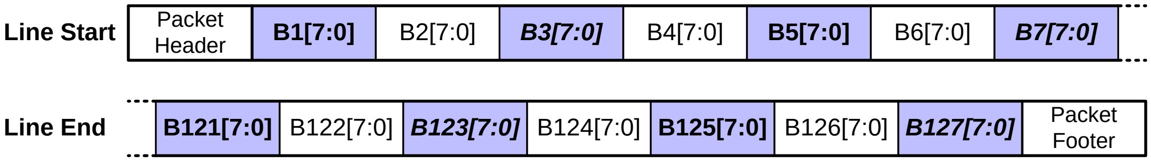


Figure 151 User Defined 8-bit Data (128 Byte Packet)

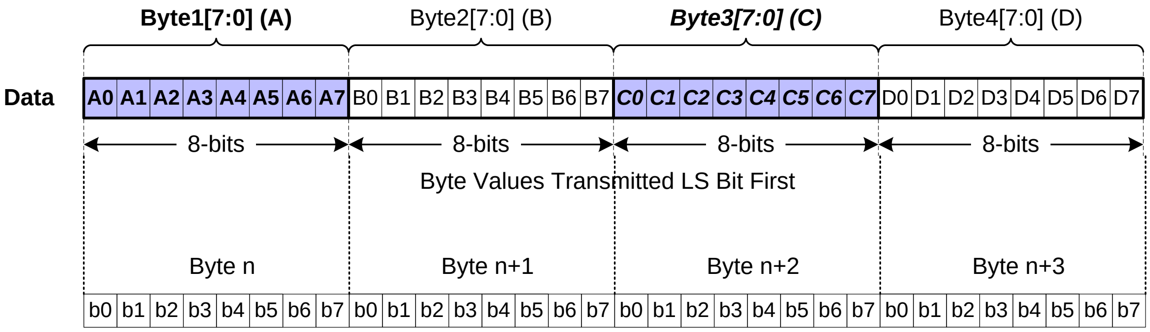


Figure 152 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration

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

For User Defined data:

- The frame is transmitted as a sequence of arbitrary sized packets.
- The packet size may vary from packet to packet.
- The packet spacing may vary between packets.

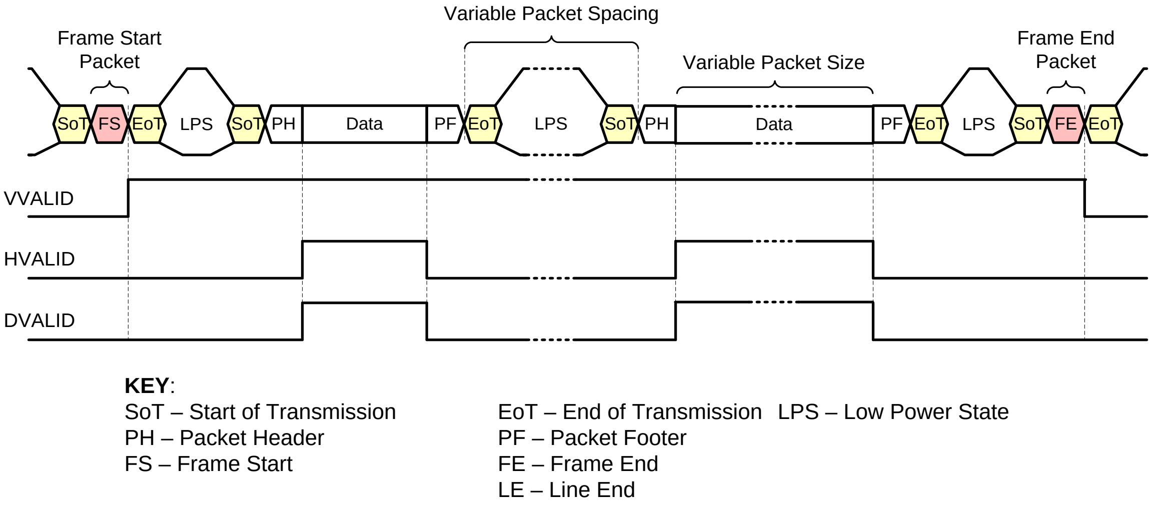


Figure 153 Transmission of User Defined 8-bit Data

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

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

This section is informative.

The CSI-2 data protocol requires certain behavior from the receiver connected to the CSI transmitter. The following sections describe how different data formats should be stored inside the receiver. While informative, this section is provided to ease application software development by suggesting a common data storage format among different receivers.

12.1 General/Arbitrary Data Reception

In the generic case and for arbitrary data the first byte of payload data transmitted maps the LS byte of the 32-bit memory word and the fourth byte of payload data transmitted maps to the MS byte of the 32-bit memory word.

Figure 154 shows the generic CSI-2 byte to 32-bit memory word mapping rule.

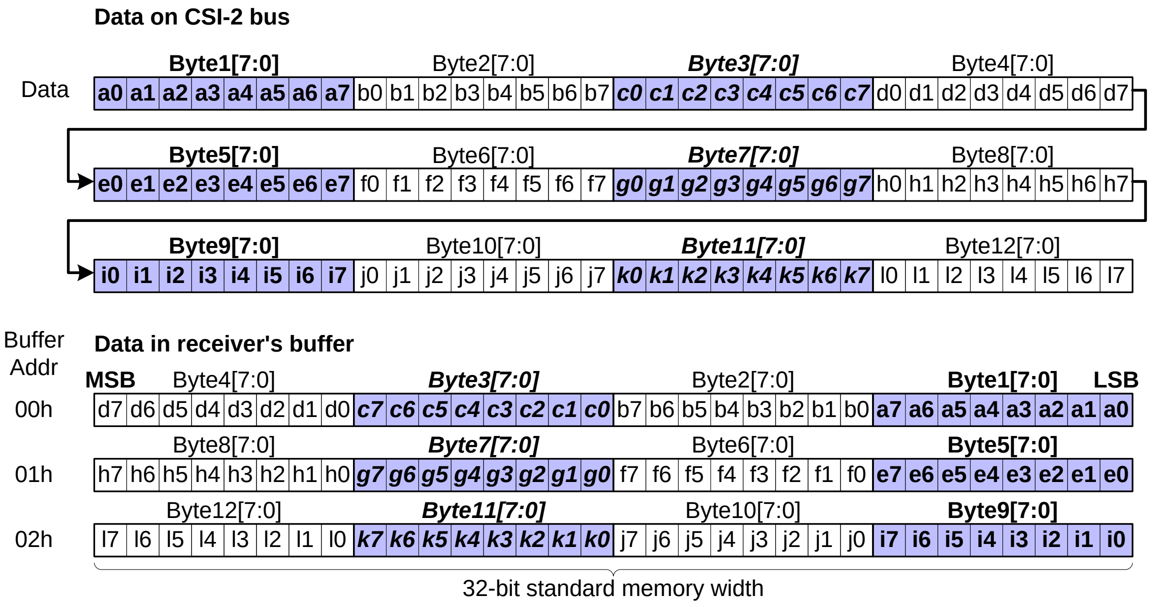
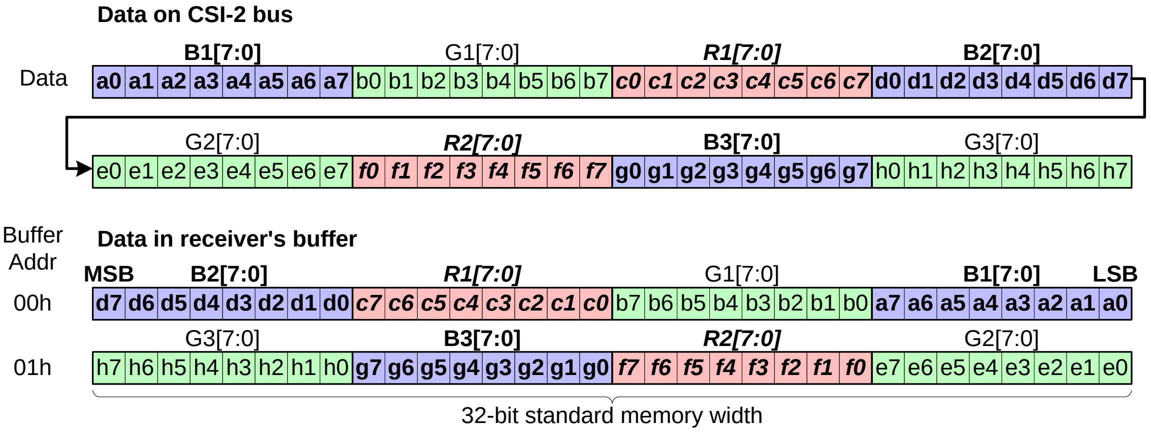

Data on CSI-2 bus
Data
Byte1[7:0] Byte2[7:0] Byte3[7:0] Byte4[7:0]
a0 a1 a2 a3 a4 a5 a6 a7 b0 b1 b2 b3 b4 b5 b6 b7 c0 c1 c2 c3 c4 c5 c6 c7 d0 d1 d2 d3 d4 d5 d6 d7
Byte5[7:0] Byte6[7:0] Byte7[7:0] Byte8[7:0]
e0 e1 e2 e3 e4 e5 e6 e7 f0 f1 f2 f3 f4 f5 f6 f7 g0 g1 g2 g3 g4 g5 g6 g7 h0 h1 h2 h3 h4 h5 h6 h7
Byte9[7:0] Byte10[7:0] Byte11[7:0] Byte12[7:0]
i0 i1 i2 i3 i4 i5 i6 i7 j0 j1 j2 j3 j4 j5 j6 j7 k0 k1 k2 k3 k4 k5 k6 k7 l0 l1 l2 l3 l4 l5 l6 l7
Buffer Addr
Data in receiver's buffer
MSB
LSB
Byte4[7:0] Byte3[7:0] Byte2[7:0] Byte1[7:0]
00h
d7 d6 d5 d4 d3 d2 d1 d0 c7 c6 c5 c4 c3 c2 c1 c0 b7 b6 b5 b4 b3 b2 b1 b0 a7 a6 a5 a4 a3 a2 a1 a0
Byte8[7:0] Byte7[7:0] Byte6[7:0] Byte5[7:0]
01h
h7 h6 h5 h4 h3 h2 h1 h0 g7 g6 g5 g4 g3 g2 g1 g0 f7 f6 f5 f4 f3 f2 f1 f0 e7 e6 e5 e4 e3 e2 e1 e0
Byte12[7:0] Byte11[7:0] Byte10[7:0] Byte9[7:0]
02h
l7 l6 l5 l4 l3 l2 l1 l0 k7 k6 k5 k4 k3 k2 k1 k0 j7 j6 j5 j4 j3 j2 j1 j0 i7 i6 i5 i4 i3 i2 i1 i0
32-bit standard memory width

Figure 154 General/Arbitrary Data Reception

12.2 RGB888 Data Reception

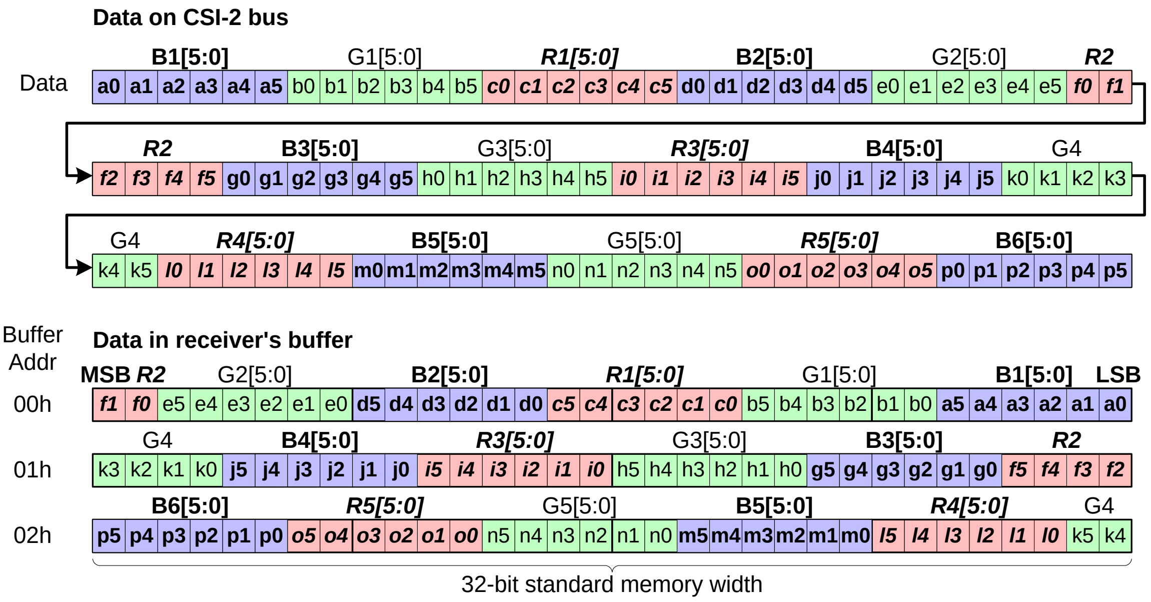
1821 The RGB888 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.


Data on CSI-2 bus
Data
B1[7:0] G1[7:0] R1[7:0] B2[7:0]
a0 a1 a2 a3 a4 a5 a6 a7 b0 b1 b2 b3 b4 b5 b6 b7 c0 c1 c2 c3 c4 c5 c6 c7 d0 d1 d2 d3 d4 d5 d6 d7
G2[7:0] R2[7:0] B3[7:0] G3[7:0]
e0 e1 e2 e3 e4 e5 e6 e7 f0 f1 f2 f3 f4 f5 f6 f7 g0 g1 g2 g3 g4 g5 g6 g7 h0 h1 h2 h3 h4 h5 h6 h7
Buffer Addr
Data in receiver's buffer
MSB B2[7:0] R1[7:0] G1[7:0] B1[7:0] LSB
00h
d7 d6 d5 d4 d3 d2 d1 d0 c7 c6 c5 c4 c3 c2 c1 c0 b7 b6 b5 b4 b3 b2 b1 b0 a7 a6 a5 a4 a3 a2 a1 a0
G3[7:0] B3[7:0] R2[7:0] G2[7:0]
01h
h7 h6 h5 h4 h3 h2 h1 h0 g7 g6 g5 g4 g3 g2 g1 g0 f7 f6 f5 f4 f3 f2 f1 f0 e7 e6 e5 e4 e3 e2 e1 e0
32-bit standard memory width

1822

Figure 155 RGB888 Data Format Reception

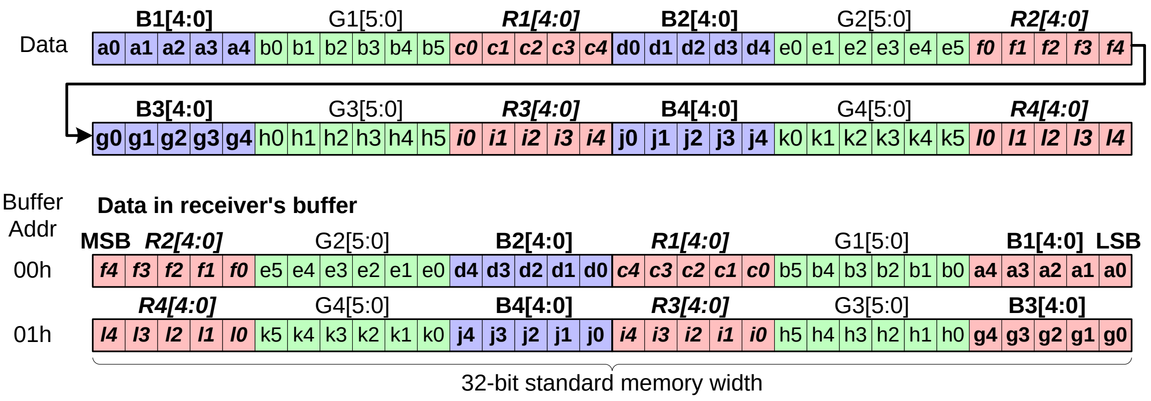
12.3 RGB666 Data Reception


Data on CSI-2 bus
Data
B1[5:0] G1[5:0] R1[5:0] B2[5:0] G2[5:0] R2
a0 a1 a2 a3 a4 a5 b0 b1 b2 b3 b4 b5 c0 c1 c2 c3 c4 c5 d0 d1 d2 d3 d4 d5 e0 e1 e2 e3 e4 e5 f0 f1
R2 B3[5:0] G3[5:0] R3[5:0] B4[5:0] G4
f2 f3 f4 f5 g0 g1 g2 g3 g4 g5 h0 h1 h2 h3 h4 h5 i0 i1 i2 i3 i4 i5 j0 j1 j2 j3 j4 j5 k0 k1 k2 k3
G4 R4[5:0] B5[5:0] G5[5:0] R5[5:0] B6[5:0]
k4 k5 l0 l1 l2 l3 l4 l5 m0 m1 m2 m3 m4 m5 n0 n1 n2 n3 n4 n5 o0 o1 o2 o3 o4 o5 p0 p1 p2 p3 p4 p5
Buffer Addr
Data in receiver's buffer
MSB R2 G2[5:0] B2[5:0] R1[5:0] G1[5:0] B1[5:0] LSB
00h
f1 f0 e5 e4 e3 e2 e1 e0 d5 d4 d3 d2 d1 d0 c5 c4 c3 c2 c1 c0 b5 b4 b3 b2 b1 b0 a5 a4 a3 a2 a1 a0
G4 B4[5:0] R3[5:0] G3[5:0] B3[5:0] R2
01h
k3 k2 k1 k0 j5 j4 j3 j2 j1 j0 i5 i4 i3 i2 i1 i0 h5 h4 h3 h2 h1 h0 g5 g4 g3 g2 g1 g0 f5 f4 f3 f2
B6[5:0] R5[5:0] G5[5:0] B5[5:0] R4[5:0] G4
02h
p5 p4 p3 p2 p1 p0 o5 o4 o3 o2 o1 o0 n5 n4 n3 n2 n1 n0 m5 m4 m3 m2 m1 m0 l5 l4 l3 l2 l1 l0 k5 k4
32-bit standard memory width

1823

Figure 156 RGB666 Data Format Reception

12.4 RGB565 Data Reception



Data on CSI-2 bus

	B1[4:0]					G1[5:0]						R1[4:0]					B2[4:0]					G2[5:0]						R2[4:0]				
Data	a0	a1	a2	a3	a4	b0	b1	b2	b3	b4	b5	*c0*	*c1*	*c2*	*c3*	*c4*	d0	d1	d2	d3	d4	e0	e1	e2	e3	e4	e5	*f0*	*f1*	*f2*	*f3*	*f4*
	B3[4:0]					G3[5:0]						***R3[4:0]***					**B4[4:0]**					G4[5:0]						***R4[4:0]***				
	g0	g1	g2	g3	g4	h0	h1	h2	h3	h4	h5	*i0*	*i1*	*i2*	*i3*	*i4*	j0	j1	j2	j3	j4	k0	k1	k2	k3	k4	k5	*l0*	*l1*	*l2*	*l3*	*l4*

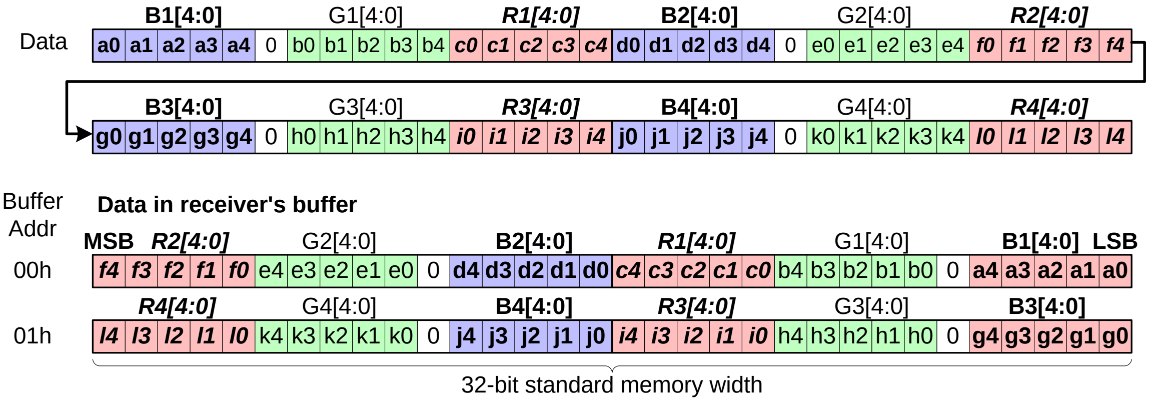
Data in receiver's buffer

Buffer Addr	MSB ***R2[4:0]***					G2[5:0]						**B2[4:0]**					***R1[4:0]***					G1[5:0]						**B1[4:0]** LSB				
00h	*f4*	*f3*	*f2*	*f1*	*f0*	e5	e4	e3	e2	e1	e0	d4	d3	d2	d1	d0	*c4*	*c3*	*c2*	*c1*	*c0*	b5	b4	b3	b2	b1	b0	a4	a3	a2	a1	a0
	R4[4:0]					G4[5:0]						**B4[4:0]**					***R3[4:0]***					G3[5:0]						**B3[4:0]**				
01h	*l4*	*l3*	*l2*	*l1*	*l0*	k5	k4	k3	k2	k1	k0	j4	j3	j2	j1	j0	*i4*	*i3*	*i2*	*i1*	*i0*	h5	h4	h3	h2	h1	h0	g4	g3	g2	g1	g0

32-bit standard memory width

Figure 157 RGB565 Data Format Reception

12.5 RGB555 Data Reception



Data on CSI-2 bus

	B1[4:0]						G1[4:0]					R1[4:0]					B2[4:0]						G2[4:0]					R2[4:0]				
Data	a0	a1	a2	a3	a4	0	b0	b1	b2	b3	b4	*c0*	*c1*	*c2*	*c3*	*c4*	d0	d1	d2	d3	d4	0	e0	e1	e2	e3	e4	*f0*	*f1*	*f2*	*f3*	*f4*
	B3[4:0]						G3[4:0]					***R3[4:0]***					**B4[4:0]**						G4[4:0]					***R4[4:0]***				
	g0	g1	g2	g3	g4	0	h0	h1	h2	h3	h4	*i0*	*i1*	*i2*	*i3*	*i4*	j0	j1	j2	j3	j4	0	k0	k1	k2	k3	k4	*l0*	*l1*	*l2*	*l3*	*l4*

Data in receiver's buffer

Buffer Addr	MSB ***R2[4:0]***					G2[4:0]						**B2[4:0]**					***R1[4:0]***					G1[4:0]						**B1[4:0]** LSB				
00h	*f4*	*f3*	*f2*	*f1*	*f0*	e4	e3	e2	e1	e0	0	d4	d3	d2	d1	d0	*c4*	*c3*	*c2*	*c1*	*c0*	b4	b3	b2	b1	b0	0	a4	a3	a2	a1	a0
	R4[4:0]					G4[4:0]						**B4[4:0]**					***R3[4:0]***					G3[4:0]						**B3[4:0]**				
01h	*l4*	*l3*	*l2*	*l1*	*l0*	k4	k3	k2	k1	k0	0	j4	j3	j2	j1	j0	*i4*	*i3*	*i2*	*i1*	*i0*	h4	h3	h2	h1	h0	0	g4	g3	g2	g1	g0

32-bit standard memory width

Figure 158 RGB555 Data Format Reception

12.6 RGB444 Data Reception

The RGB444 data format byte to 32-bit memory word mapping has a special transform as shown in ***Figure 159***.

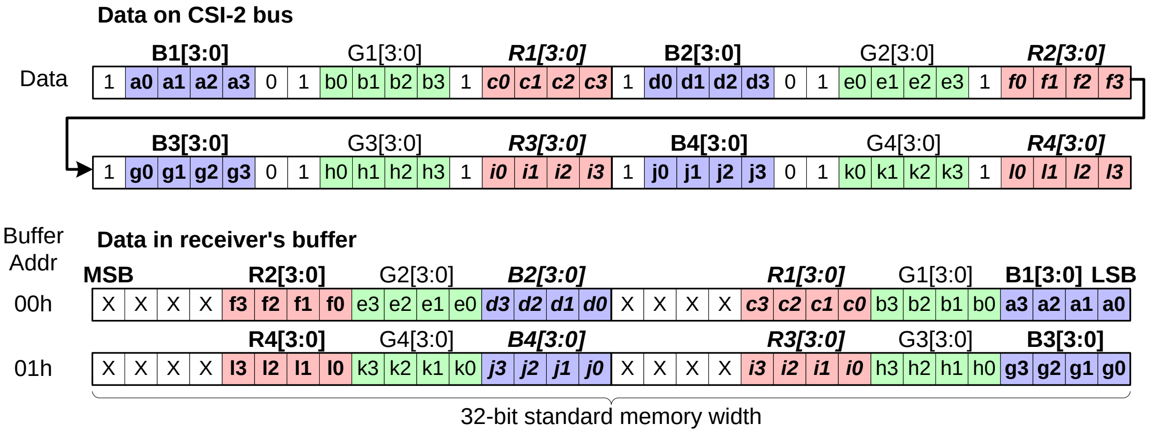

Data on CSI-2 bus
B1[3:0] G1[3:0] R1[3:0] B2[3:0] G2[3:0] R2[3:0]
Data
1 a0 a1 a2 a3 0 1 b0 b1 b2 b3 1 c0 c1 c2 c3 1 d0 d1 d2 d3 0 1 e0 e1 e2 e3 1 f0 f1 f2 f3
B3[3:0] G3[3:0] R3[3:0] B4[3:0] G4[3:0] R4[3:0]
1 g0 g1 g2 g3 0 1 h0 h1 h2 h3 1 i0 i1 i2 i3 1 j0 j1 j2 j3 0 1 k0 k1 k2 k3 1 l0 l1 l2 l3
Buffer Addr
Data in receiver's buffer
MSB R2[3:0] G2[3:0] B2[3:0] R1[3:0] G1[3:0] B1[3:0] LSB
00h
X X X X f3 f2 f1 f0 e3 e2 e1 e0 d3 d2 d1 d0 X X X X c3 c2 c1 c0 b3 b2 b1 b0 a3 a2 a1 a0
R4[3:0] G4[3:0] B4[3:0] R3[3:0] G3[3:0] B3[3:0]
01h
X X X X l3 l2 l1 l0 k3 k2 k1 k0 j3 j2 j1 j0 X X X X i3 i2 i1 i0 h3 h2 h1 h0 g3 g2 g1 g0
32-bit standard memory width

Figure 159 RGB444 Data Format Reception

12.7 YUV422 8-bit Data Reception

The YUV422 8-bit data format the byte to 32-bit memory word mapping does not follow the generic CSI-2 rule.

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.

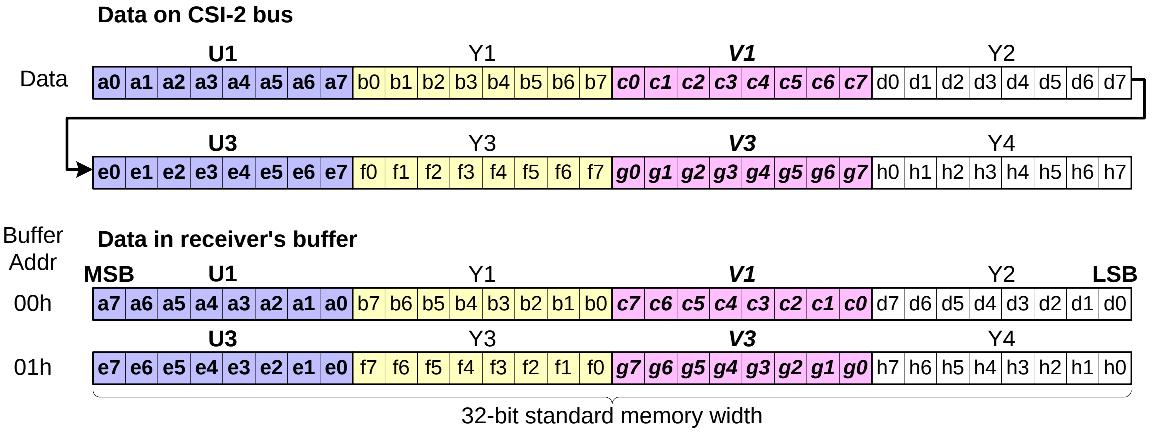
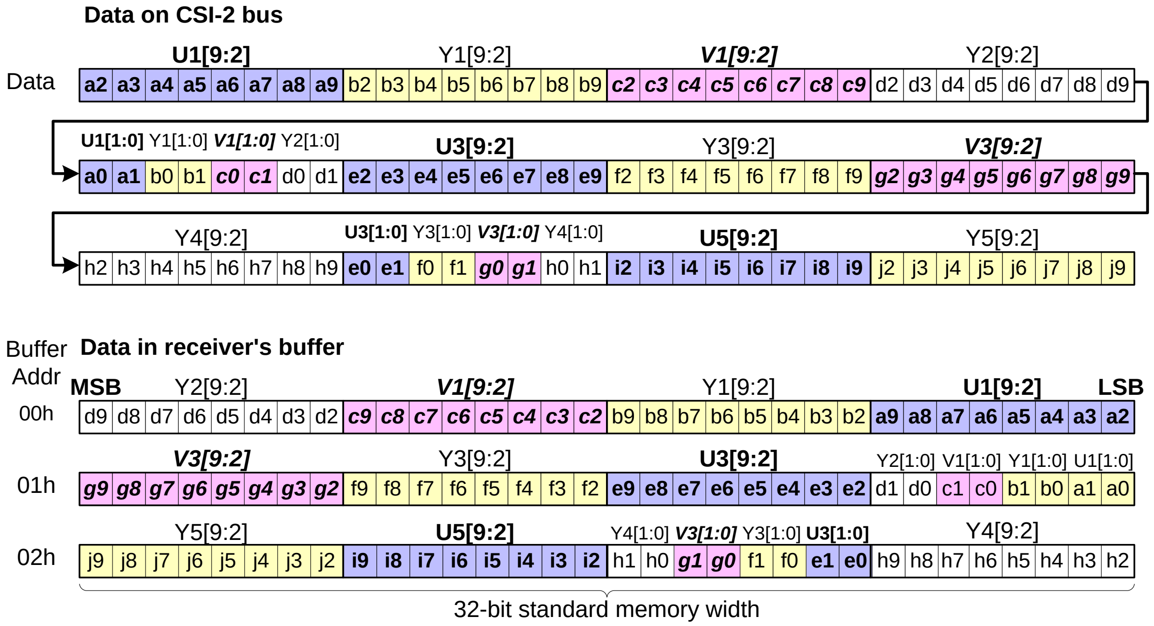

Data on CSI-2 bus
U1 Y1 V1 Y2
Data
a0 a1 a2 a3 a4 a5 a6 a7 b0 b1 b2 b3 b4 b5 b6 b7 c0 c1 c2 c3 c4 c5 c6 c7 d0 d1 d2 d3 d4 d5 d6 d7
U3 Y3 V3 Y4
e0 e1 e2 e3 e4 e5 e6 e7 f0 f1 f2 f3 f4 f5 f6 f7 g0 g1 g2 g3 g4 g5 g6 g7 h0 h1 h2 h3 h4 h5 h6 h7
Buffer Addr
Data in receiver's buffer
MSB U1 Y1 V1 Y2 LSB
00h
a7 a6 a5 a4 a3 a2 a1 a0 b7 b6 b5 b4 b3 b2 b1 b0 c7 c6 c5 c4 c3 c2 c1 c0 d7 d6 d5 d4 d3 d2 d1 d0
U3 Y3 V3 Y4
01h
e7 e6 e5 e4 e3 e2 e1 e0 f7 f6 f5 f4 f3 f2 f1 f0 g7 g6 g5 g4 g3 g2 g1 g0 h7 h6 h5 h4 h3 h2 h1 h0
32-bit standard memory width

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.


Data on CSI-2 bus
Data
U1[9:2] Y1[9:2] V1[9:2] Y2[9:2]
a2 a3 a4 a5 a6 a7 a8 a9 b2 b3 b4 b5 b6 b7 b8 b9 c2 c3 c4 c5 c6 c7 c8 c9 d2 d3 d4 d5 d6 d7 d8 d9
U1[1:0] Y1[1:0] V1[1:0] Y2[1:0] U3[9:2] Y3[9:2] V3[9:2]
a0 a1 b0 b1 c0 c1 d0 d1 e2 e3 e4 e5 e6 e7 e8 e9 f2 f3 f4 f5 f6 f7 f8 f9 g2 g3 g4 g5 g6 g7 g8 g9
Y4[9:2] U3[1:0] Y3[1:0] V3[1:0] Y4[1:0] U5[9:2] Y5[9:2]
h2 h3 h4 h5 h6 h7 h8 h9 e0 e1 f0 f1 g0 g1 h0 h1 i2 i3 i4 i5 i6 i7 i8 i9 j2 j3 j4 j5 j6 j7 j8 j9
Buffer Addr
Data in receiver's buffer
MSB Y2[9:2] V1[9:2] Y1[9:2] U1[9:2] LSB
00h
d9 d8 d7 d6 d5 d4 d3 d2 c9 c8 c7 c6 c5 c4 c3 c2 b9 b8 b7 b6 b5 b4 b3 b2 a9 a8 a7 a6 a5 a4 a3 a2
V3[9:2] Y3[9:2] U3[9:2] Y2[1:0] V1[1:0] Y1[1:0] U1[1:0]
01h
g9 g8 g7 g6 g5 g4 g3 g2 f9 f8 f7 f6 f5 f4 f3 f2 e9 e8 e7 e6 e5 e4 e3 e2 d1 d0 c1 c0 b1 b0 a1 a0
Y5[9:2] U5[9:2] Y4[1:0] V3[1:0] Y3[1:0] U3[1:0] Y4[9:2]
02h
j9 j8 j7 j6 j5 j4 j3 j2 i9 i8 i7 i6 i5 i4 i3 i2 h1 h0 g1 g0 f1 f0 e1 e0 h9 h8 h7 h6 h5 h4 h3 h2
32-bit standard memory width

1836

Figure 161 YUV422 10-bit Data Format Reception

12.9 YUV420 8-bit (Legacy) Data Reception

The YUV420 8-bit (legacy) data format the byte to 32-bit memory word mapping does not follow the generic CSI-2 rule.

For YUV422 8-bit (legacy) 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.

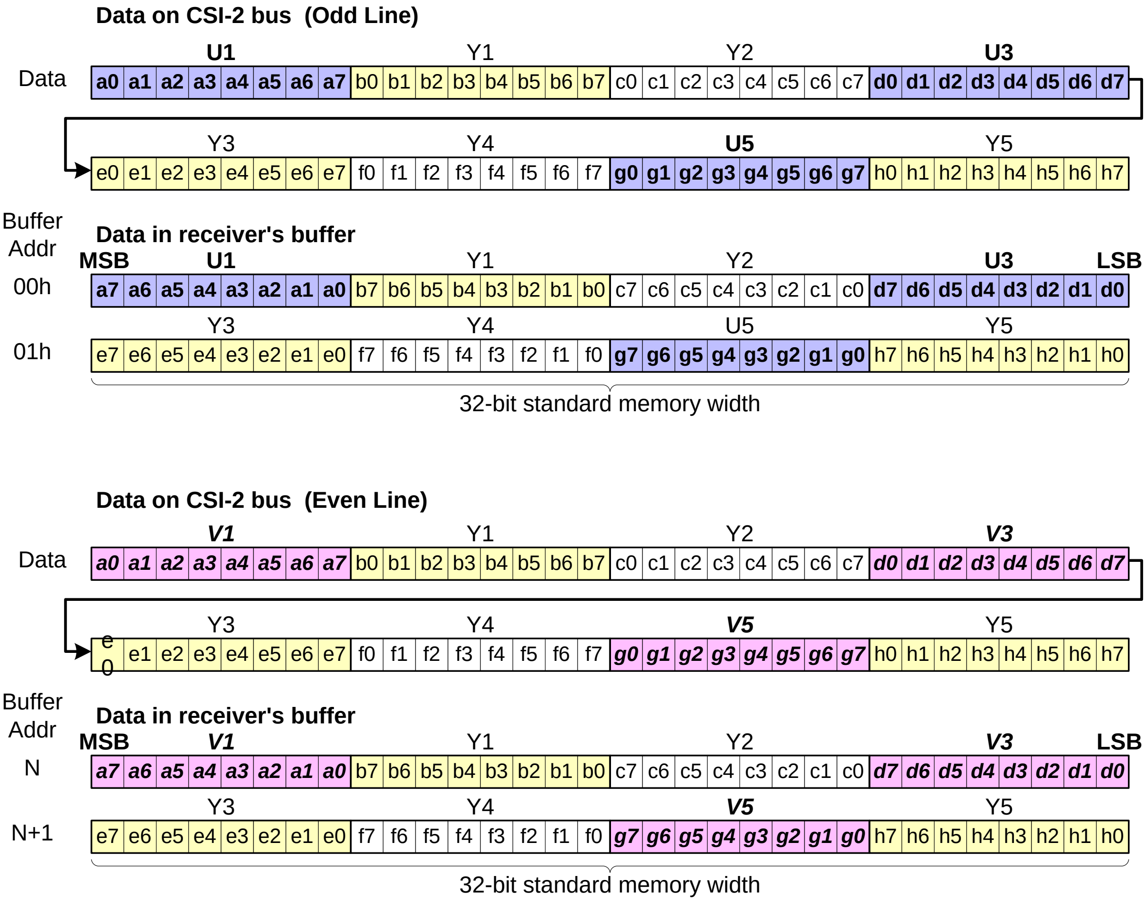

Data on CSI-2 bus (Odd Line)
Data
U1 Y1 Y2 U3
a0 a1 a2 a3 a4 a5 a6 a7 b0 b1 b2 b3 b4 b5 b6 b7 c0 c1 c2 c3 c4 c5 c6 c7 d0 d1 d2 d3 d4 d5 d6 d7
Y3 Y4 U5 Y5
e0 e1 e2 e3 e4 e5 e6 e7 f0 f1 f2 f3 f4 f5 f6 f7 g0 g1 g2 g3 g4 g5 g6 g7 h0 h1 h2 h3 h4 h5 h6 h7
Buffer Addr
Data in receiver's buffer
MSB U1 Y1 Y2 U3 LSB
00h
a7 a6 a5 a4 a3 a2 a1 a0 b7 b6 b5 b4 b3 b2 b1 b0 c7 c6 c5 c4 c3 c2 c1 c0 d7 d6 d5 d4 d3 d2 d1 d0
Y3 Y4 U5 Y5
01h
e7 e6 e5 e4 e3 e2 e1 e0 f7 f6 f5 f4 f3 f2 f1 f0 g7 g6 g5 g4 g3 g2 g1 g0 h7 h6 h5 h4 h3 h2 h1 h0
32-bit standard memory width
Data on CSI-2 bus (Even Line)
Data
V1 Y1 Y2 V3
a0 a1 a2 a3 a4 a5 a6 a7 b0 b1 b2 b3 b4 b5 b6 b7 c0 c1 c2 c3 c4 c5 c6 c7 d0 d1 d2 d3 d4 d5 d6 d7
Y3 Y4 V5 Y5
e0 e1 e2 e3 e4 e5 e6 e7 f0 f1 f2 f3 f4 f5 f6 f7 g0 g1 g2 g3 g4 g5 g6 g7 h0 h1 h2 h3 h4 h5 h6 h7
Buffer Addr
Data in receiver's buffer
MSB V1 Y1 Y2 V3 LSB
N
a7 a6 a5 a4 a3 a2 a1 a0 b7 b6 b5 b4 b3 b2 b1 b0 c7 c6 c5 c4 c3 c2 c1 c0 d7 d6 d5 d4 d3 d2 d1 d0
Y3 Y4 V5 Y5
N+1
e7 e6 e5 e4 e3 e2 e1 e0 f7 f6 f5 f4 f3 f2 f1 f0 g7 g6 g5 g4 g3 g2 g1 g0 h7 h6 h5 h4 h3 h2 h1 h0
32-bit standard memory width

Figure 162 YUV420 8-bit Legacy Data Format Reception

12.10 YUV420 8-bit Data Reception

The YUV420 8-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

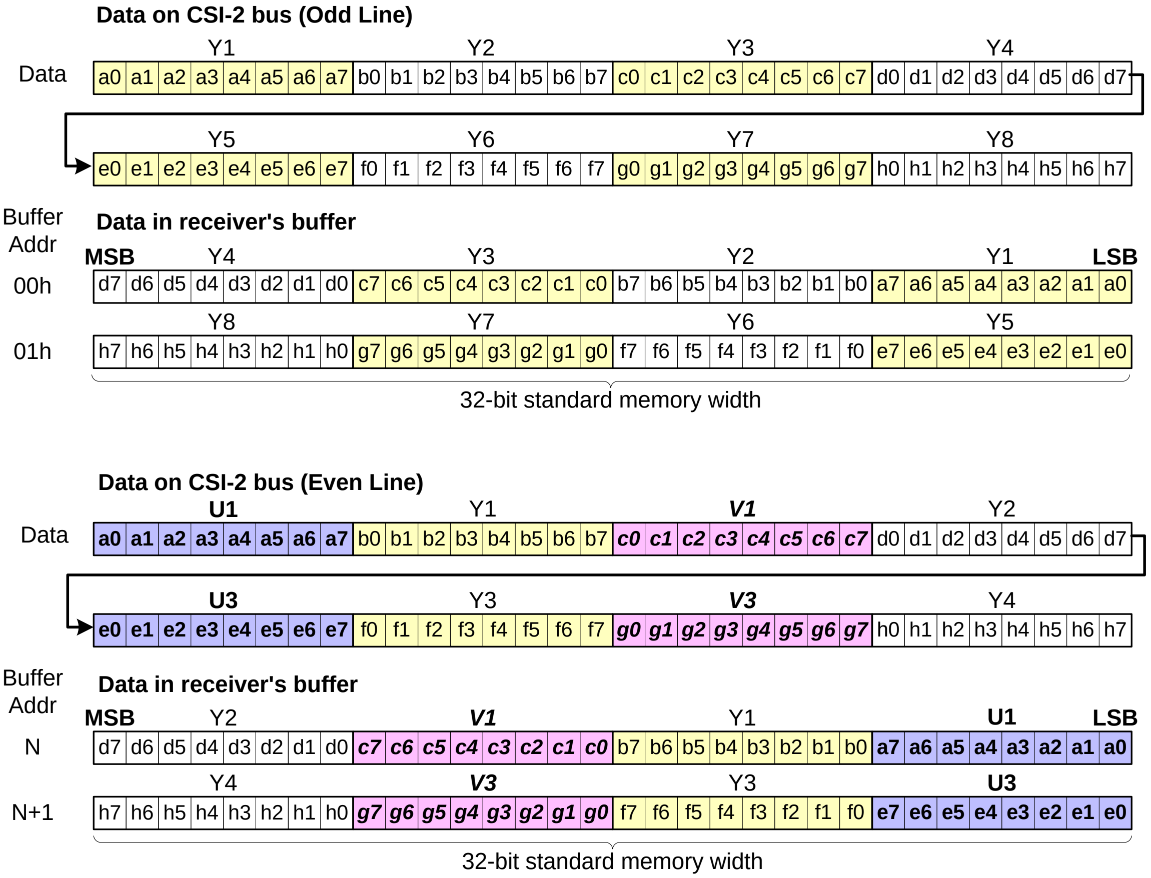

Data on CSI-2 bus (Odd Line)
Data
Y1 Y2 Y3 Y4
a0 a1 a2 a3 a4 a5 a6 a7 b0 b1 b2 b3 b4 b5 b6 b7 c0 c1 c2 c3 c4 c5 c6 c7 d0 d1 d2 d3 d4 d5 d6 d7
Y5 Y6 Y7 Y8
e0 e1 e2 e3 e4 e5 e6 e7 f0 f1 f2 f3 f4 f5 f6 f7 g0 g1 g2 g3 g4 g5 g6 g7 h0 h1 h2 h3 h4 h5 h6 h7
Buffer Addr
Data in receiver's buffer
MSB Y4 Y3 Y2 Y1 LSB
00h
d7 d6 d5 d4 d3 d2 d1 d0 c7 c6 c5 c4 c3 c2 c1 c0 b7 b6 b5 b4 b3 b2 b1 b0 a7 a6 a5 a4 a3 a2 a1 a0
Y8 Y7 Y6 Y5
01h
h7 h6 h5 h4 h3 h2 h1 h0 g7 g6 g5 g4 g3 g2 g1 g0 f7 f6 f5 f4 f3 f2 f1 f0 e7 e6 e5 e4 e3 e2 e1 e0
32-bit standard memory width
Data on CSI-2 bus (Even Line)
Data
U1 Y1 V1 Y2
a0 a1 a2 a3 a4 a5 a6 a7 b0 b1 b2 b3 b4 b5 b6 b7 c0 c1 c2 c3 c4 c5 c6 c7 d0 d1 d2 d3 d4 d5 d6 d7
U3 Y3 V3 Y4
e0 e1 e2 e3 e4 e5 e6 e7 f0 f1 f2 f3 f4 f5 f6 f7 g0 g1 g2 g3 g4 g5 g6 g7 h0 h1 h2 h3 h4 h5 h6 h7
Buffer Addr
Data in receiver's buffer
MSB Y2 V1 Y1 U1 LSB
N
d7 d6 d5 d4 d3 d2 d1 d0 c7 c6 c5 c4 c3 c2 c1 c0 b7 b6 b5 b4 b3 b2 b1 b0 a7 a6 a5 a4 a3 a2 a1 a0
Y4 V3 Y3 U3
N+1
h7 h6 h5 h4 h3 h2 h1 h0 g7 g6 g5 g4 g3 g2 g1 g0 f7 f6 f5 f4 f3 f2 f1 f0 e7 e6 e5 e4 e3 e2 e1 e0
32-bit standard memory width

Figure 163 YUV420 8-bit Data Format Reception

12.11 YUV420 10-bit Data Reception

The YUV420 10-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

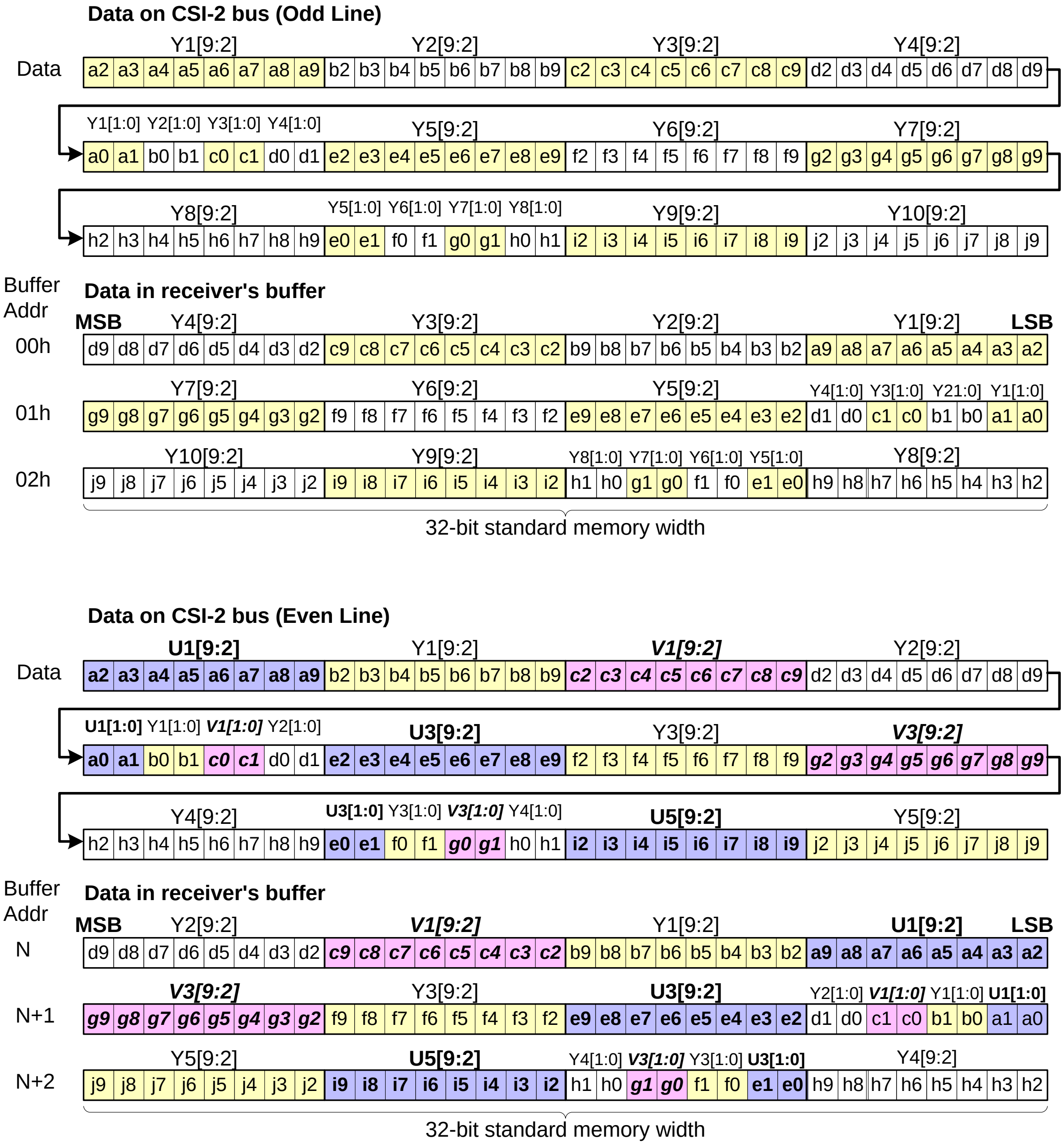

Data on CSI-2 bus (Odd Line)
Data
Y1[9:2] Y2[9:2] Y3[9:2] Y4[9:2]
a2 a3 a4 a5 a6 a7 a8 a9 b2 b3 b4 b5 b6 b7 b8 b9 c2 c3 c4 c5 c6 c7 c8 c9 d2 d3 d4 d5 d6 d7 d8 d9
Y1[1:0] Y2[1:0] Y3[1:0] Y4[1:0] Y5[9:2] Y6[9:2] Y7[9:2]
a0 a1 b0 b1 c0 c1 d0 d1 e2 e3 e4 e5 e6 e7 e8 e9 f2 f3 f4 f5 f6 f7 f8 f9 g2 g3 g4 g5 g6 g7 g8 g9
Y8[9:2] Y5[1:0] Y6[1:0] Y7[1:0] Y8[1:0] Y9[9:2] Y10[9:2]
h2 h3 h4 h5 h6 h7 h8 h9 e0 e1 f0 f1 g0 g1 h0 h1 i2 i3 i4 i5 i6 i7 i8 i9 j2 j3 j4 j5 j6 j7 j8 j9
Buffer Addr
Data in receiver's buffer
MSB Y4[9:2] Y3[9:2] Y2[9:2] Y1[9:2] LSB
00h d9 d8 d7 d6 d5 d4 d3 d2 c9 c8 c7 c6 c5 c4 c3 c2 b9 b8 b7 b6 b5 b4 b3 b2 a9 a8 a7 a6 a5 a4 a3 a2
Y7[9:2] Y6[9:2] Y5[9:2] Y4[1:0] Y3[1:0] Y21:0] Y1[1:0]
01h g9 g8 g7 g6 g5 g4 g3 g2 f9 f8 f7 f6 f5 f4 f3 f2 e9 e8 e7 e6 e5 e4 e3 e2 d1 d0 c1 c0 b1 b0 a1 a0
Y10[9:2] Y9[9:2] Y8[1:0] Y7[1:0] Y6[1:0] Y5[1:0] Y8[9:2]
02h j9 j8 j7 j6 j5 j4 j3 j2 i9 i8 i7 i6 i5 i4 i3 i2 h1 h0 g1 g0 f1 f0 e1 e0 h9 h8 h7 h6 h5 h4 h3 h2
32-bit standard memory width
Data on CSI-2 bus (Even Line)
Data
U1[9:2] Y1[9:2] V1[9:2] Y2[9:2]
a2 a3 a4 a5 a6 a7 a8 a9 b2 b3 b4 b5 b6 b7 b8 b9 c2 c3 c4 c5 c6 c7 c8 c9 d2 d3 d4 d5 d6 d7 d8 d9
U1[1:0] Y1[1:0] V1[1:0] Y2[1:0] U3[9:2] Y3[9:2] V3[9:2]
a0 a1 b0 b1 c0 c1 d0 d1 e2 e3 e4 e5 e6 e7 e8 e9 f2 f3 f4 f5 f6 f7 f8 f9 g2 g3 g4 g5 g6 g7 g8 g9
Y4[9:2] U3[1:0] Y3[1:0] V3[1:0] Y4[1:0] U5[9:2] Y5[9:2]
h2 h3 h4 h5 h6 h7 h8 h9 e0 e1 f0 f1 g0 g1 h0 h1 i2 i3 i4 i5 i6 i7 i8 i9 j2 j3 j4 j5 j6 j7 j8 j9
Buffer Addr
Data in receiver's buffer
MSB Y2[9:2] V1[9:2] Y1[9:2] U1[9:2] LSB
N d9 d8 d7 d6 d5 d4 d3 d2 c9 c8 c7 c6 c5 c4 c3 c2 b9 b8 b7 b6 b5 b4 b3 b2 a9 a8 a7 a6 a5 a4 a3 a2
V3[9:2] Y3[9:2] U3[9:2] Y2[1:0] V1[1:0] Y1[1:0] U1[1:0]
N+1 g9 g8 g7 g6 g5 g4 g3 g2 f9 f8 f7 f6 f5 f4 f3 f2 e9 e8 e7 e6 e5 e4 e3 e2 d1 d0 c1 c0 b1 b0 a1 a0
Y5[9:2] U5[9:2] Y4[1:0] V3[1:0] Y3[1:0] U3[1:0] Y4[9:2]
N+2 j9 j8 j7 j6 j5 j4 j3 j2 i9 i8 i7 i6 i5 i4 i3 i2 h1 h0 g1 g0 f1 f0 e1 e0 h9 h8 h7 h6 h5 h4 h3 h2
32-bit standard memory width

Figure 164 YUV420 10-bit Data Format Reception

12.12 RAW6 Data Reception

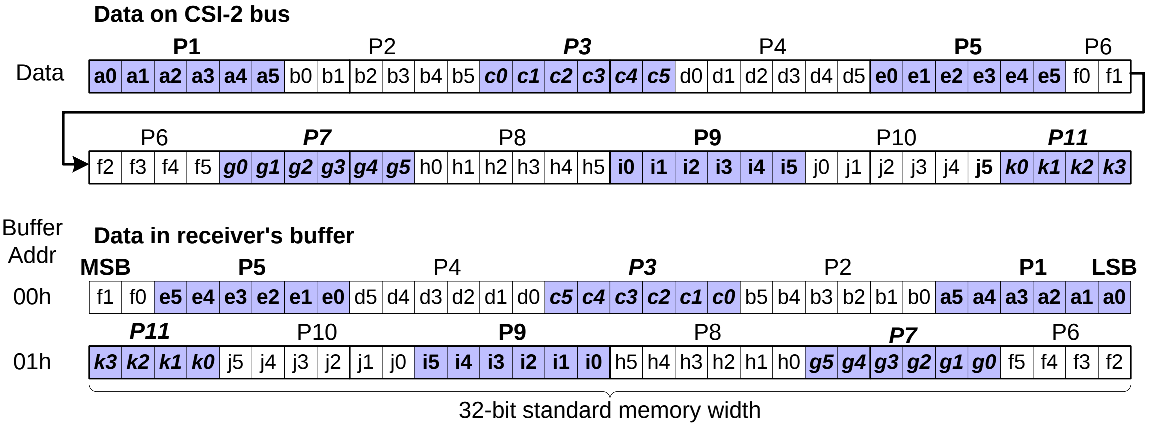

Data on CSI-2 bus
Data
P1 P2 P3 P4 P5 P6
a0 a1 a2 a3 a4 a5 b0 b1 b2 b3 b4 b5 c0 c1 c2 c3 c4 c5 d0 d1 d2 d3 d4 d5 e0 e1 e2 e3 e4 e5 f0 f1
P6 P7 P8 P9 P10 P11
f2 f3 f4 f5 g0 g1 g2 g3 g4 g5 h0 h1 h2 h3 h4 h5 i0 i1 i2 i3 i4 i5 j0 j1 j2 j3 j4 j5 k0 k1 k2 k3
Buffer Addr
Data in receiver's buffer
MSB P5 P4 P3 P2 P1 LSB
00h
f1 f0 e5 e4 e3 e2 e1 e0 d5 d4 d3 d2 d1 d0 c5 c4 c3 c2 c1 c0 b5 b4 b3 b2 b1 b0 a5 a4 a3 a2 a1 a0
P11 P10 P9 P8 P7 P6
01h
k3 k2 k1 k0 j5 j4 j3 j2 j1 j0 i5 i4 i3 i2 i1 i0 h5 h4 h3 h2 h1 h0 g5 g4 g3 g2 g1 g0 f5 f4 f3 f2
32-bit standard memory width

Figure 165 RAW6 Data Format Reception

12.13 RAW7 Data Reception

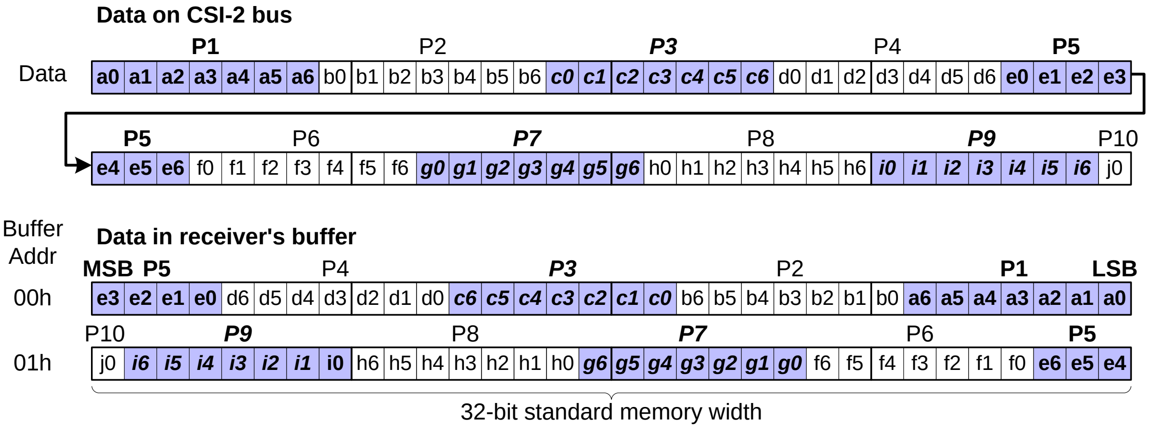
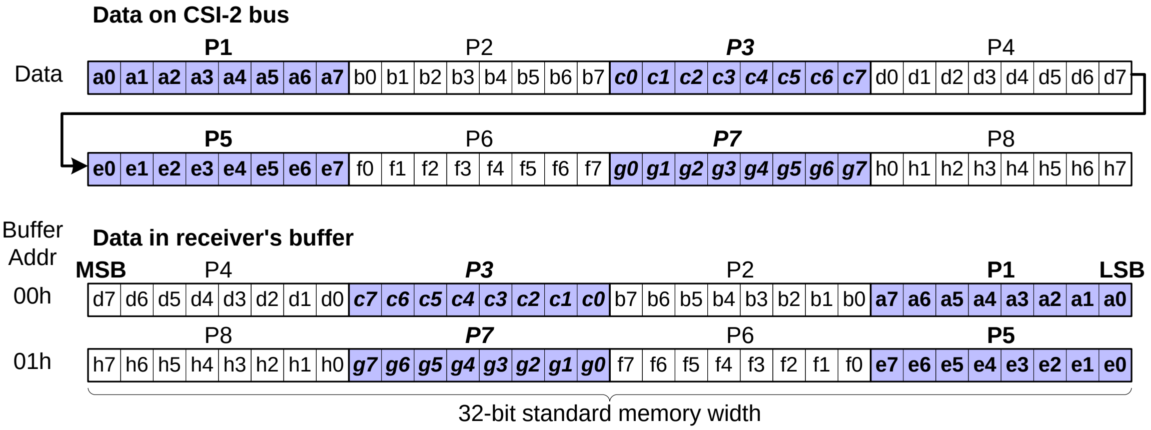

Data on CSI-2 bus
Data
P1 P2 P3 P4 P5
a0 a1 a2 a3 a4 a5 a6 b0 b1 b2 b3 b4 b5 b6 c0 c1 c2 c3 c4 c5 c6 d0 d1 d2 d3 d4 d5 d6 e0 e1 e2 e3
P5 P6 P7 P8 P9 P10
e4 e5 e6 f0 f1 f2 f3 f4 f5 f6 g0 g1 g2 g3 g4 g5 g6 h0 h1 h2 h3 h4 h5 h6 i0 i1 i2 i3 i4 i5 i6 j0
Buffer Addr
Data in receiver's buffer
MSB P5 P4 P3 P2 P1 LSB
00h
e3 e2 e1 e0 d6 d5 d4 d3 d2 d1 d0 c6 c5 c4 c3 c2 c1 c0 b6 b5 b4 b3 b2 b1 b0 a6 a5 a4 a3 a2 a1 a0
P10 P9 P8 P7 P6 P5
01h
j0 i6 i5 i4 i3 i2 i1 i0 h6 h5 h4 h3 h2 h1 h0 g6 g5 g4 g3 g2 g1 g0 f6 f5 f4 f3 f2 f1 f0 e6 e5 e4
32-bit standard memory width

Figure 166 RAW7 Data Format Reception

12.14 RAW8 Data Reception

The RAW8 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.



Data on CSI-2 bus

	P1								P2								P3								P4							
Data	a0	a1	a2	a3	a4	a5	a6	a7	b0	b1	b2	b3	b4	b5	b6	b7	c0	c1	c2	c3	c4	c5	c6	c7	d0	d1	d2	d3	d4	d5	d6	d7
	P5								P6								P7								P8							
	e0	e1	e2	e3	e4	e5	e6	e7	f0	f1	f2	f3	f4	f5	f6	f7	g0	g1	g2	g3	g4	g5	g6	g7	h0	h1	h2	h3	h4	h5	h6	h7

Data in receiver's buffer

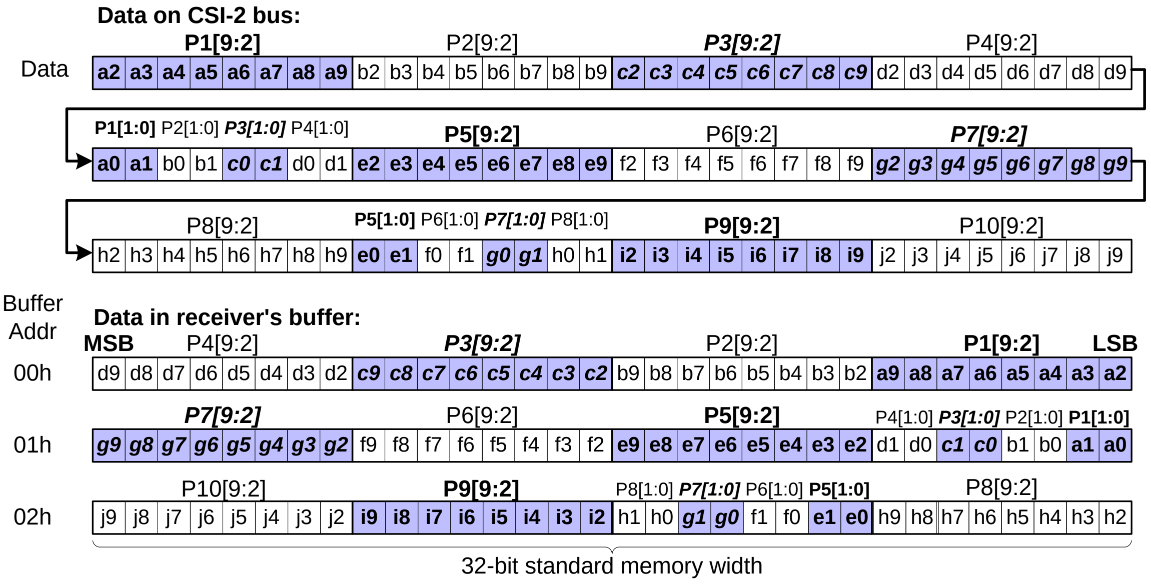
Buffer Addr	MSB P4								P3								P2								P1							LSB
00h	d7	d6	d5	d4	d3	d2	d1	d0	c7	c6	c5	c4	c3	c2	c1	c0	b7	b6	b5	b4	b3	b2	b1	b0	a7	a6	a5	a4	a3	a2	a1	a0
	P8								P7								P6								P5							
01h	h7	h6	h5	h4	h3	h2	h1	h0	g7	g6	g5	g4	g3	g2	g1	g0	f7	f6	f5	f4	f3	f2	f1	f0	e7	e6	e5	e4	e3	e2	e1	e0

32-bit standard memory width

Figure 167 RAW8 Data Format Reception

12.15 RAW10 Data Reception

The RAW10 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.



Data on CSI-2 bus:

	P1[9:2]								P2[9:2]								P3[9:2]								P4[9:2]							
Data	a2	a3	a4	a5	a6	a7	a8	a9	b2	b3	b4	b5	b6	b7	b8	b9	c2	c3	c4	c5	c6	c7	c8	c9	d2	d3	d4	d5	d6	d7	d8	d9
	P1[1:0]		P2[1:0]		P3[1:0]		P4[1:0]		P5[9:2]								P6[9:2]								P7[9:2]							
	a0	a1	b0	b1	c0	c1	d0	d1	e2	e3	e4	e5	e6	e7	e8	e9	f2	f3	f4	f5	f6	f7	f8	f9	g2	g3	g4	g5	g6	g7	g8	g9
	P8[9:2]								P5[1:0]		P6[1:0]		P7[1:0]		P8[1:0]		P9[9:2]								P10[9:2]							
	h2	h3	h4	h5	h6	h7	h8	h9	e0	e1	f0	f1	g0	g1	h0	h1	i2	i3	i4	i5	i6	i7	i8	i9	j2	j3	j4	j5	j6	j7	j8	j9

Data in receiver's buffer:

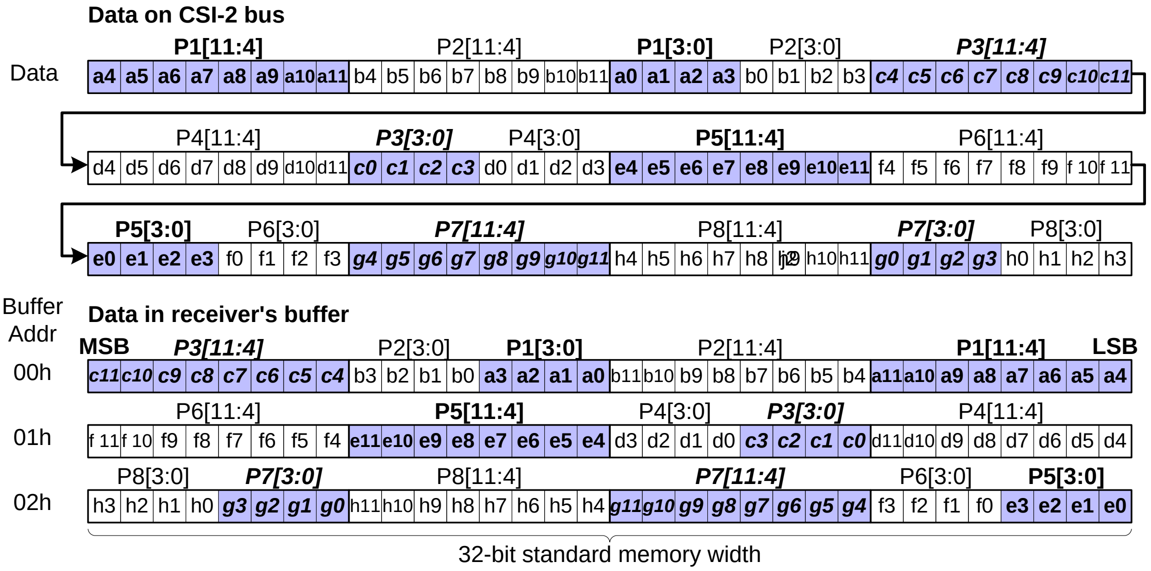
Buffer Addr	MSB P4[9:2]								P3[9:2]								P2[9:2]								P1[9:2]							LSB
00h	d9	d8	d7	d6	d5	d4	d3	d2	c9	c8	c7	c6	c5	c4	c3	c2	b9	b8	b7	b6	b5	b4	b3	b2	a9	a8	a7	a6	a5	a4	a3	a2
	P7[9:2]								P6[9:2]								P5[9:2]								P4[1:0]		P3[1:0]		P2[1:0]		P1[1:0]	
01h	g9	g8	g7	g6	g5	g4	g3	g2	f9	f8	f7	f6	f5	f4	f3	f2	e9	e8	e7	e6	e5	e4	e3	e2	d1	d0	c1	c0	b1	b0	a1	a0
	P10[9:2]								P9[9:2]								P8[1:0]		P7[1:0]		P6[1:0]		P5[1:0]		P8[9:2]							
02h	j9	j8	j7	j6	j5	j4	j3	j2	i9	i8	i7	i6	i5	i4	i3	i2	h1	h0	g1	g0	f1	f0	e1	e0	h9	h8	h7	h6	h5	h4	h3	h2

32-bit standard memory width

Figure 168 RAW10 Data Format Reception

12.16 RAW12 Data Reception

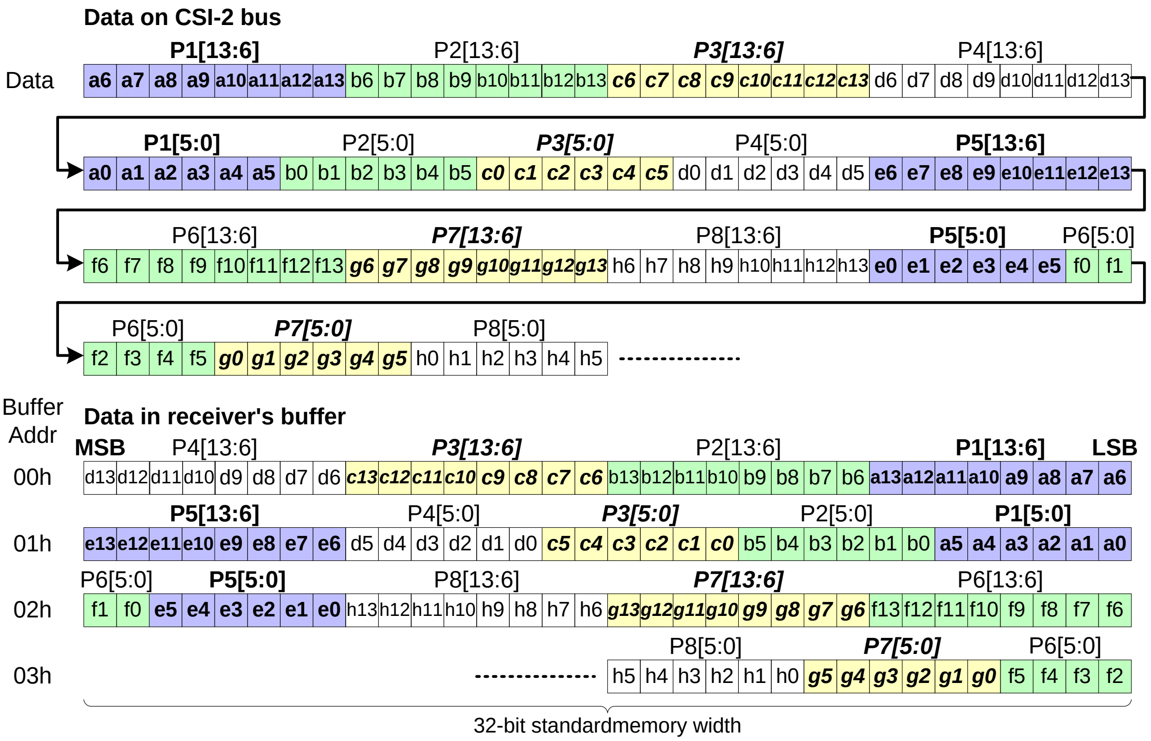
1853 The RAW12 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.


Data on CSI-2 bus
Data
P1[11:4] P2[11:4] P1[3:0] P2[3:0] P3[11:4]
a4 a5 a6 a7 a8 a9 a10 a11 b4 b5 b6 b7 b8 b9 b10 b11 a0 a1 a2 a3 b0 b1 b2 b3 c4 c5 c6 c7 c8 c9 c10 c11
P4[11:4] P3[3:0] P4[3:0] P5[11:4] P6[11:4]
d4 d5 d6 d7 d8 d9 d10 d11 c0 c1 c2 c3 d0 d1 d2 d3 e4 e5 e6 e7 e8 e9 e10 e11 f4 f5 f6 f7 f8 f9 f 10 f 11
P5[3:0] P6[3:0] P7[11:4] P8[11:4] P7[3:0] P8[3:0]
e0 e1 e2 e3 f0 f1 f2 f3 g4 g5 g6 g7 g8 g9 g10 g11 h4 h5 h6 h7 h8 h9 h10 h11 g0 g1 g2 g3 h0 h1 h2 h3
Buffer Addr
Data in receiver's buffer
MSB LSB
00h
P3[11:4] P2[3:0] P1[3:0] P2[11:4] P1[11:4]
c11 c10 c9 c8 c7 c6 c5 c4 b3 b2 b1 b0 a3 a2 a1 a0 b11 b10 b9 b8 b7 b6 b5 b4 a11 a10 a9 a8 a7 a6 a5 a4
01h
P6[11:4] P5[11:4] P4[3:0] P3[3:0] P4[11:4]
f 11 f 10 f9 f8 f7 f6 f5 f4 e11 e10 e9 e8 e7 e6 e5 e4 d3 d2 d1 d0 c3 c2 c1 c0 d11 d10 d9 d8 d7 d6 d5 d4
02h
P8[3:0] P7[3:0] P8[11:4] P7[11:4] P6[3:0] P5[3:0]
h3 h2 h1 h0 g3 g2 g1 g0 h11 h10 h9 h8 h7 h6 h5 h4 g11 g10 g9 g8 g7 g6 g5 g4 f3 f2 f1 f0 e3 e2 e1 e0
32-bit standard memory width

1854

Figure 169 RAW12 Data Format Reception

12.17 RAW14 Data Reception


Data on CSI-2 bus
Data
P1[13:6] P2[13:6] P3[13:6] P4[13:6]
a6 a7 a8 a9 a10 a11 a12 a13 b6 b7 b8 b9 b10 b11 b12 b13 c6 c7 c8 c9 c10 c11 c12 c13 d6 d7 d8 d9 d10 d11 d12 d13
P1[5:0] P2[5:0] P3[5:0] P4[5:0] P5[13:6]
a0 a1 a2 a3 a4 a5 b0 b1 b2 b3 b4 b5 c0 c1 c2 c3 c4 c5 d0 d1 d2 d3 d4 d5 e6 e7 e8 e9 e10 e11 e12 e13
P6[13:6] P7[13:6] P8[13:6] P5[5:0] P6[5:0]
f6 f7 f8 f9 f10 f11 f12 f13 g6 g7 g8 g9 g10 g11 g12 g13 h6 h7 h8 h9 h10 h11 h12 h13 e0 e1 e2 e3 e4 e5 f0 f1
P6[5:0] P7[5:0] P8[5:0]
f2 f3 f4 f5 g0 g1 g2 g3 g4 g5 h0 h1 h2 h3 h4 h5 ---------------
Buffer Addr
Data in receiver's buffer
MSB LSB
00h
P4[13:6] P3[13:6] P2[13:6] P1[13:6]
d13 d12 d11 d10 d9 d8 d7 d6 c13 c12 c11 c10 c9 c8 c7 c6 b13 b12 b11 b10 b9 b8 b7 b6 a13 a12 a11 a10 a9 a8 a7 a6
01h
P5[13:6] P4[5:0] P3[5:0] P2[5:0] P1[5:0]
e13 e12 e11 e10 e9 e8 e7 e6 d5 d4 d3 d2 d1 d0 c5 c4 c3 c2 c1 c0 b5 b4 b3 b2 b1 b0 a5 a4 a3 a2 a1 a0
02h
P6[5:0] P5[5:0] P8[13:6] P7[13:6] P6[13:6]
f1 f0 e5 e4 e3 e2 e1 e0 h13 h12 h11 h10 h9 h8 h7 h6 g13 g12 g11 g10 g9 g8 g7 g6 f13 f12 f11 f10 f9 f8 f7 f6
03h
P8[5:0] P7[5:0] P6[5:0]
--------------- h5 h4 h3 h2 h1 h0 g5 g4 g3 g2 g1 g0 f5 f4 f3 f2
32-bit standardmemory width

1855

Figure 170 RAW 14 Data Format Reception

12.18 RAW16 Data Reception

The RAW16 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.

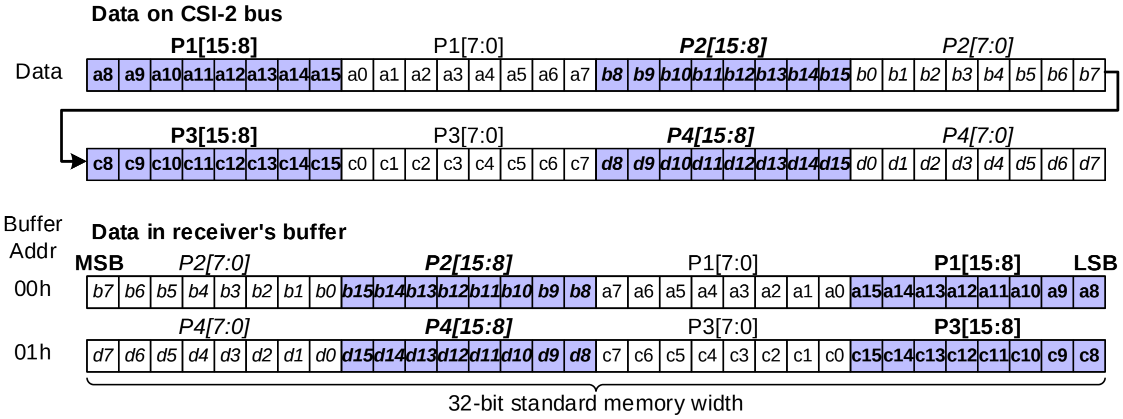

Data on CSI-2 bus
Data
P1[15:8] P1[7:0] P2[15:8] P2[7:0]
a8 a9 a10 a11 a12 a13 a14 a15 a0 a1 a2 a3 a4 a5 a6 a7 b8 b9 b10 b11 b12 b13 b14 b15 b0 b1 b2 b3 b4 b5 b6 b7
P3[15:8] P3[7:0] P4[15:8] P4[7:0]
c8 c9 c10 c11 c12 c13 c14 c15 c0 c1 c2 c3 c4 c5 c6 c7 d8 d9 d10 d11 d12 d13 d14 d15 d0 d1 d2 d3 d4 d5 d6 d7
Buffer Addr
Data in receiver's buffer
MSB P2[7:0] P2[15:8] P1[7:0] P1[15:8] LSB
00h
b7 b6 b5 b4 b3 b2 b1 b0 b15 b14 b13 b12 b11 b10 b9 b8 a7 a6 a5 a4 a3 a2 a1 a0 a15 a14 a13 a12 a11 a10 a9 a8
P4[7:0] P4[15:8] P3[7:0] P3[15:8]
01h
d7 d6 d5 d4 d3 d2 d1 d0 d15 d14 d13 d12 d11 d10 d9 d8 c7 c6 c5 c4 c3 c2 c1 c0 c15 c14 c13 c12 c11 c10 c9 c8
32-bit standard memory width

Figure 171 RAW16 Data Format Reception

12.19 RAW20 Data Reception

The RAW20 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.

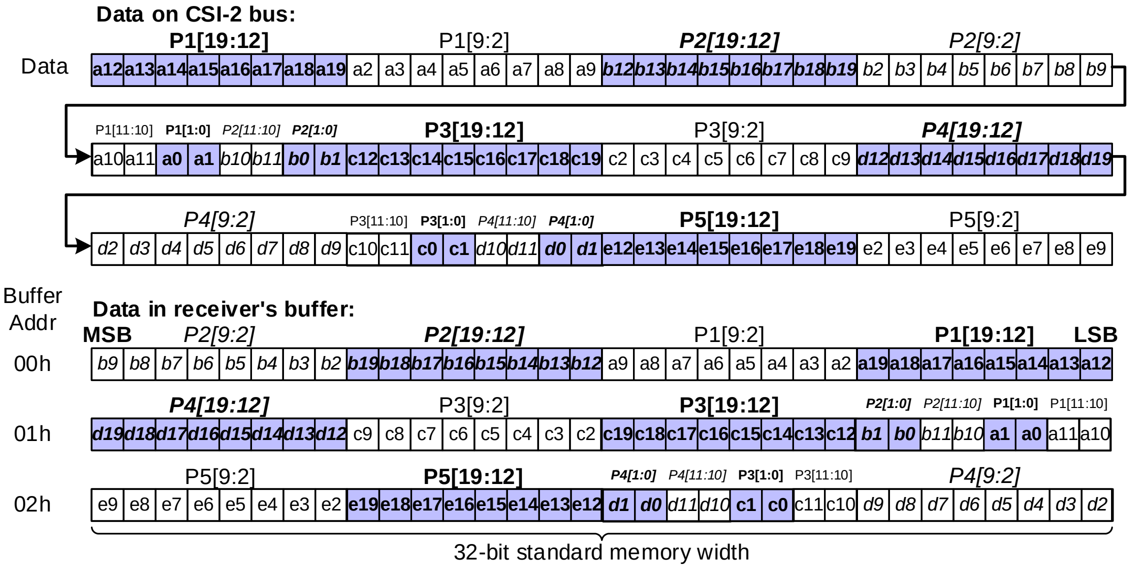

Data on CSI-2 bus:
Data
P1[19:12] P1[9:2] P2[19:12] P2[9:2]
a12 a13 a14 a15 a16 a17 a18 a19 a2 a3 a4 a5 a6 a7 a8 a9 b12 b13 b14 b15 b16 b17 b18 b19 b2 b3 b4 b5 b6 b7 b8 b9
P1[11:10] P1[1:0] P2[11:10] P2[1:0] P3[19:12] P3[9:2] P4[19:12]
a10 a11 a0 a1 b10 b11 b0 b1 c12 c13 c14 c15 c16 c17 c18 c19 c2 c3 c4 c5 c6 c7 c8 c9 d12 d13 d14 d15 d16 d17 d18 d19
P4[9:2] P3[11:10] P3[1:0] P4[11:10] P4[1:0] P5[19:12] P5[9:2]
d2 d3 d4 d5 d6 d7 d8 d9 c10 c11 c0 c1 d10 d11 d0 d1 e12 e13 e14 e15 e16 e17 e18 e19 e2 e3 e4 e5 e6 e7 e8 e9
Buffer Addr
Data in receiver's buffer:
MSB P2[9:2] P2[19:12] P1[9:2] P1[19:12] LSB
00h
b9 b8 b7 b6 b5 b4 b3 b2 b19 b18 b17 b16 b15 b14 b13 b12 a9 a8 a7 a6 a5 a4 a3 a2 a19 a18 a17 a16 a15 a14 a13 a12
P4[19:12] P3[9:2] P3[19:12] P2[1:0] P2[11:10] P1[1:0] P1[11:10]
01h
d19 d18 d17 d16 d15 d14 d13 d12 c9 c8 c7 c6 c5 c4 c3 c2 c19 c18 c17 c16 c15 c14 c13 c12 b1 b0 b11 b10 a1 a0 a11 a10
P5[9:2] P5[19:12] P4[1:0] P4[11:10] P3[1:0] P3[11:10] P4[9:2]
02h
e9 e8 e7 e6 e5 e4 e3 e2 e19 e18 e17 e16 e15 e14 e13 e12 d1 d0 d11 d10 c1 c0 c11 c10 d9 d8 d7 d6 d5 d4 d3 d2
32-bit standard memory width

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***.

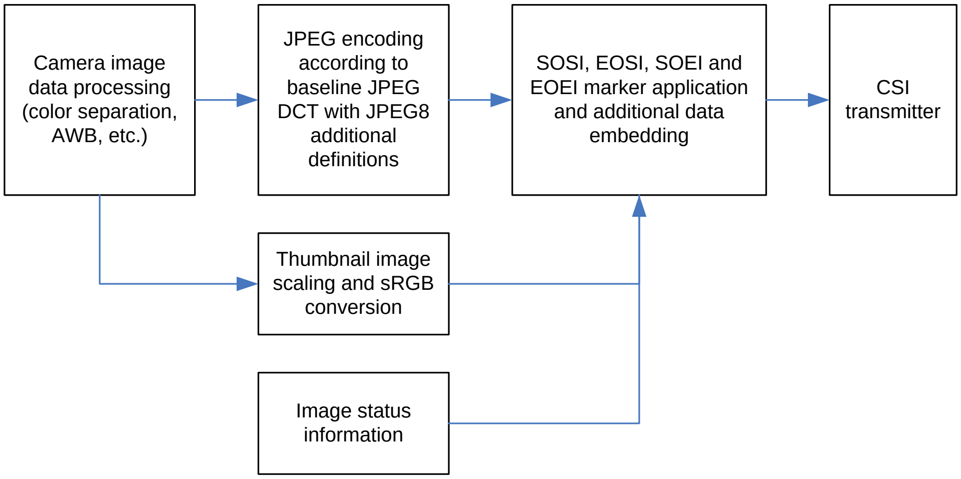

Camera image data processing (color separation, AWB, etc.)
JPEG encoding according to baseline JPEG DCT with JPEG8 additional definitions
SOSI, EOSI, SOEI and EOEI marker application and additional data embedding
CSI transmitter
Thumbnail image scaling and sRGB conversion
Image status information

Figure 173 JPEG8 Data Flow in the Encoder

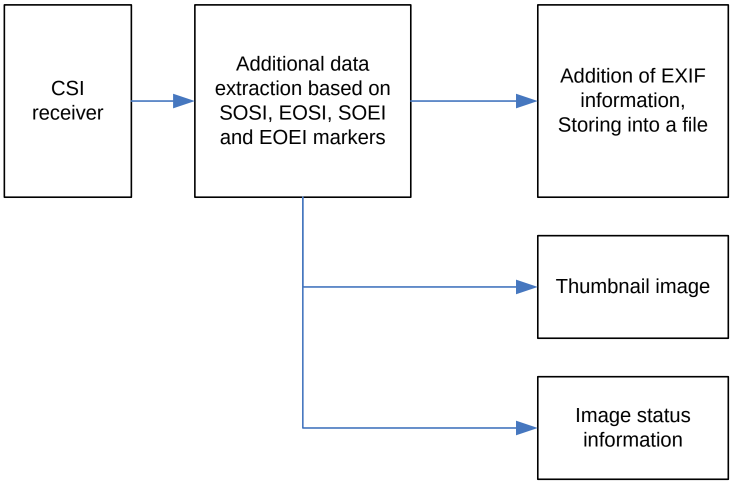

CSI receiver
Additional data extraction based on SOSI, EOSI, SOEI and EOEI markers
Addition of EXIF information, Storing into a file
Thumbnail image
Image status information

Figure 174 JPEG8 Data Flow in the Decoder

A.2 JPEG Data Definition

The JPEG data generated in camera module is baseline JPEG DCT format defined in ISO/IEC 10918-1, with following additional definitions or modifications:

- sRGB color space shall be used. The JPEG is generated from YCbCr format after sRGB to YCbCr conversion.
- The JPEG metadata has to be EXIF compatible, i.e. metadata within application segments has to be placed in beginning of file, in the order illustrated in ***Figure 175***.
- A status line is added in the end of JPEG data as defined in ***Section A.3***.
- If needed, an embedded image is interlaced in order which is free of choice as defined in ***Section A.4***.
- Prior to storing into a file, the CSI-2 JPEG data is processed by the data separation process described in ***Section A.1***.

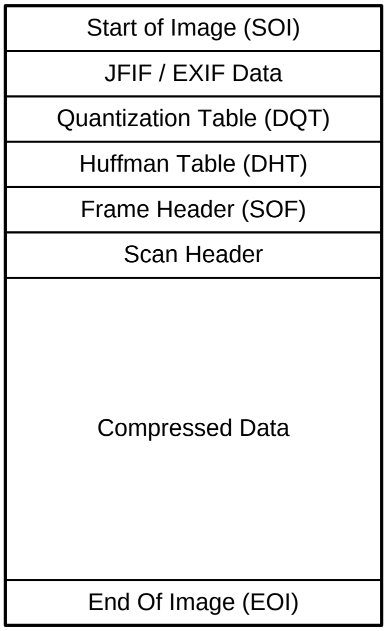


Figure 175 EXIF Compatible Baseline JPEG DCT Format

A.3 Image Status Information

Information of at least the following items has to be stored in the end of the JPEG sequence as illustrated in ***Figure 176***:

- Image exposure time
- Analog & digital gains used
- White balancing gains for each color component
- Camera version number
- Camera register settings
- Image resolution and possible thumbnail resolution

The camera register settings may include a subset of camera's registers. The essential information needed for JPEG8 image is the information needed for converting the image back to linear space. This is necessary e.g. for printing service. An example of register settings is following:

- Sample frequency
- Exposure
- Analog and digital gain
- Gamma
- Color gamut conversion matrix
- Contrast
- Brightness
- Pre-gain

The status information content has to be defined in the product specification of each camera module containing the JPEG8 feature. The format and content is manufacturer specific.

The image status data should be arranged so that each byte is split into two 4-bit nibbles and "1010" padding sequence is added to MSB, as presented in ***Table 46***. This ensures that no JPEG escape sequences (0xFF 0x00) are present in the status data.

The SOSI and EOSI markers are defined in ***Section A.5***.

Table 46 Status Data Padding

Data Word	After Padding
D7D6D5D4 D3D2D1D0	1010D7D6D5D4 1010D3D2D1D0

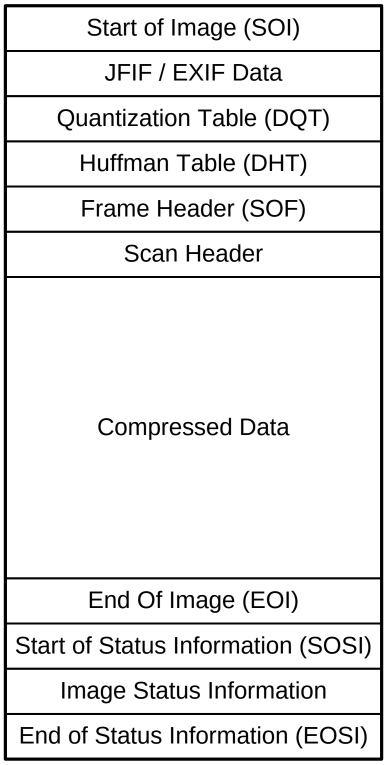


Figure 176 Status Information Field in the End of Baseline JPEG Frame

A.4 Embedded Images

An image may be embedded inside the JPEG data, if needed. The embedded image feature is not compulsory for each camera module containing the JPEG8 feature. An example of embedded data is a 24-bit RGB thumbnail image.

The philosophy of embedded / interleaved thumbnail additions is to minimize the needed frame memory. The EI (Embedded Image) data can be included in any part of the compressed image data segment and in as many pieces as needed. See ***Figure 177***.

Embedded Image data is separated from compressed data by SOEI (Start Of Embedded Image) and EOEI (End Of Embedded Image) non-standard markers, which are defined in ***Section A.5***. The amount of fields separated by SOEI and EOEI is not limited.

The pixel to byte packing for image data within an EI data field should be as specified for the equivalent CSI-2 data format. However there is an additional restriction; the embedded image data must not generate any false JPEG marker sequences (0xFFXX).

The suggested method of preventing false JPEG marker codes from occurring within the embedded image data it to limit the data range for the pixel values. For example

- For RGB888 data the suggested way to solve the false synchronization code issue is to constrain the numerical range of R, G and B values from 1 to 254.
- For RGB565 data the suggested way to solve the false synchronization code issue is to constrain the numerical range of G component from 1-62 and R component from 1-30.

Each EI data field is separated by the SOEI / EOEI markers, and has to contain an equal amount bytes and a complete number of pixels. An EI data field may contain multiple lines or a full frame of image data.

The embedded image data is decoded and removed apart from the JPEG compressed data prior to writing the JPEG into a file. In the process, EI data fields are appended one after each other, in order of occurrence in the received JPEG data.

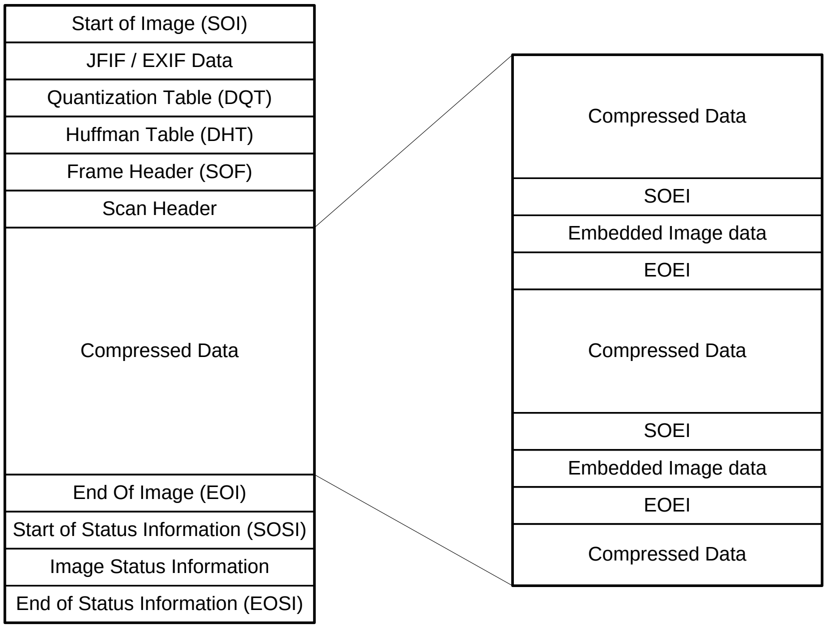


Figure 177 Example of TN Image Embedding Inside the Compressed JPEG Data Block

A.5 JPEG8 Non-standard Markers

JPEG8 uses the reserved JPEG data markers for special purposes, marking the additional segments inside the data file. These segments are not part of the JPEG, JFIF [0], EXIF [0] or any other specifications; instead their use is specified in this document in ***Section A.3*** and ***Section A.4***.

The use of the non-standard markers is always internal to a product containing the JPEG8 camera module, and these markers are always removed from the JPEG data before storing it into a file.

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

The compressed data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

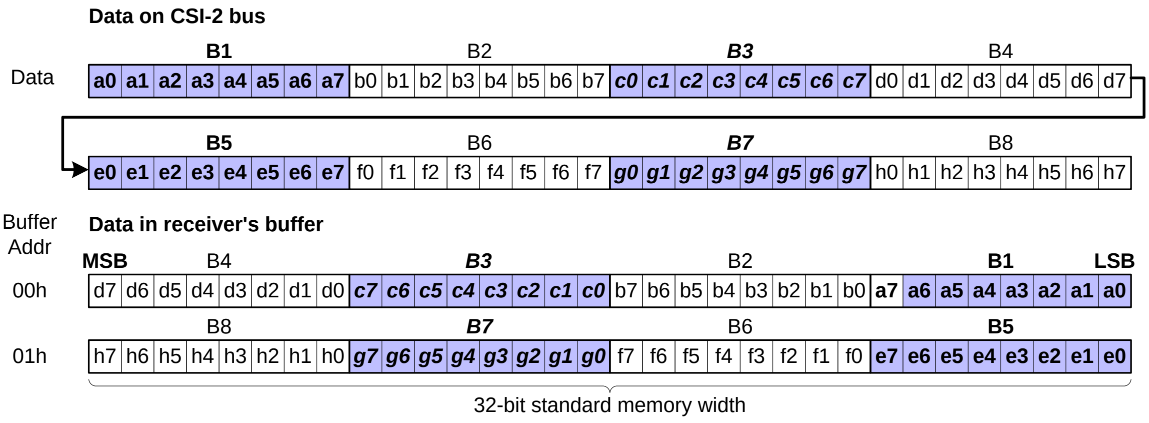

Data on CSI-2 bus
Data
B1 B2 B3 B4
a0 a1 a2 a3 a4 a5 a6 a7 b0 b1 b2 b3 b4 b5 b6 b7 c0 c1 c2 c3 c4 c5 c6 c7 d0 d1 d2 d3 d4 d5 d6 d7
B5 B6 B7 B8
e0 e1 e2 e3 e4 e5 e6 e7 f0 f1 f2 f3 f4 f5 f6 f7 g0 g1 g2 g3 g4 g5 g6 g7 h0 h1 h2 h3 h4 h5 h6 h7
Buffer Addr
Data in receiver's buffer
MSB B4 B3 B2 B1 LSB
00h
d7 d6 d5 d4 d3 d2 d1 d0 c7 c6 c5 c4 c3 c2 c1 c0 b7 b6 b5 b4 b3 b2 b1 b0 a7 a6 a5 a4 a3 a2 a1 a0
B8 B7 B6 B5
01h
h7 h6 h5 h4 h3 h2 h1 h0 g7 g6 g5 g4 g3 g2 g1 g0 f7 f6 f5 f4 f3 f2 f1 f0 e7 e6 e5 e4 e3 e2 e1 e0
32-bit standard memory width

Figure 178 JPEG8 Data Format Reception

Annex B CSI-2 Implementation Example (informative)

B.1 Overview

The CSI-2 implementation example assumes that the interface comprises of D-PHY unidirectional Clock and Data, with forward escape mode and optional deskew functionality. The scope in this implementation example refers only to the unidirectional data link without any references to the CCI interface, as it can be seen in ***Figure 179***. This implementation example varies from the informative PPI example in ***[MIPI01]***.

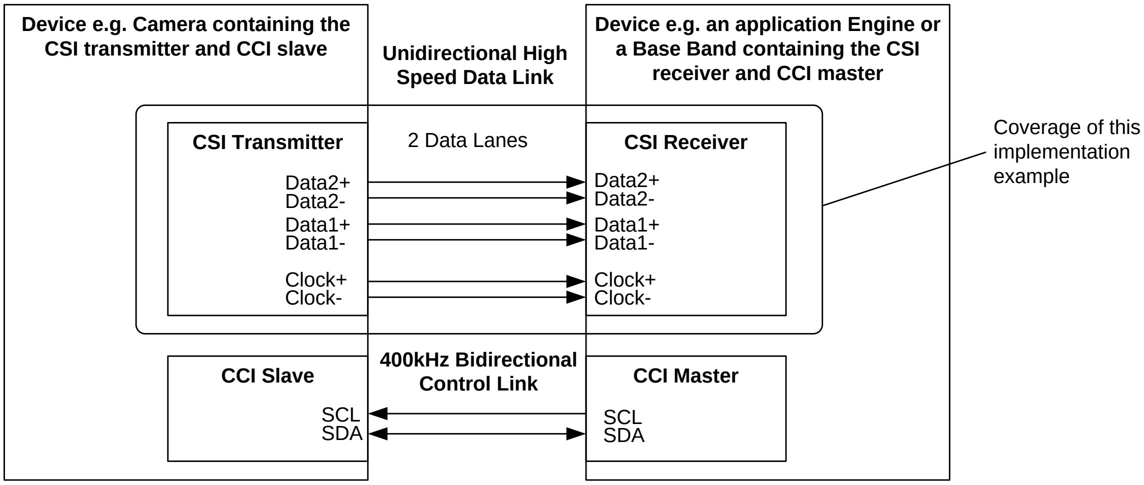

Device e.g. Camera containing the CSI transmitter and CCI slave
Unidirectional High Speed Data Link
Device e.g. an application Engine or a Base Band containing the CSI receiver and CCI master
CSI Transmitter
2 Data Lanes
CSI Receiver
Data2+
Data2-
Data1+
Data1-
Clock+
Clock-
Data2+
Data2-
Data1+
Data1-
Clock+
Clock-
Coverage of this implementation example
CCI Slave
400kHz Bidirectional Control Link
CCI Master
SCL
SDA
SCL
SDA

Figure 179 Implementation Example Block Diagram and Coverage

For this implementation example a layered structure is described with the following parts:

- D-PHY implementation details
- Multi lane merger details
- Protocol layer details

This implementation example refers to a RAW8 data type only; hence no packing/unpacking or byte clock/pixel clock timing will be referenced as for this type of implementation they are not needed.

No error recovery mechanism or error processing details will be presented, as the intent of the document is to present an implementation from the data flow perspective.

B.2 CSI-2 Transmitter Detailed Block Diagram

Using the layered structure described in the overview the CSI-2 transmitter could have the block diagram in ***Figure 180***.

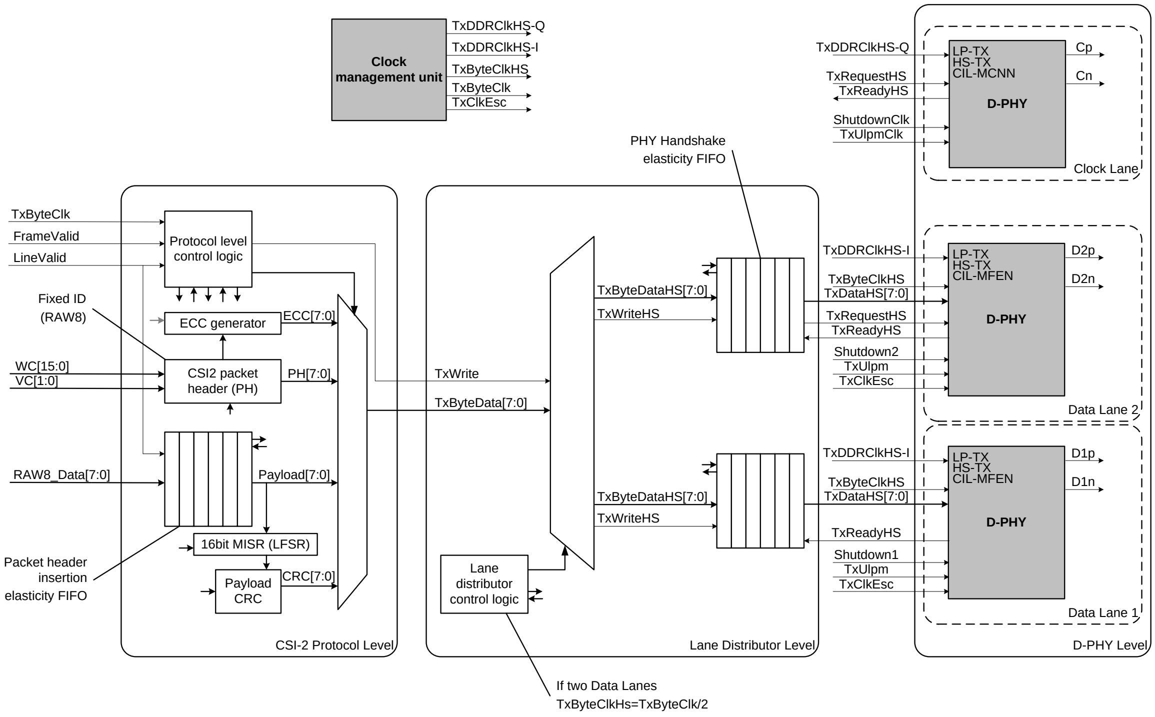


Figure 180 CSI-2 Transmitter Block Diagram

B.3 CSI-2 Receiver Detailed Block Diagram

Using the layered structure described in the overview, the CSI-2 receiver could have the block diagram in ***Figure 181***.

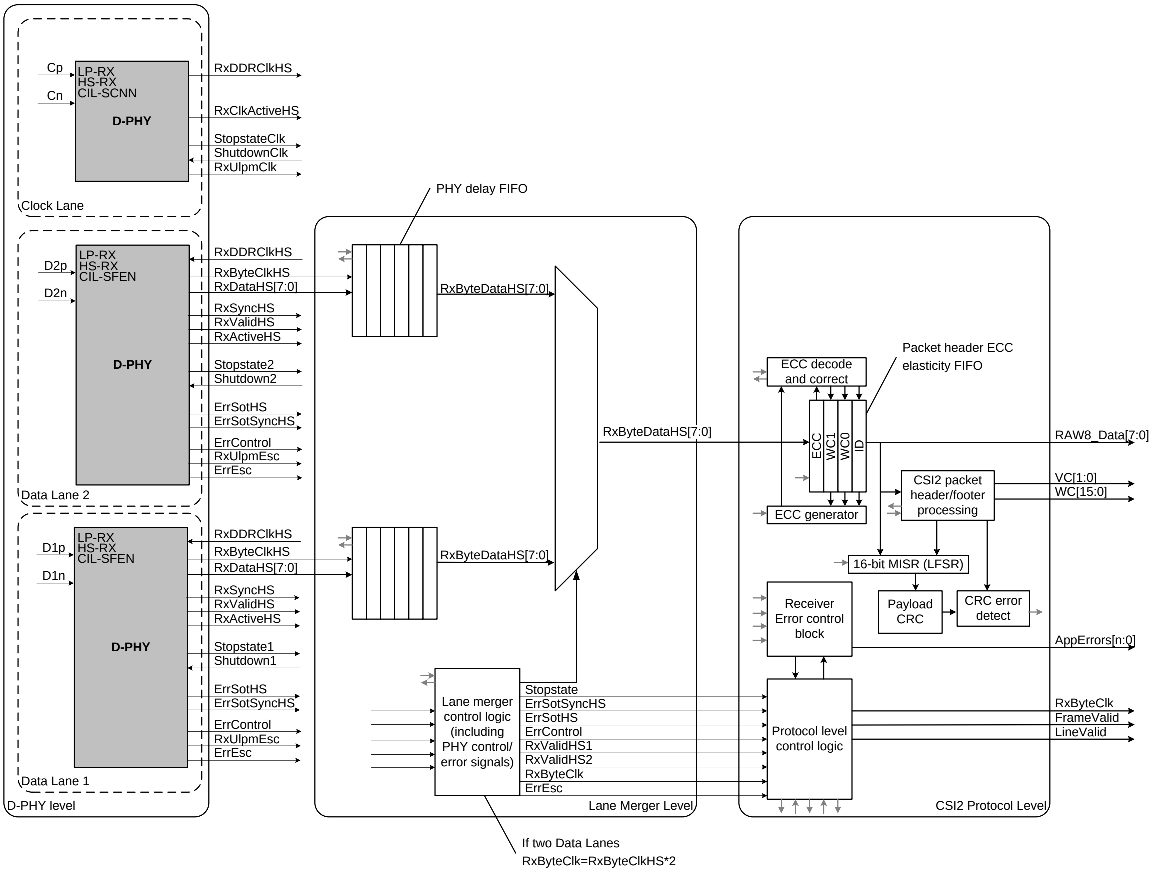


Figure 181 CSI-2 Receiver Block Diagram

B.4 Details on the D-PHY Implementation

The PHY level of implementation has the top level structure as seen in ***Figure 182***.

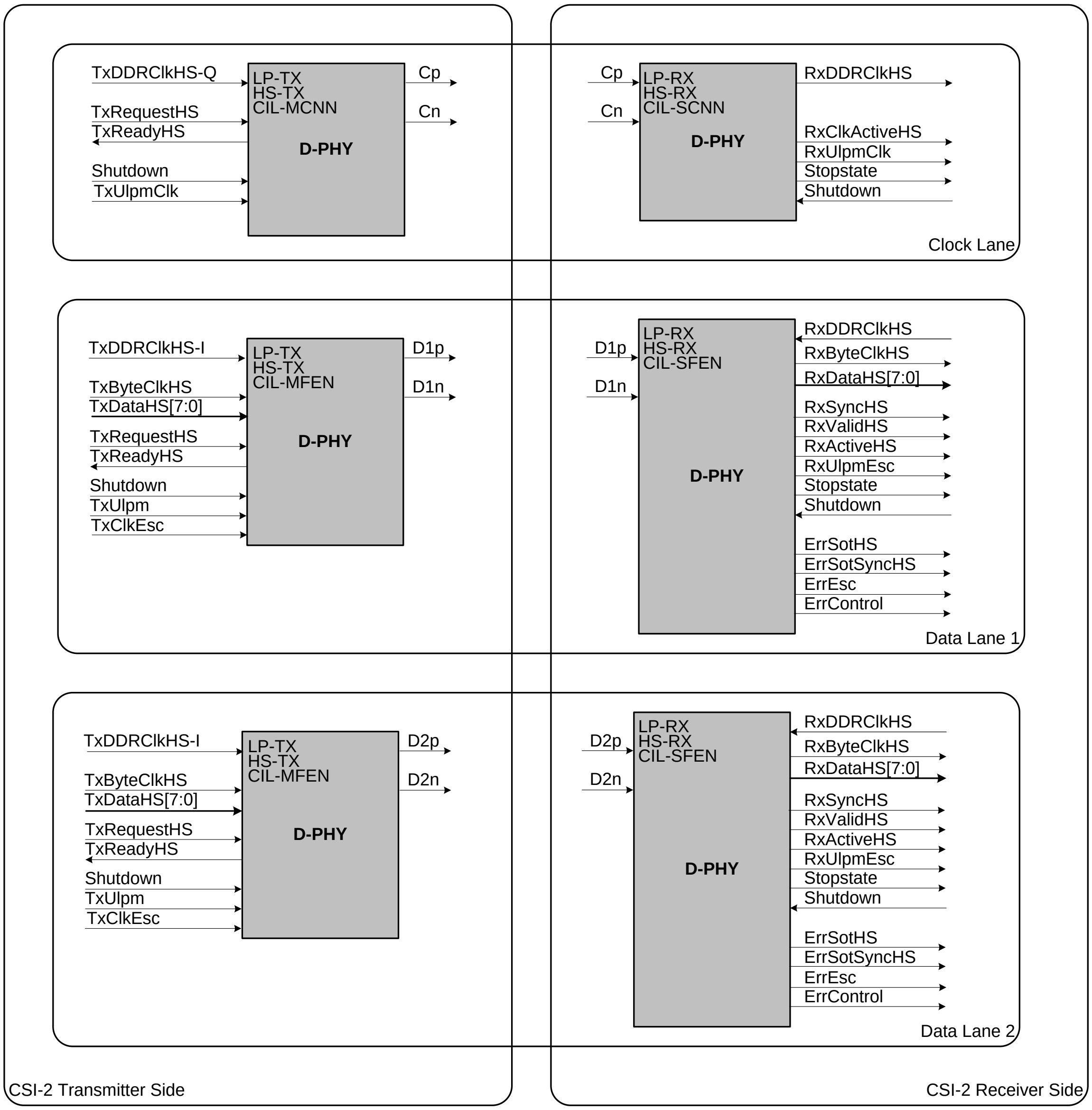


Figure 182 D-PHY Level Block Diagram

The components can be categorized as:

- CSI-2 Transmitter side:
  - Clock lane (Transmitter)
  - Data1 lane (Transmitter)
  - Data2 lane (Transmitter)
- CSI-2 Receiver side:
  - Clock lane (Receiver)
  - Data1 lane (Receiver)

- Data2 lane (Receiver)

B.4.1 CSI-2 Clock Lane Transmitter

The suggested implementation can be seen in ***Figure 183***.

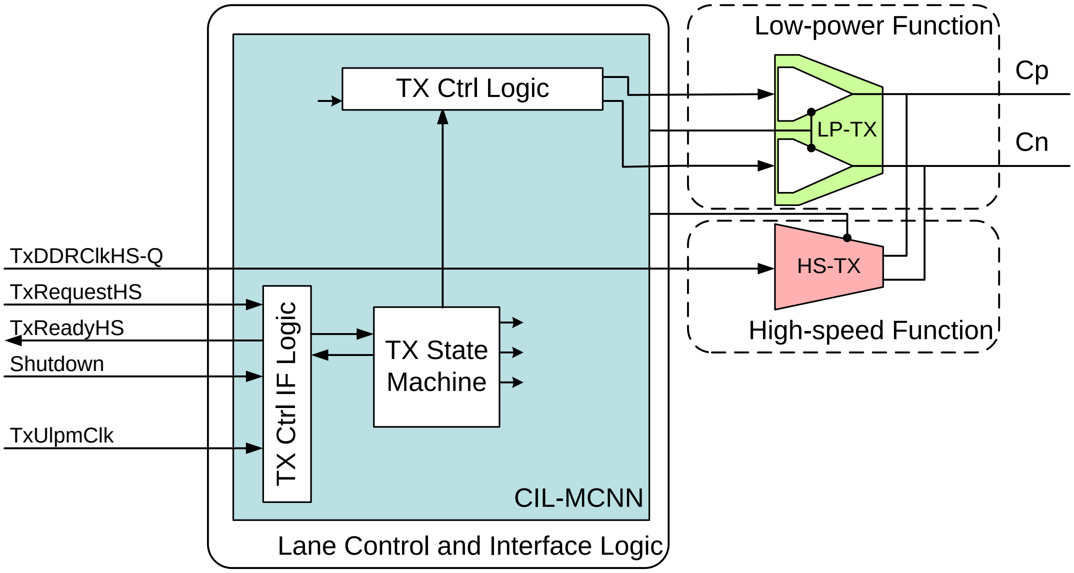


Figure 183 CSI-2 Clock Lane Transmitter

The modular D-PHY components used to build a CSI-2 clock lane transmitter are:

- **LP-TX** for the Low-power function
- **HS-TX** for the High-speed function
- **CIL-MCNN** for the Lane control and interface logic

The PPI interface signals to the CSI-2 clock lane transmitter are:

- **TxDDRClkHS-Q** (Input): High-Speed Transmit DDR Clock (Quadrature).
- **TxRequestHS** (Input): High-Speed Transmit Request. This active high signal causes the lane module to begin transmitting a high-speed clock.
- **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that the clock lane is transmitting HS clock.
- **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.
- **TxUlpmClk** (Input): Transmit Ultra Low-Power mode on Clock Lane This active high signal is asserted to cause a Clock Lane module to enter the Ultra Low-Power mode. The lane module remains in this mode until TxUlpmClk is de-asserted.

B.4.2 CSI-2 Clock Lane Receiver

The suggested implementation can be seen in ***Figure 184***.

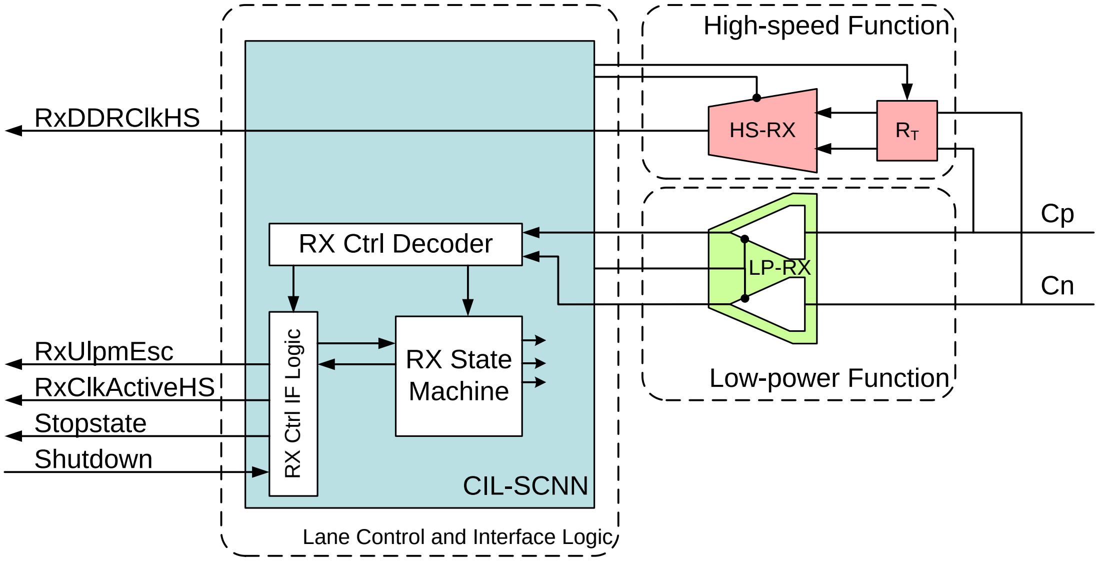


Figure 184 CSI-2 Clock Lane Receiver

The modular D-PHY components used to build a CSI-2 clock lane receiver are:

- **LP-RX** for the Low-power function
- **HS-RX** for the High-speed function
- **CIL-SCNN** for the Lane control and interface logic

The PPI interface signals to the CSI-2 clock lane receiver are:

- **RxDDRClkHS** (Output): High-Speed Receive DDR Clock used to sample the data in all data lanes.
- **RxClkActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the clock lane is receiving valid clock. This signal is asynchronous.
- **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is currently in Stop state. This signal is asynchronous.
- **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 PPI outputs are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on any clock.
- **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is asserted to indicate that the lane module has entered the Ultra Low-Power mode. The lane module remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane interconnect.

B.4.3 CSI-2 Data Lane Transmitter

The suggested implementation can be seen in ***Figure 185***.

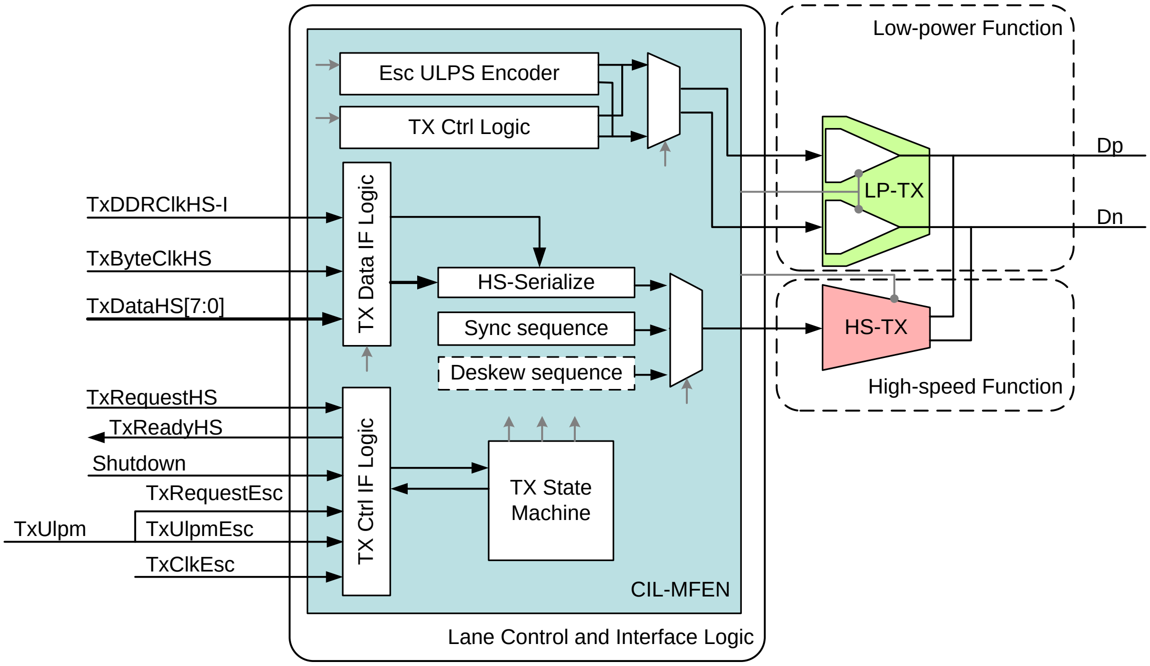


Figure 185 CSI-2 Data Lane Transmitter

The modular D-PHY components used to build a CSI-2 data lane transmitter are:

- **LP-TX** for the Low-power function
- **HS-TX** for the High-speed function
- **CIL-MFEN** for the Lane control and interface logic. For optional deskew calibration support, the data lane transmitter transmits a deskew sequence. The deskew sequence transmission is enabled by a mechanism out of the scope of this specification.

The PPI interface signals to the CSI-2 data lane transmitter are:

- **TxDDRClkHS-I** (Input): High-Speed Transmit DDR Clock (in-phase).
- **TxByteClkHS** (Input): High-Speed Transmit Byte Clock. This is used to synchronize PPI signals in the high-speed transmit clock domain. It is recommended that both transmitting data lane modules share one TxByteClkHS signal. The frequency of TxByteClkHS must be exactly 1/8 the high-speed bit rate.
- **TxDataHS[7:0]** (Input): High-Speed Transmit Data. Eight bit high-speed data to be transmitted. The signal connected to TxDataHS[0] is transmitted first. Data is registered on rising edges of TxByteClkHS.
- **TxRequestHS** (Input): High-Speed Transmit Request. A low-to-high transition on TxRequestHS causes the lane module to initiate a Start-of-Transmission sequence. A high-to-low transition on TxRequest causes the lane module to initiate an End-of-Transmission sequence. This active high signal also indicates that the protocol is driving valid data on TxByteDataHS to be transmitted. The lane module accepts the data when both TxRequestHS and TxReadyHS are active on the same rising TxByteClkHS clock edge. The protocol always provides valid transmit data when TxRequestHS is active. Once asserted, TxRequestHS should remain high until the all the data has 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

The suggested implementation can be seen in ***Figure 186***.

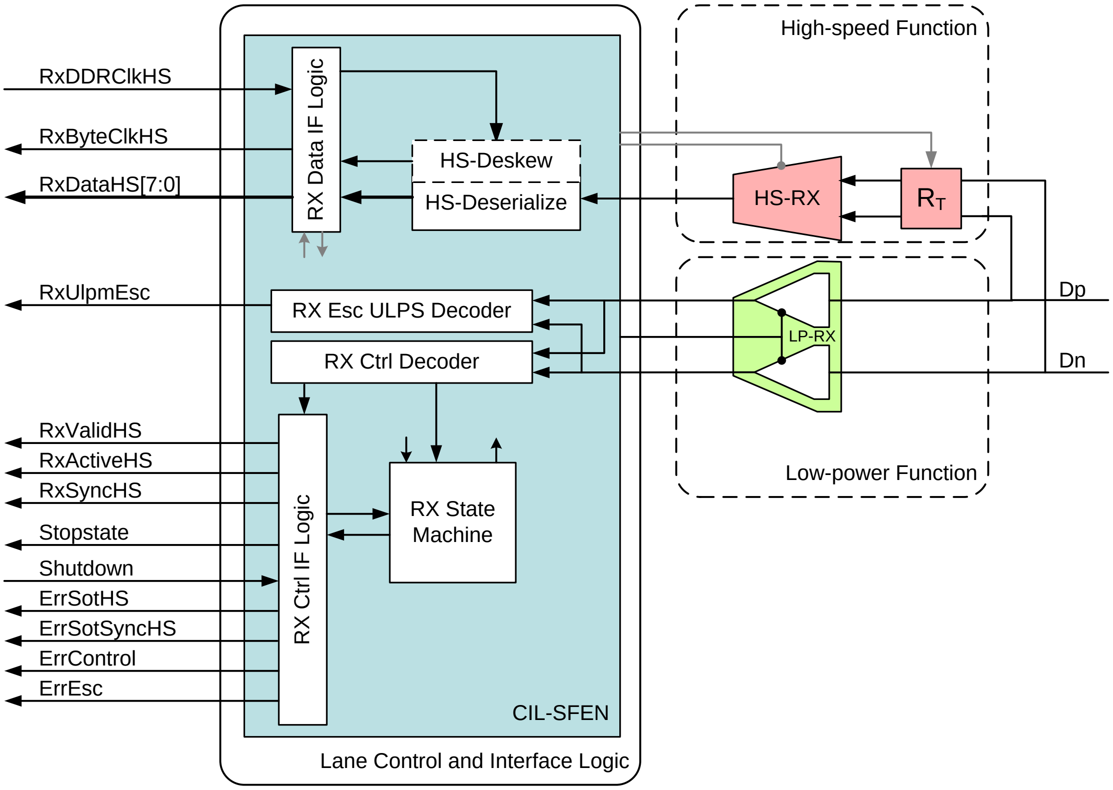


Figure 186 CSI-2 Data Lane Receiver

The modular D-PHY components used to build a CSI-2 data lane receiver are:

- **LP-RX** for the Low-power function
- **HS-RX** for the High-speed function
- **CIL-SFEN** for the Lane control and interface logic. For optional deskew calibration support the data lane receiver detects a transmitted deskew calibration pattern and performs optimum deskew of the Data with respect to the RxDDRClkHS Clock.

The PPI interface signals to the CSI-2 data lane receiver are:

- **RxDDRClkHS** (Input): High-Speed Receive DDR Clock used to sample the date in all data lanes. This signal is supplied by the CSI-2 clock lane receiver.
- **RxByteClkHS** (Output): High-Speed Receive Byte Clock. This signal is used to synchronize signals in the high-speed receive clock domain. The RxByteClkHS is generated by dividing the received RxDDRClkHS.
- **RXDataHS[7:0]** (Output): High-Speed Receive Data. Eight bit high-speed data received by the lane module. The signal connected to RxDataHS[0] was received first. Data is transferred on rising edges of RxByteClkHS.
- **RxValidHS** (Output): High-Speed Receive Data Valid. This active high signal indicates that the lane module is driving valid data to the protocol on the RxDataHS output. There is no "RxReadyHS" signal, and the protocol is expected to capture RxDataHS on every rising edge of RxByteClkHS where RxValidHS is asserted. There is no provision for the protocol to slow down ("throttle") the receive data.

- **RxActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the lane module is actively receiving a high-speed transmission from the lane interconnect.
- **RxSyncHS** (Output): Receiver Synchronization Observed. This active high signal indicates that the lane module has seen an appropriate synchronization event. In a typical high-speed transmission, RxSyncHS is high for one cycle of RxByteClkHS at the beginning of a high-speed transmission when RxActiveHS is first asserted. This signal missing is signaled using ErrSotSyncHS.
- **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is asserted to indicate that the lane module has entered the Ultra Low-Power mode. The lane module remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane interconnect.
- **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is currently in Stop state. This signal is asynchronous.
- **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 PPI outputs are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on any clock.
- **ErrSotHS** (Output): Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is corrupted, but in such a way that proper synchronization can still be achieved, this error signal is asserted for one cycle of RxByteClkHS. This is considered to be a “soft error” in the leader sequence and confidence in the payload data is reduced.
- **ErrSotSyncHS** (Output): Start-of-Transmission Synchronization Error. If the high-speed SoT leader sequence is corrupted in a way that proper synchronization cannot be expected, this error is asserted for one cycle of RxByteClkHS.
- **ErrControl** (Output): Control Error. This signal is asserted when an incorrect line state sequence is detected.
- **ErrEsc** (Output): Escape Entry Error. If an unrecognized escape entry command is received, this signal is asserted and remains high until the next change in line state. The only escape entry command supported by the receiver is the ULPS.

Annex C CSI-2 Recommended Receiver Error Behavior (informative)

C.1 Overview

This section proposes one approach to handling error conditions at the receiving side of a CSI-2 Link. Although the section is informative and therefore does not affect compliance for CSI-2, the approach is offered by the MIPI Camera Working Group as a recommended approach. The CSI-2 receiver assumes the case of a CSI-2 Link comprised of unidirectional Lanes for D-PHY Clock and Data Lanes with Escape Mode functionality on the Data Lanes and a continuously running clock. This Annex does not discuss other cases, including those that differ widely in implementation, where the implementer should consider other potential error situations.

Because of the layered structure of a compliant CSI-2 receiver implementation, the error behavior is described in a similar way with several "levels" where errors could occur, each requiring some 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

The proposed methodology for handling errors is signal based, since it offers an easy path to a viable CSI-2 implementation that handles all three error levels. Even so, error handling at the Protocol Decoding Level should implement sequential behavior using a state machine for proper operation.

C.2 D-PHY Level Error

The recommended behavior for handling this error level covers only those errors generated by the Data Lane(s), since an implementation can assume that the Clock Lane is running reliably as provided by the expected BER of the Link, as discussed in ***[MIPI01]***. Note that this error handling behavior assumes unidirectional Data Lanes without escape mode functionality. Considering this, and using the signal names and descriptions from the ***[MIPI01]***, PPI Annex, signal errors at the PHY-Protocol Interface (PPI) level consist of the following:

- **ErrSotHS:** Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is corrupted, but in such a way that proper synchronization can still be achieved, this error signal is asserted for one cycle of RxByteClkHS. This is considered to be a “soft error” in the leader sequence and confidence in the payload data is reduced.
- **ErrSotSyncHS:** Start-of-Transmission Synchronization Error. If the high-speed SoT leader sequence is corrupted in a way that proper synchronization cannot be expected, this error signal is asserted for one cycle of RxByteClkHS.
- **ErrControl:** Control Error. This signal is asserted when an incorrect line state sequence is detected. For example, if a Turn-around request or Escape Mode request is immediately followed by a Stop state instead of the required Bridge state, this signal is asserted and remains high until the next change in line state.

The recommended receiver error behavior for this level is:

- **ErrSotHS** should be passed to the Application Layer. Even though the error was detected and corrected and the Sync mechanism was unaffected, confidence in the data integrity is reduced and the application should be informed. This signal should be referenced to the corresponding data packet.
- **ErrSotSyncHS** should be passed to the Protocol Decoding Level, since this is an unrecoverable error. An unrecoverable type of error should also be signaled to the Application Layer, since the whole transmission until the first D-PHY Stop state should be ignored if this type of error occurs.
- **ErrControl** should be passed to the Application Layer, since this type of error doesn’t normally occur if the interface is configured to be unidirectional. Even so, the application should be aware of the error and configure the interface accordingly through other, implementation specific-means that are out of scope for this specification.

Also, it is recommended that the PPI StopState signal for each implemented Lane should be propagated to the Application Layer during configuration or initialization to indicate the Lane is ready.

C.3 Packet Level Error

The recommended behavior for this error level covers only errors recognized by decoding the Packet Header's ECC field and computing the CRC of the data payload.

Decoding and applying the ECC field of the Packet Header should signal the following errors:

- **ErrEccDouble:** Asserted when an ECC syndrome was computed and two bit-errors are detected in the received Packet Header.
- **ErrEccCorrected:** Asserted when an ECC syndrome was computed and a single bit-error in the Packet Header was detected and corrected.
- **ErrEccNoError:** Asserted when an ECC syndrome was computed and the result is zero indicating a Packet Header that is considered to be without errors or has more than two bit-errors. CSI-2's ECC mechanism cannot detect this type of error.

Also, computing the CRC code over the whole payload of the received packet could generate the following errors:

- **ErrCrc:** Asserted when the computed CRC code is different than the received CRC code.
- **ErrID:** Asserted when a Packet Header is decoded with an unrecognized or unimplemented data ID.

The recommended receiver error behavior for this level is:

- **ErrEccDouble** should be passed to the Application Layer since assertion of this signal proves that the Packet Header information is corrupt, and therefore the WC is not usable, and thus the packet end cannot be estimated. Commonly, this type of error will be accompanied with an ErrCrc. This type of error should also be passed to the Protocol Decoding Level, since the whole transmission until D-PHY Stop state should be ignored.
- **ErrEccCorrected** should be passed to the Application Layer since the application should be informed that an error had occurred but was corrected, so the received Packet Header was unaffected, although the confidence in the data integrity is reduced.
- **ErrEccNoError** can be passed to the Protocol Decoding Level to signal the validity of the current Packet Header.
- **ErrCrc** should be passed to the Protocol Decoding Level to indicate that the packet's payload data might be corrupt.
- **ErrID** should be passed to the Application Layer to indicate that the data packet is unidentified and cannot be unpacked by the receiver. This signal should be asserted after the ID has been 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

Since a camera in a mobile terminal spends most of its time in an inactive state, implementers need a way to put the CSI-2 Link into a low power mode that approaches, or may be as low as, the leakage level. This section proposes one approach for putting a CSI-2 Link in a "Sleep Mode" (SLM). Although the section is informative and therefore does not affect compliance for CSI-2, the approach is offered by the MIPI Camera Working Group as a recommended approach.

This approach relies on an aspect of a D-PHY or C-PHY transmitter's behavior that permits regulators to be disabled safely when LP-00 (Space state) is on the Link. Accordingly, this will be the output state for a CSI-2 camera transmitter in SLM.

SLM can be thought of as a three-phase process:

3. SLM Command Phase. The 'ENTER SLM' command is issued to the TX side only, or to both sides of the Link.
4. SLM Entry Phase. The CSI-2 Link has entered, or is entering, the SLM in a controlled or synchronized manner. This phase is also part of the power-down process.
5. SLM Exit Phase. The CSI-2 Link has exited the SLM and the interface/device is operational. This phase is also part of the power-up process.

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

For the first phase, initiation of SLM occurs by a mechanism outside the scope of CSI-2. Of the many mechanisms available, two examples would be:

1. An External SLEEP signal input to the CSI-2 transmitter and optionally also to the CSI-2 Receiver. When at logic 0, the CSI-2 Transmitter and the CSI Receiver (if connected) will enter Sleep mode. When at logic 1, normal operation will take place.
2. A CCI control command, provided on the I2C control Link, is used to trigger ULPS.

D.3 SLM Entry Phase

For the second phase, consider one option:

Only the TX side enters SLM and propagates the ULPS to the RX side by sending a D-PHY or C-PHY ‘ULPS’ command on each Lane. In ***Figure 187***, only the Data Lane ‘ULPS’ command is used as an example. The D-PHY Dp, Dn, and C-PHY Data_A, Data_C are logical signal names and do not imply specific multiplexing on dual mode (combined D-PHY and C-PHY) implementations.

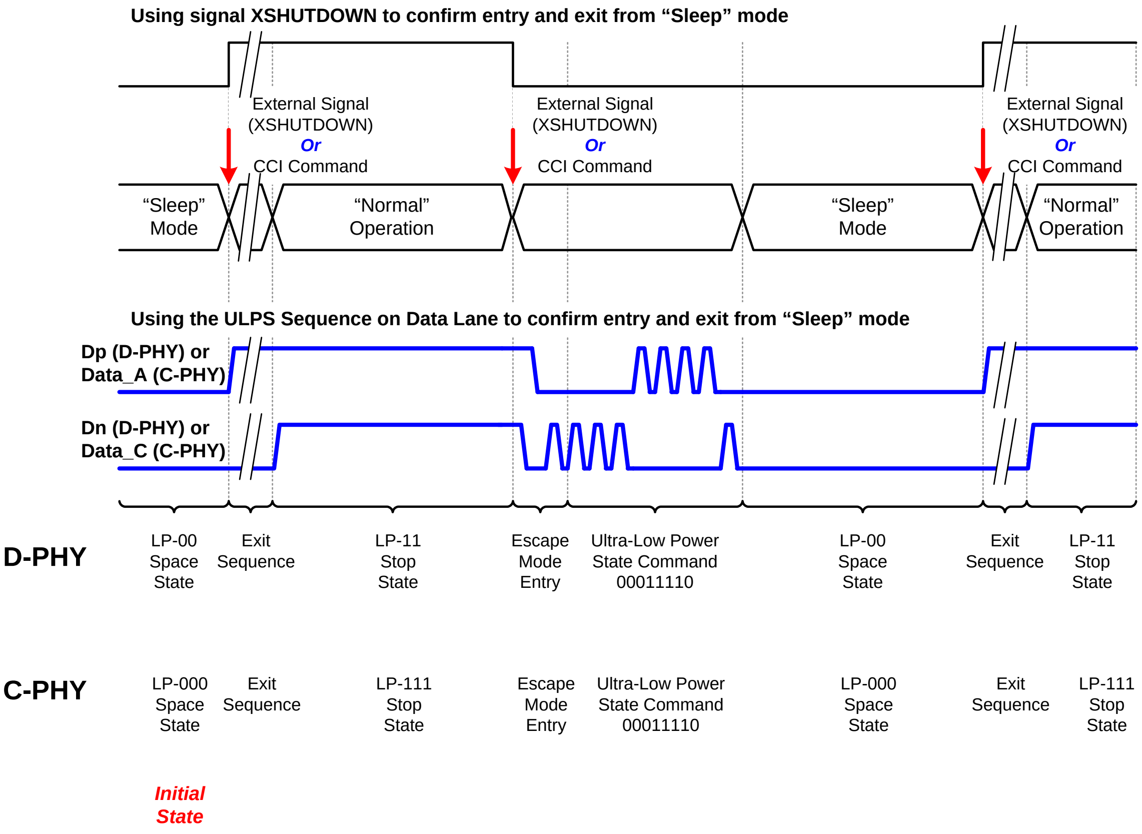

Using signal XSHUTDOWN to confirm entry and exit from “Sleep” mode
External Signal (XSHUTDOWN) Or CCI Command
External Signal (XSHUTDOWN) Or CCI Command
External Signal (XSHUTDOWN) Or CCI Command
“Sleep” Mode
“Normal” Operation
“Sleep” Mode
“Normal” Operation
Using the ULPS Sequence on Data Lane to confirm entry and exit from “Sleep” mode
Dp (D-PHY) or Data_A (C-PHY)
Dn (D-PHY) or Data_C (C-PHY)
D-PHY
LP-00 Space State
Exit Sequence
LP-11 Stop State
Escape Mode Entry
Ultra-Low Power State Command 00011110
LP-00 Space State
Exit Sequence
LP-11 Stop State
C-PHY
LP-000 Space State
Exit Sequence
LP-111 Stop State
Escape Mode Entry
Ultra-Low Power State Command 00011110
LP-000 Space State
Exit Sequence
LP-111 Stop State
Initial State

Figure 187 SLM Synchronization

D.4 SLM Exit Phase

For the third phase, three options are presented and assume the camera peripheral is in ULPS or Sleep mode at power-up:

1. Use a SLEEP signal to power-up both sides of the interface.
2. Detect any CCI activity on the I2C control Link, which was in the 00 state ({SCL, SDA}), after receiving the I2C instruction to enter ULPS command as per ***Section D.2***, option 2. Any change on those lines should wake up the camera peripheral. The drawback of this method is that I2C lines are used exclusively for control of the camera.
3. Detect a wake-up sequence on the I2C lines. This sequence, which may vary by implementation, shall not disturb the I2C interface so that it can be used by other devices. One example sequence is: StopI2C-StartI2C-StopI2C. See ***Section 6*** for details on CCI.

A handshake using the ‘ULPS’ mechanism as described in ***[MIPI01]*** or ***[MIPI02]***, as appropriate, should be used for powering up the interface.

Annex E Data Compression for RAW Data Types (normative)

A CSI-2 implementation using RAW data types may support compression on the interface to reduce the data bandwidth requirements between the host processor and a camera module. Data compression is not mandated by this Specification. However, if data compression is used, it shall be implemented as described in this annex.

Data compression schemes use an X–Y–Z naming convention where X is the number of bits per pixel in the original image, Y is the encoded (compressed) bits per pixel and Z is the decoded (uncompressed) bits per pixel.

The following data compression schemes are defined:

- 12-10-12
- 12–8–12
- 12–7–12
- 12–6–12
- 10–8–10
- 10–7–10
- 10–6–10

To identify the type of data on the CSI-2 interface, packets with compressed data shall have a User Defined Data Type value as indicated in ***Table 45***. Note that User Defined data type codes are not reserved for compressed data types. Therefore, a CSI-2 device shall be able to communicate over the CCI the data compression scheme represented by a particular User Defined data type code for each scheme supported by the device. Note that the method to communicate the data compression scheme to Data Type code mapping is beyond the scope of this document.

The number of bits in a packet shall be a multiple of eight. Therefore, implementations with data compression schemes that result in each pixel having other than eight encoded bits per pixel shall transfer the encoded data in a packed pixel format. For example, the 12–7–12 data compression scheme uses a packed pixel format as described in ***Section 11.4.2*** except the Data Type value in the Packet Header is a User Defined data type code.

The data compression schemes in this annex are lossy and designed to encode each line independent of the other lines in the image.

The following definitions are used in the description of the data compression schemes:

- **Xorig** is the original pixel value
- **Xpred** is the predicted pixel value
- **Xdiff** is the difference value (**Xorig** - **Xpred**)
- **Xenco** is the encoded value
- **Xdeco** is the decoded pixel value

The data compression system consists of encoder, decoder and predictor blocks as shown in ***Figure 188***.

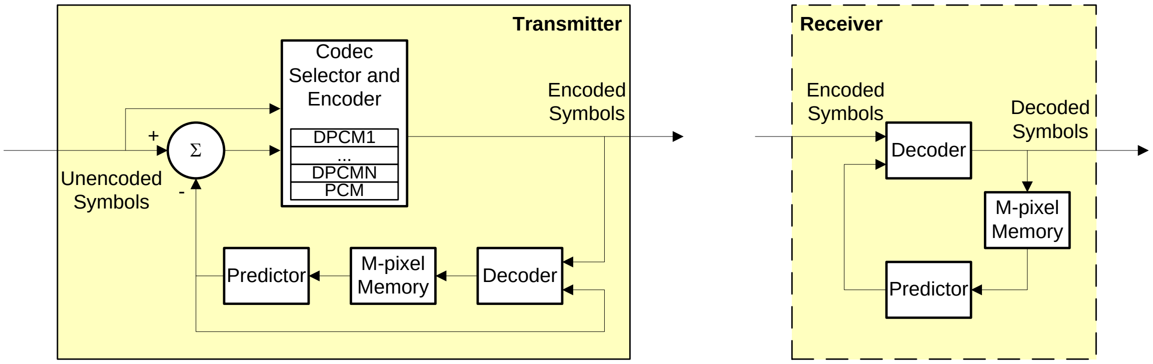


Figure 188 Data Compression System Block Diagram

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***.

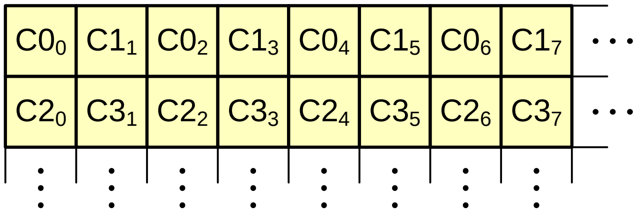

$C0_0$ $C1_1$ $C0_2$ $C1_3$ $C0_4$ $C1_5$ $C0_6$ $C1_7$ ...
$C2_0$ $C3_1$ $C2_2$ $C3_3$ $C2_4$ $C3_5$ $C2_6$ $C3_7$ ...

Figure 189 Pixel Order of the Original Image

Figure 190 shows an example of the pixel order with RGB data.

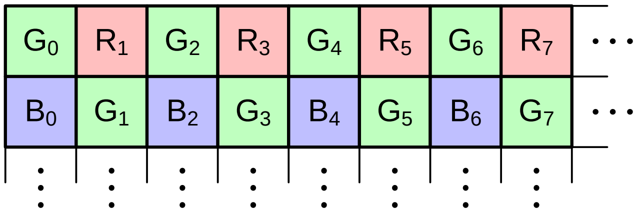

G_0 R_1 G_2 R_3 G_4 R_5 G_6 R_7 ...
B_0 G_1 B_2 G_3 B_4 G_5 B_6 G_7 ...

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

There are seven different encoders available, one for each data compression scheme.

For all encoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula for predicted pixels.

E.2.1 Coder for 10–8–10 Data Compression

The 10–8–10 coder offers a 20% bit rate reduction with very high image quality.

Pixels without prediction are encoded using the following formula:

```
Xenco( n ) = Xorig( n ) / 4
```

To avoid a full-zero encoded value, the following check is performed:

```
if (Xenco( n ) == 0) then
   Xenco( n ) = 1
endif
```

Pixels with prediction are encoded using the following formula:

```
if (abs(Xdiff( n )) < 32) then
   use DPCM1
else if (abs(Xdiff( n )) < 64) then
   use DPCM2
else if (abs(Xdiff( n )) < 128) then
   use DPCM3
else
   use PCM
endif
```

E.2.1.1 DPCM1 for 10–8–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “00 s xxxxx”
```

where,

```
“00” is the code word
“s” is the sign bit
“xxxxx” is the five bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) <= 0) then
   sign = 1
else
   sign = 0
endif
value = abs(Xdiff( n ))
```

Note: Zero code has been avoided (0 is sent as -0).

E.2.1.2 DPCM2 for 10–8–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xxxx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 32) / 2
```

E.2.1.3 DPCM3 for 10–8–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xxxx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 64) / 4
```

E.2.1.4 PCM for 10–8–10 Coder

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 coder equation is described as follows:

```
value = Xorig( n ) / 8
```

E.2.2 Coder for 10–7–10 Data Compression

The 10–7–10 coder offers 30% bit rate reduction with high image quality.

Pixels without prediction are encoded using the following formula:

```
Xenco( n ) = Xorig( n ) / 8
```

To avoid a full-zero encoded value, the following check is performed:

```
if (Xenco( n ) == 0) then
    Xenco( n ) = 1
```

Pixels with prediction are encoded using the following formula:

```
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

Xenco(**n**) has the following format:

```
Xenco( n ) = "000 s xxx"
```

where,

```
"000" is the code word
"s" is the sign bit
"xxx" is the three bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) <= 0) then
    sign = 1
else
    sign = 0
endif
value = abs(Xdiff( n ))
```

Note: Zero code has been avoided (0 is sent as -0).

E.2.2.2 DPCM2 for 10–7–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0010 s xx”
```

where,

```
“0010” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 8) / 2
```

E.2.2.3 DPCM3 for 10–7–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0011 s xx”
```

where,

```
“0011” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 16) / 4
```

E.2.2.4 DPCM4 for 10–7–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “01 s xxxx”
```

where,

```
“01” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 32) / 8
```

E.2.2.5 PCM for 10–7–10 Coder

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 coder equation is described as follows:

```
value = Xorig( n ) / 16
```

E.2.3 Coder for 10–6–10 Data Compression

The 10–6–10 coder offers 40% bit rate reduction with acceptable image quality.

Pixels without prediction are encoded using the following formula:

```
Xenco( n ) = Xorig( n ) / 16
```

To avoid a full-zero encoded value, the following check is performed:

```
if (Xenco( n ) == 0) then
    Xenco( n ) = 1
endif
```

Pixels with prediction are encoded using the following formula:

```
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

Xenco(**n**) has the following format:

```
Xenco( n ) = “00000 s”
```

where,

```
“00000” is the code word
“s” is the sign bit
the value field is not used
```

The coder equation is described as follows:

```
sign = 1
Note: Zero code has been avoided (0 is sent as -0).
```

E.2.3.2 DPCM2 for 10–6–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “00001 s”
```

where,

```
“00001” is the code word
“s” is the sign bit
the value field is not used
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
    sign = 1
else
    sign = 0
endif
```

E.2.3.3 DPCM3 for 10–6–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0001 s x”
```

where,

```
“0001” is the code word
“s” is the sign bit
“x” is the one bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
value = (abs(Xdiff( n )) - 3) / 4
endif
```

E.2.3.4 DPCM4 for 10–6–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “001 s xx”
```

where,

```
“001” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 11) / 8
```

E.2.3.5 DPCM5 for 10–6–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “01 s xxx”
```

where,

```
“01” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 43) / 16
```

E.2.3.6 PCM for 10–6–10 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “1 xxxxx”
```

where,

```
“1” is the code word
the sign bit is not used
“xxxxx” is the five bit value field
```

The coder equation is described as follows:

```
value = Xorig( n ) / 32
```

E.2.4 Coder for 12-10-12 Data Compression

The 12–10–12 coder offers a 16.7% bit rate reduction with very high image quality.

Pixels without prediction are encoded using the following formula:

```
Xenco( n ) = Xorig( n ) / 4
```

To avoid a full-zero encoded value, the following check is performed:

```
if (Xenco( n ) == 0) then
   Xenco( n ) = 1
endif
```

Pixels with prediction are encoded using the following formula:

```
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

Xenco(**n**) has the following format:

```
Xenco( n ) = “00 s xxxxxxx”
```

where,

```
“00” is the code word
“s” is the sign bit
“xxxxxxx” is the seven bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) <= 0) then
   sign = 1
else
   sign = 0
endif
value = abs(Xdiff( n ))
```

Note:

Zero code has been avoided (0 is sent as -0).

E.2.4.2 DPCM2 for 12–10–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xxxxxx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xxxxxx” is the six bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 128) / 2
```

E.2.4.3 DPCM3 for 12–10–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xxxxxx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xxxxxx” is the six bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 256) / 4
```

E.2.4.4 PCM for 12–10–12 Coder

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 coder equation is described as follows:

```
value = Xorig( n ) / 8
```

E.2.5 Coder for 12–8–12 Data Compression

The 12–8–12 coder offers 33% bit rate reduction with very high image quality.

Pixels without prediction are encoded using the following formula:

```
Xenco( n ) = Xorig( n ) / 16
```

To avoid a full-zero encoded value, the following check is performed:

```
if (Xenco( n ) == 0) then
   Xenco( n ) = 1
endif
```

Pixels with prediction are encoded using the following formula:

```
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

Xenco(**n**) has the following format:

```
Xenco( n ) = “0000 s xxx”
```

where,

```
“0000” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) <= 0) then
   sign = 1
else
   sign = 0
endif
value = abs(Xdiff( n ))
```

Note: Zero code has been avoided (0 is sent as -0).

E.2.5.2 DPCM2 for 12–8–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xxxx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 8) / 2
```

E.2.5.3 DPCM3 for 12–8–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xxxx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 40) / 4
```

E.2.5.4 DPCM4 for 12–8–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “001 s xxxx”
```

where,

```
“001” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 104) / 8
```

E.2.5.5 DPCM5 for 12–8–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0001 s xxx”
```

where,

```
“0001” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 232) / 16
```

E.2.5.6 PCM for 12–8–12 Coder

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 coder equation is described as follows:

```
value = Xorig( n ) / 32
```

E.2.6 Coder for 12–7–12 Data Compression

The 12–7–12 coder offers 42% bit rate reduction with high image quality.

Pixels without prediction are encoded using the following formula:

```
Xenco( n ) = Xorig( n ) / 32
```

To avoid a full-zero encoded value, the following check is performed:

```
if (Xenco( n ) == 0) then
   Xenco( n ) = 1
endif
```

Pixels with prediction are encoded using the following formula:

```
if (abs(Xdiff( n )) < 4) then
   use DPCM1
else if (abs(Xdiff( n )) < 12) then
   use DPCM2
else if (abs(Xdiff( n )) < 28) then
   use DPCM3
else if (abs(Xdiff( n )) < 92) then
   use DPCM4
else if (abs(Xdiff( n )) < 220) then
   use DPCM5
else if (abs(Xdiff( n )) < 348) then
   use DPCM6
else
   use PCM
endif
```

E.2.6.1 DPCM1 for 12–7–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0000 s xx”
```

where,

```
“0000” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) <= 0) then
   sign = 1
else
   sign = 0
endif
value = abs(Xdiff( n ))
```

Note: Zero code has been avoided (0 is sent as -0).

E.2.6.2 DPCM2 for 12–7–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0001 s xx”
```

where,

```
“0001” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 4) / 2
```

E.2.6.3 DPCM3 for 12–7–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0010 s xx”
```

where,

```
“0010” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 12) / 4
```

E.2.6.4 DPCM4 for 12–7–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xxx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 28) / 8
```

E.2.6.5 DPCM5 for 12–7–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xxx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 92) / 16
```

E.2.6.6 DPCM6 for 12–7–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0011 s xx”
```

where,

```
“0011” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 220) / 32
```

E.2.6.7 PCM for 12–7–12 Coder

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 coder equation is described as follows:

```
value = Xorig( n ) / 64
```

E.2.7 Coder for 12–6–12 Data Compression

The 12–6–12 coder offers 50% bit rate reduction with acceptable image quality.

Pixels without prediction are encoded using the following formula:

```
Xenco( n ) = Xorig( n ) / 64
```

To avoid a full-zero encoded value, the following check is performed:

```
if (Xenco( n ) == 0) then
   Xenco( n ) = 1
endif
```

Pixels with prediction are encoded using the following formula:

```
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
```

*Note: **DPCM2** is not used.*

E.2.7.1 DPCM1 for 12–6–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0000 s x”
```

where,

```
“0000” is the code word
“s” is the sign bit
“x” is the one bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) <= 0) then
   sign = 1
else
   sign = 0
endif
value = abs(Xdiff( n ))
```

Note: Zero code has been avoided (0 is sent as -0).

E.2.7.2 DPCM3 for 12–6–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0001 s x”
```

where,

```
“0001” is the code word
“s” is the sign bit
“x” is the one bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 2) / 4
```

E.2.7.3 DPCM4 for 12–6–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 10) / 8
```

E.2.7.4 DPCM5 for 12–6–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0010 s x”
```

where,

```
“0010” is the code word
“s” is the sign bit
“x” is the one bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 42) / 16
```

E.2.7.5 DPCM6 for 12–6–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 74) / 32
```

E.2.7.6 DPCM7 for 12–6–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0011 s x”
```

where,

```
“0011” is the code word
“s” is the sign bit
“x” is the one bit value field
```

The coder equation is described as follows:

```
if (Xdiff( n ) < 0) then
   sign = 1
else
   sign = 0
endif
value = (abs(Xdiff( n )) - 202) / 64
```

E.2.7.7 PCM for 12–6–12 Coder

Xenco(**n**) has the following format:

```
Xenco( n ) = “1 xxxxx”
```

where,

```
“1” is the code word
the sign bit is not used
“xxxxx” is the five bit value field
```

The coder equation is described as follows:

```
value = Xorig( n ) / 128
```

E.3 Decoders

There are six different decoders available, one for each data compression scheme.

For all decoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula for predicted pixels.

E.3.1 Decoder for 10–8–10 Data Compression

Pixels without prediction are decoded using the following formula:

```
Xdeco( n ) = 4 * Xenco( n ) + 2
```

Pixels with prediction are decoded using the following formula:

```
if (Xenco( n ) & 0xc0 == 0x00) then
    use DPCM1
else if (Xenco( n ) & 0xe0 == 0x40) then
    use DPCM2
else if (Xenco( n ) & 0xe0 == 0x60) then
    use DPCM3
else
    use PCM
endif
```

E.3.1.1 DPCM1 for 10–8–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “00 s xxxxx”
```

where,

```
“00” is the code word
“s” is the sign bit
“xxxxx” is the five bit value field
```

The decoder equation is described as follows:

```
sign = Xenco( n ) & 0x20
value = Xenco( n ) & 0x1f
if (sign > 0) then
    Xdeco( n ) = Xpred( n ) - value
else
    Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.1.2 DPCM2 for 10–8–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xxxx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The decoder equation is described as follows:

```
sign = Xenco( n ) & 0x10
value = 2 * (Xenco( n ) & 0xf) + 32
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
else
   Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.1.3 DPCM3 for 10–8–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xxxx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The decoder equation is described as follows:

```
sign = Xenco( n ) & 0x10
value = 4 * (Xenco( n ) & 0xf) + 64 + 1
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 1023) then
      Xdeco( n ) = 1023
   endif
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

Pixels without prediction are decoded using the following formula:

```
Xdeco( n ) = 8 * Xenco( n ) + 4
```

Pixels with prediction are decoded using the following formula:

```
if (Xenco( n ) & 0x70 == 0x00) then
    use DPCM1
else if (Xenco( n ) & 0x78 == 0x10) then
    use DPCM2
else if (Xenco( n ) & 0x78 == 0x18) then
    use DPCM3
else if (Xenco( n ) & 0x60 == 0x20) then
    use DPCM4
else
    use PCM
endif
```

E.3.2.1 DPCM1 for 10–7–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “000 s xxx”
```

where,

```
“000” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The codec equation is described as follows:

```
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.3.2.2 DPCM2 for 10–7–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0010 s xx”
```

where,

```
“0010” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = 2 * (Xenco( n ) & 0x3) + 8
if (sign > 0) then
    Xdeco( n ) = Xpred( n ) - value
else
    Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.2.3 DPCM3 for 10–7–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0011 s xx”
```

where,

```
“0011” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = 4 * (Xenco( n ) & 0x3) + 16 + 1
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 1023) then
      Xdeco( n ) = 1023
   endif
endif
```

E.3.2.4 DPCM4 for 10–7–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “01 s xxxx”
```

where,

```
“01” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x10
value = 8 * (Xenco( n ) & 0xf) + 32 + 3
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 1023) then
      Xdeco( n ) = 1023
   endif
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

Pixels without prediction are decoded using the following formula:

```
Xdeco( n ) = 16 * Xenco( n ) + 8
```

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

Xenco(**n**) has the following format:

```
Xenco( n ) = “00000 s”
```

where,

```
“00000” is the code word
“s” is the sign bit
the value field is not used
```

The codec equation is described as follows:

```
Xdeco( n ) = Xpred( n )
```

E.3.3.2 DPCM2 for 10–6–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “00001 s”
```

where,

```
“00001” is the code word
“s” is the sign bit
the value field is not used
```

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

Xenco(**n**) has the following format:

```
Xenco( n ) = “0001 s x”
```

where,

```
“0001” is the code word
“s” is the sign bit
“x” is the one bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x2
value = 4 * (Xenco( n ) & 0x1) + 3 + 1
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 1023) then
      Xdeco( n ) = 1023
   endif
endif
```

E.3.3.4 DPCM4 for 10–6–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “001 s xx”
```

where,

```
“001” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = 8 * (Xenco( n ) & 0x3) + 11 + 3
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 1023) then
      Xdeco( n ) = 1023
   endif
endif
```

E.3.3.5 DPCM5 for 10–6–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “01 s xxx”
```

where,

```
“01” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x8
value = 16 * (Xenco( n ) & 0x7) + 43 + 7
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 1023) then
      Xdeco( n ) = 1023
   endif
endif
```

E.3.3.6 PCM for 10–6–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “1 xxxxx”
```

where,

```
“1” is the code word
the sign bit is not used
“xxxxx” is the five bit value field
```

The codec equation is described as follows:

```
value = 32 * (Xenco( n ) & 0x1f)
if (value > Xpred( n )) then
   Xdeco( n ) = value + 15
else
   Xdeco( n ) = value + 16
endif
```

E.3.4 Decoder for 12–10–12 Data Compression

Pixels without prediction are decoded using the following formula:

```
Xdeco( n ) = 4 * Xenco( n ) + 2
```

Pixels with prediction are decoded using the following formula:

```
if (Xenco( n ) & 0x300 == 0x000) then
    use DPCM1
else if (Xenco( n ) & 0x380 == 0x100) then
    use DPCM2
else if (Xenco( n ) & 0x380 == 0x180) then
    use DPCM3
else
    use PCM
endif
```

E.3.4.1 DPCM1 for 12–10–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “00 s xxxxxxx”
```

where,

```
“00” is the code word
“s” is the sign bit
“xxxxxxx” is the seven bit value field
```

The decoder equation is described as follows:

```
sign = Xenco( n ) & 0x80
value = Xenco( n ) & 0x7f
if (sign > 0) then
    Xdeco( n ) = Xpred( n ) - value
else
    Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.4.2 DPCM2 for 12–10–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xxxxxx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xxxxxx” is the six bit value field
```

The decoder equation is described as follows:

```
sign = Xenco( n ) & 0x40
value = 2 * (Xenco( n ) & 0x3f) + 128
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
else
   Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.4.3 DPCM3 for 12–10–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xxxxxx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xxxxxx” is the six bit value field
```

The decoder equation is described as follows:

```
sign = Xenco( n ) & 0x40
value = 4 * (Xenco( n ) & 0x3f) + 256 + 1
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
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

Pixels without prediction are decoded using the following formula:

```
Xdeco( n ) = 16 * Xenco( n ) + 8
```

Pixels with prediction are decoded using the following formula:

```
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

Xenco(**n**) has the following format:

```
Xenco( n ) = “0000 s xxx”
```

where,

```
“0000” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The codec equation is described as follows:

```
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

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xxxx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The codec equation is described as follows:

```
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

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xxxx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x10
value = 4 * (Xenco( n ) & 0xf) + 40 + 1
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.5.4 DPCM4 for 12–8–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “001 s xxxx”
```

where,

```
“001” is the code word
“s” is the sign bit
“xxxx” is the four bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x10
value = 8 * (Xenco( n ) & 0xf) + 104 + 3
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095)
      Xdeco( n ) = 4095
   endif
endif
```

E.3.5.5 DPCM5 for 12–8–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0001 s xxx”
```

where,

```
“0001” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x8
value = 16 * (Xenco( n ) & 0x7) + 232 + 7
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.5.6 PCM for 12–8–12 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 = 32 * (Xenco( n ) & 0x7f)
if (value > Xpred( n )) then
   Xdeco( n ) = value + 15
else
   Xdeco( n ) = value + 16
endif
```

E.3.6 Decoder for 12–7–12 Data Compression

Pixels without prediction are decoded using the following formula:

```
Xdeco( n ) = 32 * Xenco( n ) + 16
```

Pixels with prediction are decoded using the following formula:

```
if (Xenco( n ) & 0x78 == 0x00) then
    use DPCM1
else if (Xenco( n ) & 0x78 == 0x08) then
    use DPCM2
else if (Xenco( n ) & 0x78 == 0x10) then
    use DPCM3
else if (Xenco( n ) & 0x70 == 0x20) then
    use DPCM4
else if (Xenco( n ) & 0x70 == 0x30) then
    use DPCM5
else if (Xenco( n ) & 0x78 == 0x18) then
    use DPCM6
else
    use PCM
endif
```

E.3.6.1 DPCM1 for 12–7–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0000 s xx”
```

where,

```
“0000” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = Xenco( n ) & 0x3
if (sign > 0) then
    Xdeco( n ) = Xpred( n ) - value
else
    Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.6.2 DPCM2 for 12–7–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0001 s xx”
```

where,

```
“0001” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = 2 * (Xenco( n ) & 0x3) + 4
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
else
   Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.6.3 DPCM3 for 12–7–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0010 s xx”
```

where,

```
“0010” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = 4 * (Xenco( n ) & 0x3) + 12 + 1
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.6.4 DPCM4 for 12–7–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xxx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x8
value = 8 * (Xenco( n ) & 0x7) + 28 + 3
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.6.5 DPCM5 for 12–7–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xxx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xxx” is the three bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x8
value = 16 * (Xenco( n ) & 0x7) + 92 + 7
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.6.6 DPCM6 for 12–7–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0011 s xx”
```

where,

```
“0011” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = 32 * (Xenco( n ) & 0x3) + 220 + 15
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.6.7 PCM for 12–7–12 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 = 64 * (Xenco( n ) & 0x3f)
if (value > Xpred( n )) then
   Xdeco( n ) = value + 31
else
   Xdeco( n ) = value + 32
endif
```

E.3.7 Decoder for 12–6–12 Data Compression

Pixels without prediction are decoded using the following formula:

```
Xdeco( n ) = 64 * Xenco( n ) + 32
```

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
```

*Note: **DPCM2** is not used.*

E.3.7.1 DPCM1 for 12–6–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0000 s x”
```

where,

```
“0000” is the code word
“s” is the sign bit
“x” is the one bit value field
```

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

Xenco(**n**) has the following format:

```
Xenco( n ) = “0001 s x”
```

where,

```
“0001” is the code word
“s” is the sign bit
“x” is the one bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x2
value = 4 * (Xenco( n ) & 0x1) + 2 + 1
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.7.3 DPCM4 for 12–6–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “010 s xx”
```

where,

```
“010” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = 8 * (Xenco( n ) & 0x3) + 10 + 3
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.7.4 DPCM5 for 12–6–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0010 s x”
```

where,

```
“0010” is the code word
“s” is the sign bit
“x” is the one bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x2
value = 16 * (Xenco( n ) & 0x1) + 42 + 7
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.7.5 DPCM6 for 12–6–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “011 s xx”
```

where,

```
“011” is the code word
“s” is the sign bit
“xx” is the two bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = 32 * (Xenco( n ) & 0x3) + 74 + 15
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.7.6 DPCM7 for 12–6–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “0011 s x”
```

where,

```
“0011” is the code word
“s” is the sign bit
“x” is the one bit value field
```

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x2
value = 64 * (Xenco( n ) & 0x1) + 202 + 31
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) - value
   if (Xdeco( n ) < 0) then
      Xdeco( n ) = 0
   endif
else
   Xdeco( n ) = Xpred( n ) + value
   if (Xdeco( n ) > 4095) then
      Xdeco( n ) = 4095
   endif
endif
```

E.3.7.7 PCM for 12–6–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = “1 xxxxx”
```

where,

```
“1” is the code word
the sign bit is not used
“xxxxx” is the five bit value field
```

The codec equation is described as follows:

```
value = 128 * (Xenco( n ) & 0x1f)
if (value > Xpred( n )) then
   Xdeco( n ) = value + 63
else
   Xdeco( n ) = value + 64
endif
```

Annex F JPEG Interleaving (informative)

This annex illustrates how the standard features of the CSI-2 protocol should be used to interleave (multiplex) JPEG image data with other types of image data, e.g. RGB565 or YUV422, without requiring a custom JPEG format such as JPEG8.

The Virtual Channel Identifier and Data Type value in the CSI-2 Packet Header provide simple methods of interleaving multiple data streams or image data types at the packet level. Interleaving at the packet level minimizes the amount of buffering required in the system.

The Data Type value in the CSI-2 Packet Header should be used to multiplex different image data types at the CSI-2 transmitter and de-multiplex the data types at the CSI-2 receiver.

The Virtual Channel Identifier in the CSI-2 Packet Header should be used to multiplex different data streams (channels) at the CSI-2 transmitter and de-multiplex the streams at the CSI-2 receiver.

The main difference between the two interleaving methods is that images with different Data Type values within the same Virtual Channel use the same frame and line synchronization information, whereas multiple Virtual Channels (data streams) each have their own independent frame and line synchronization information and thus potentially each channel may have different frame rates.

Since the predefined Data Type values represent only YUV, RGB and RAW data types, one of the User Defined Data Type values should be used to represent JPEG image data.

Figure 191 illustrates interleaving JPEG image data with YUV422 image data using Data Type values.

Figure 192 illustrates interleaving JPEG image data with YUV422 image data using both Data Type values and Virtual Channel Identifiers.

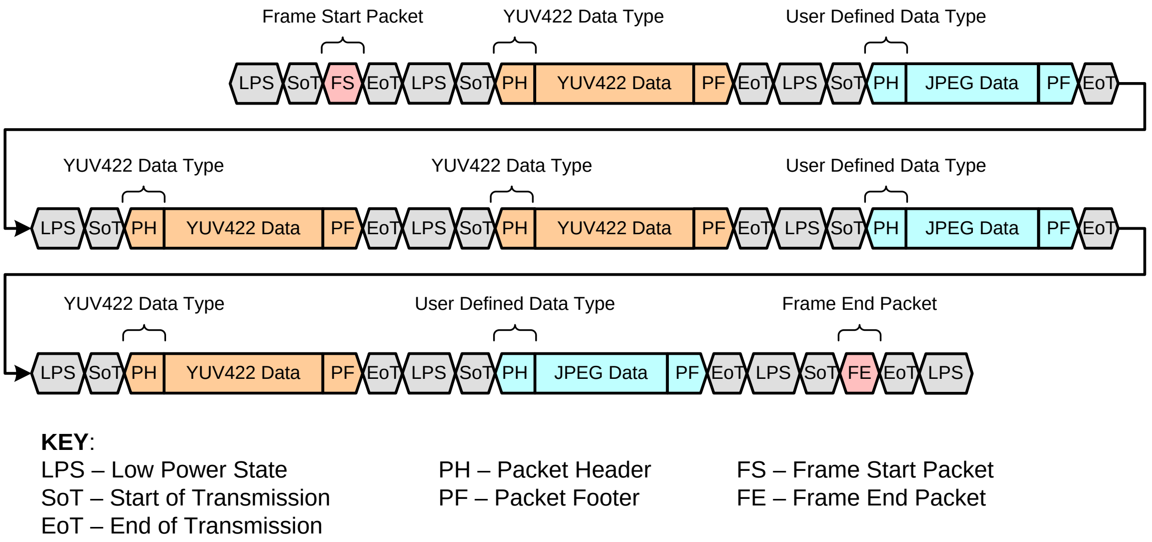


Figure 191 Data Type Interleaving: Concurrent JPEG and YUV Image Data

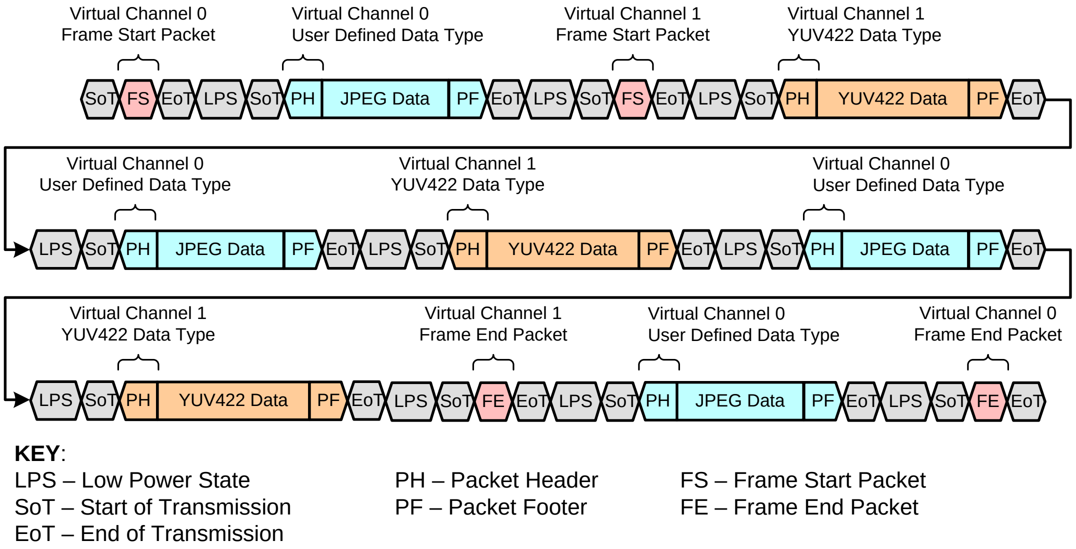


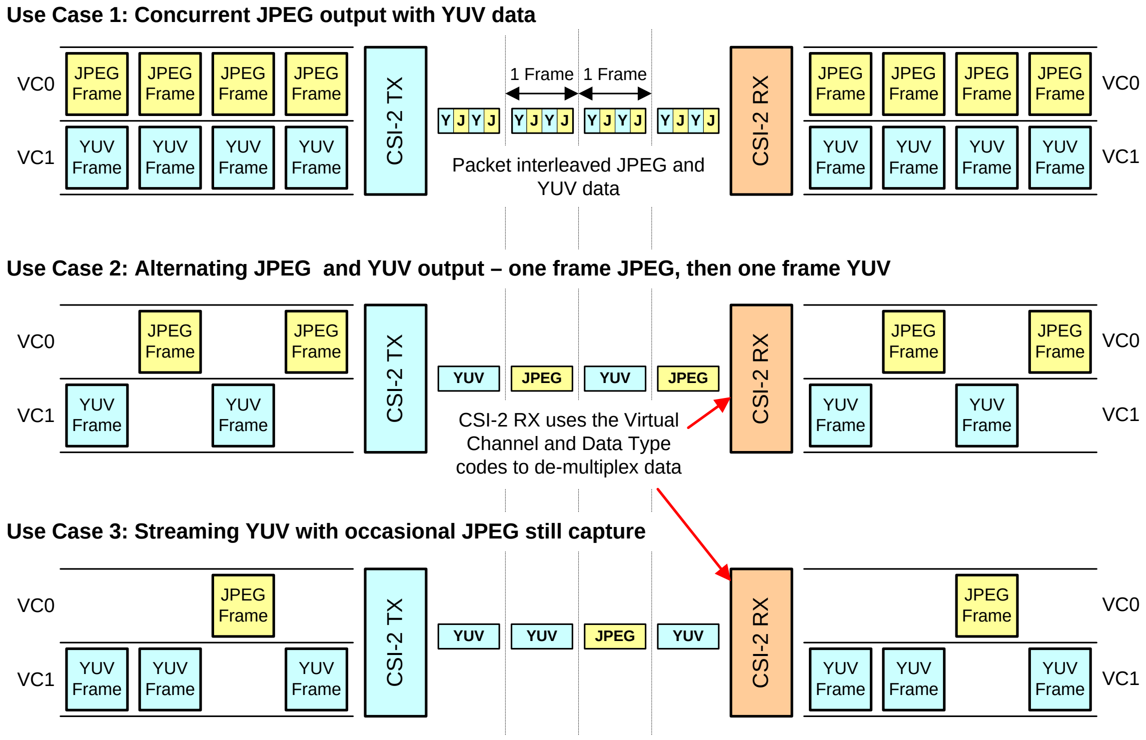
Figure 192 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data

Both ***Figure 191*** and ***Figure 192*** can be similarly extended to the interleaving of JPEG image data with any other type of image data, e.g. RGB565.

Figure 193 illustrates the use of Virtual Channels to support three different JPEG interleaving usage cases:

- Concurrent JPEG and YUV422 image data.
- Alternating JPEG and YUV422 output - one frame JPEG, then one frame YUV
- Streaming YUV22 with occasional JPEG for still capture

Again, these examples could also represent interleaving JPEG data with any other image data type.


Use Case 1: Concurrent JPEG output with YUV data
VC0
VC1
JPEG Frame
YUV Frame
CSI-2 TX
1 Frame
1 Frame
Y J Y J
Packet interleaved JPEG and YUV data
CSI-2 RX
Use Case 2: Alternating JPEG and YUV output – one frame JPEG, then one frame YUV
YUV
JPEG
CSI-2 RX uses the Virtual Channel and Data Type codes to de-multiplex data
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.

Annex H Guidance on CSI-2 Over C-PHY ALP and PPI

H.1 CSI-2 with C-PHY ALP Mode

C-PHY Alternate Low Power (ALP) Mode is an alternative to the legacy LP mode of C-PHY. ALP Mode uses solely High-Speed signaling with a special state where the signals can cease toggling and collapse to zero. The legacy LP Mode signaling and escape sequences have equivalent ALP Mode functions so that the high-voltage low power signaling can be replaced by ALP Mode signaling if that is beneficial in specific systems. ALP Mode replaces the legacy LP Mode line levels by the transmission of unique code words that are used only for Lane signaling events. These unique codes are never produced by the 3-Phase mapping function, so there is never ambiguity in the interpretation of these codes at the receiver.

Reasons to replace the legacy LP mode with equivalent ALP Mode functions are to begin a transitionary path to the future so that legacy LP mode might someday be eliminated in some devices. Another reason to choose ALP Mode over Legacy LP mode is to support systems that have long interconnect between the Master and Slave devices.

H.1.1 Concepts of ALP Mode and Legacy LP Mode

In ALP mode, the conventional LP receivers are not used to detect signaling states. Instead, all communication is performed using High-Speed signaling levels. The system level functions performed by ALP signaling are quite similar to the functional behavior of legacy LP mode. The intent of this is to cause the least amount of disruption to systems that support both ALP Mode and legacy LP mode. ***Figure 194*** shows a comparison of a High-Speed data burst with LP Mode versus ALP Mode. The purpose of this diagram is to show that each of the intervals in the High-Speed data burst with LP mode correspond to similar intervals in the High-Speed data burst with ALP mode.

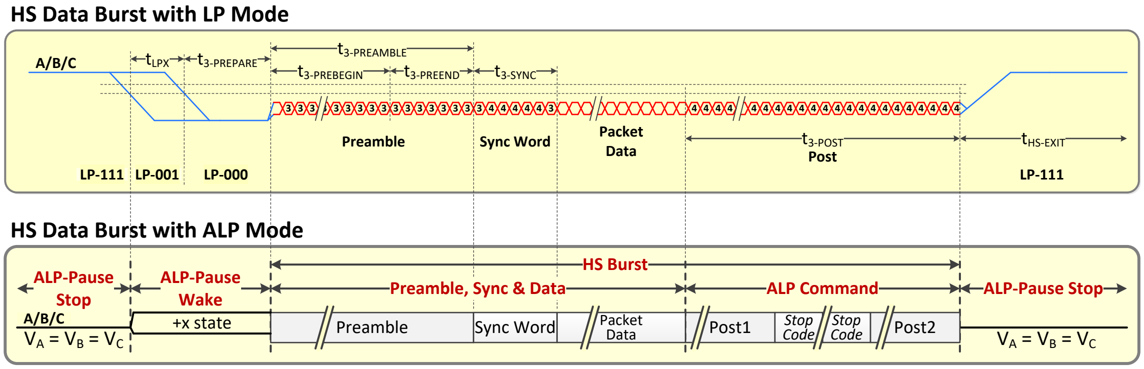


Figure 194 Comparing Data Burst Timing of Legacy LP mode versus ALP Mode

ALP Mode supports the transmission of High-Speed data bursts as well as the transmission of control sequences that are traditionally transmitted using legacy LP mode Escape Mode sequences. The format of all ALP mode bursts is like the timing diagram in ***Figure 195***.

The burst begins and ends in an ALP-Pause state. There are two types of ALP-Pause: ALP-Pause Stop and ALP-Pause ULPS. ALP-Pause Stop is analogous to the legacy LP mode Stop state; ALP-Pause ULPS is analogous to the legacy LP mode ULPS state. The only difference between these two types of ALP-Pause states is the time allowed to wake up from each, which is the duration of the ALP-Pause Wake interval. The nominal time allowed to wave from ALP-Pause Stop is 100 ns, which is about the same time as the duration of the LP-001 and LP-000 states at the beginning of a HS Data Burst using legacy LP mode. The nominal time to wake from the ALP-Pause ULPS state is 1 msec, which is approximately the time allowed in legacy LP mode for t_{WAKEUP}. (The time that a transmitter drives a Mark-1 state prior to a Stop state to initiate an exit from ULPS.) The longer wake-up time from ALP-Pause ULPS compared to ALP-Pause Stop allows a lower power consumption while in the ALP-Pause ULPS state.

The ALP-Pause Stop and ALP-Pause ULPS line states are defined by the following relationships of the 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-Pause Wake states are illustrated at the beginning and end of the waveform in ***Figure 195***. The ALP-Pause Wake state, which is very long compared to a High-Speed Unit Interval, is detected by the low-power wake-up receiver. This causes the system to leave one of the ALP-Pause states and to begin receiving a High-Speed signal.

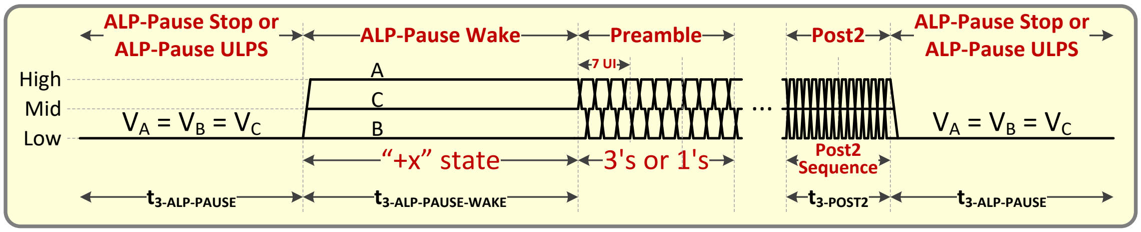


Figure 195 ALP Mode General Burst Format

To minimize power consumption while Lane activity has ceased during one of the ALP-Pause states, a special low-speed and low-power differential receiver circuit is present, in addition to the three High-Speed differential receivers for A-B, B-C and C-A. This special low-speed and low-power differential receiver has a nominal +80 mV offset input threshold voltage that detects the difference in differential levels between the ALP-Pause state ($V_{OD} = 0$) and ALP-Pause Wake state ($V_{OD} = |V_{OD}|$ Strong). This allows the line signals to collapse to zero with the 100Ω Z_{ID} termination still connected, and still have a well-defined method to detect the difference between the ALP-Pause and ALP-Pause Wake line conditions. Collapsing to zero with the terminations still connected makes it possible for implementations to have very low power consumption during the ALP-Pause states. The ALP-Pause Wake pulse is very long compared to a High-Speed Unit Interval so that the wake receiver can be slow and consume very little power compared to the High-Speed differential receivers.

An example of the differential receiver circuit to support ALP mode is shown in ***Figure 196***. Two different offset receivers are shown for wake from stop versus wake from ULPS, because the power consumption in the ALP-Pause ULPS state is expected to be lower than in ALP-Pause Stop state. The ALP-Pause Wake pulse from the ULPS state can be longer than waking from ALP-Pause Stop, so the ALP ULPS receiver can be slower and consume less power compared to the ALP Stop receiver.

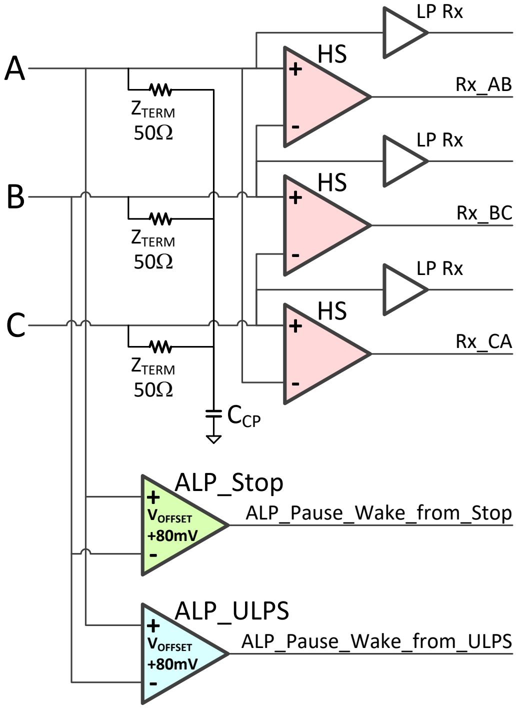


Figure 196 High-Speed and ALP-Pause Wake Receiver Example

The C-PHY specification defines twelve unique 7-symbol ALP Code Words that are the functional equivalent of the LP pulse sequences of legacy LP mode. In some cases, a single 7-symbol ALP Code Word can replace the transmission of a long sequence of legacy LP mode pulses, such as for the transmission of Escape Mode triggers or low-power data transmission. The CSI-2 specification needs only three of these LP mode pulse sequences to emulate the functionality of legacy LP mode: Stop Code, ULPS Code, and Post.

Exit from and entry into the ALP-Pause state, which is the functional equivalent of the legacy LP mode Stop state, requires a special ALP Mode sequence consisting of one or more Stop Codes or ULPS codes followed by a string of Post codes followed by setting the voltage of all three Lines of a Lane to the same value.

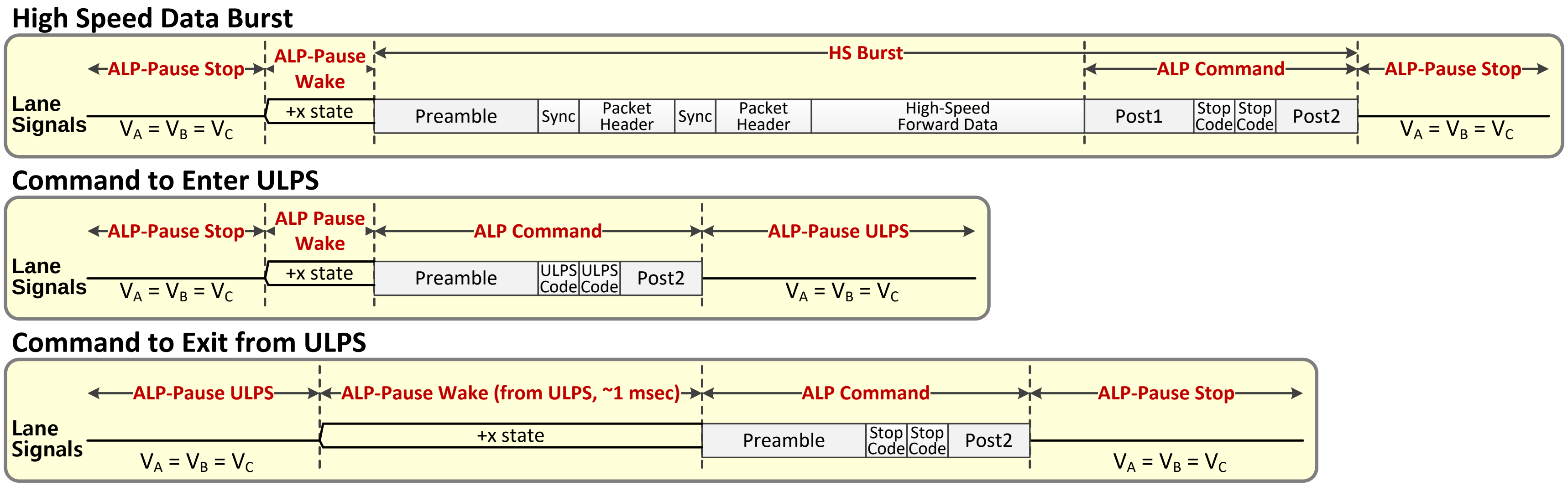
As illustrated in ***Figure 194***, the burst starting sequence of the legacy LP mode consisting of: LP-111, LP-001, and LP-000 followed by preamble, has a functional equivalent sequence in ALP Mode consisting of: ALP-Pause Stop followed by ALP Pause Wake followed by preamble. Similarly, the burst ending sequence of legacy LP mode consisting of Post sequence followed by LP-111, has a functional equivalent sequence in ALP Mode consisting of: the Post1 field by two or more Stop Codes followed by the Post2 field 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.



3870

Figure 197 Examples of Bursts to Send High-Speed Data and ALP Commands

Figure 198 shows the ALP state machine transitions (highlighted in red) necessary to transmit a High-Speed data burst in ALP mode. States and state transitions that are not used by CSI-2 for any type of burst are shown using dashed lines. The red highlighted states and transitions indicate the path required to transmit and receive the High-Speed Data Burst example in ***Figure 197***.

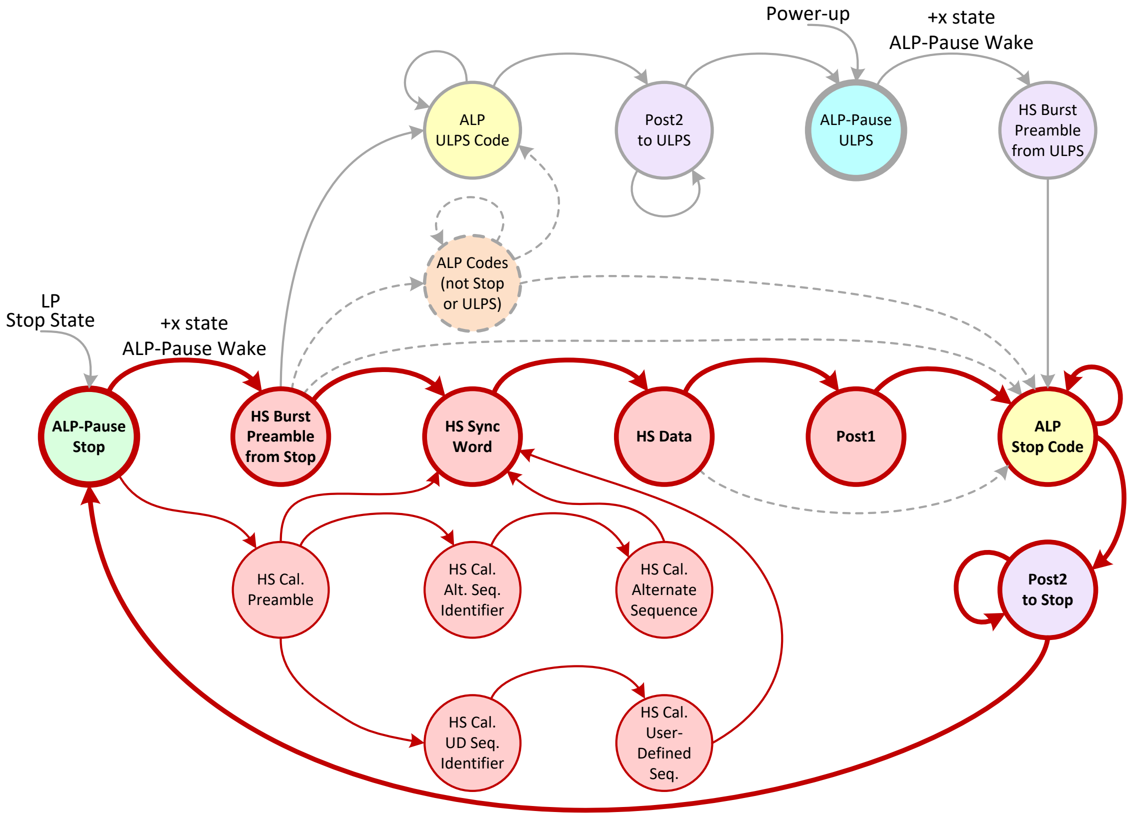
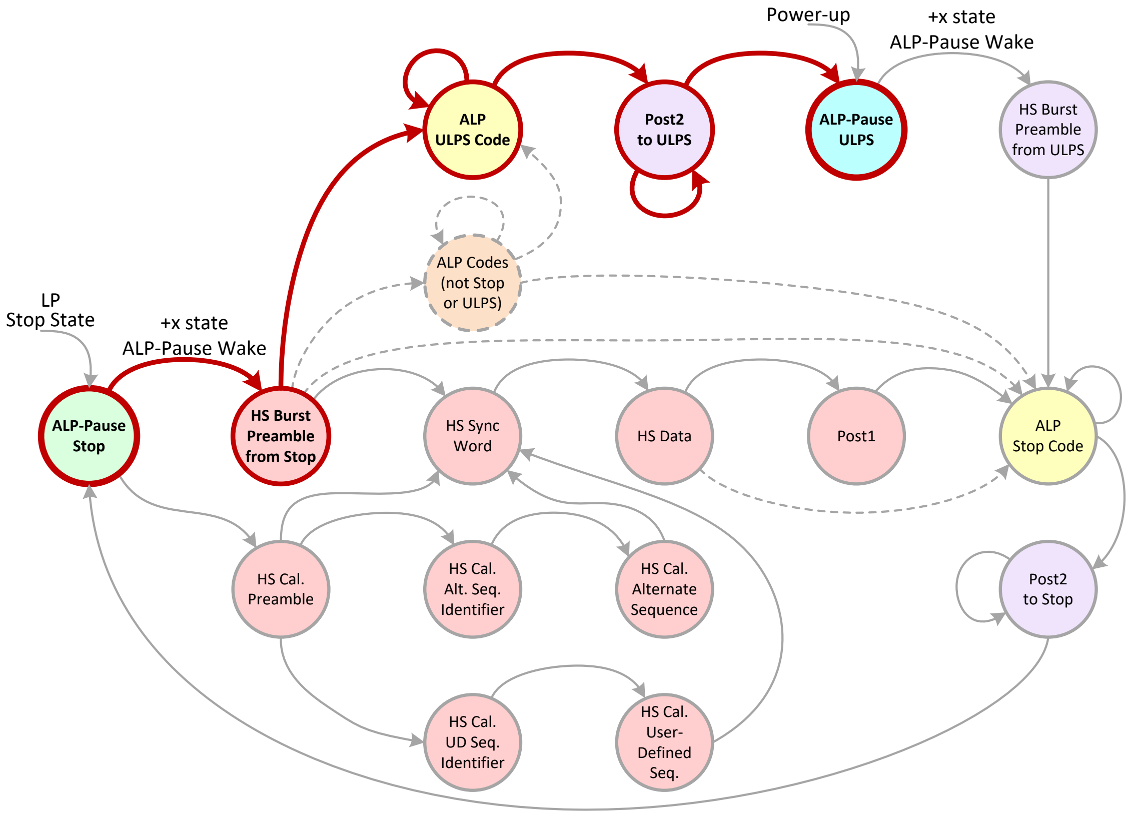


Figure 198 State Transitions for an HS Data Burst

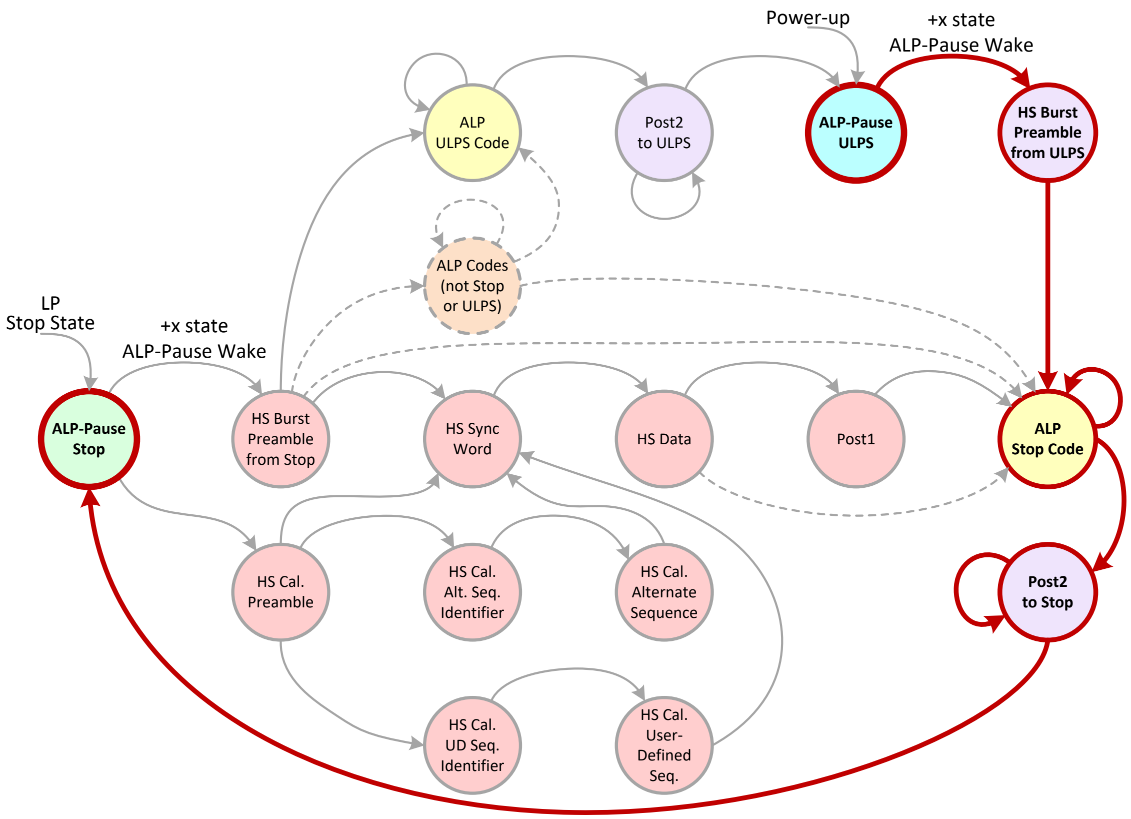
3876 ***Figure 199*** shows the ALP state machine transitions (highlighted in red) necessary to enter the ALP-Pause
3877 ULPS state.



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.


Power-up
+x state
ALP-Pause Wake
ALP ULPS Code
Post2 to ULPS
ALP-Pause ULPS
HS Burst Preamble from ULPS
ALP Codes (not Stop or ULPS)
LP Stop State
+x state
ALP-Pause Wake
ALP-Pause Stop
HS Burst Preamble from Stop
HS Sync Word
HS Data
Post1
ALP Stop Code
Post2 to Stop
HS Cal. Preamble
HS Cal. Alt. Seq. Identifier
HS Cal. Alternate Sequence
HS Cal. UD Seq. Identifier
HS Cal. User-Defined Seq.

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

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***.

These three different types of code words comprise a high speed burst while in ALP mode. ***Figure 201*** highlights the control signals that facilitate the transmission of each of these three different types of code words.

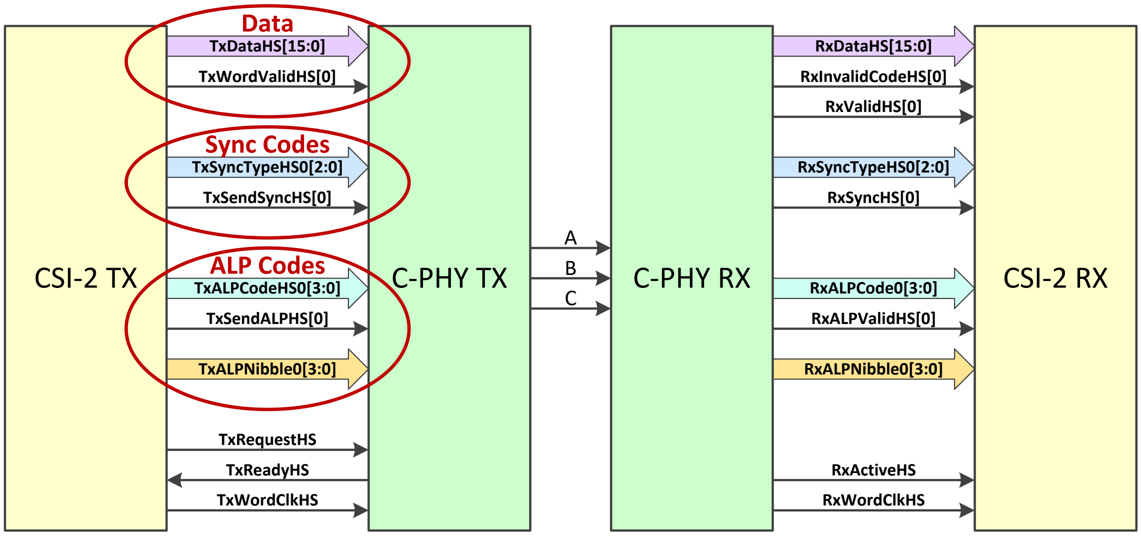


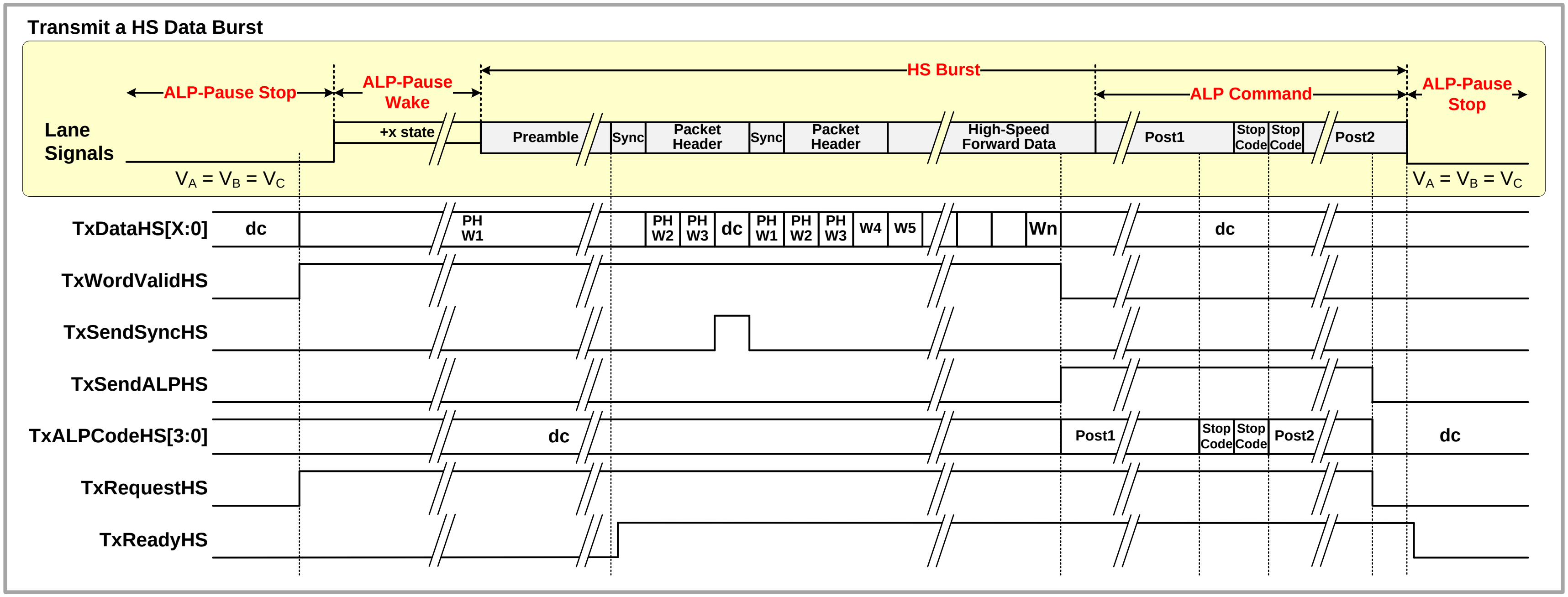
Figure 201 PPI Example: HS Signals for Transmission of Data, Sync and ALP Commands

Figure 202 and ***Figure 203*** show examples of PPI signals and the corresponding PHY data for transmission and reception of high speed data in ALP mode. These figures show additional detail of the High-Speed Data Burst waveform in ***Figure 197***.

The signal TxRequestHS is asserted simultaneously with TxWordValidHS to request that a high speed burst be transmitted. The PHY will know to send a data burst because TxWordValidHS is asserted early in the burst timing. This will cause the C-PHY Tx to transmit the first Sync Word at the end of the Preamble. Note that the first Sync Word is transmitted autonomously by the C-PHY Tx, and has the default Sync Type value of 3. Subsequent Sync Words transmitted in a burst are sent as a result of asserting the TxSendSyncHS[0] signal, and the associated Sync Type is defined by the TxSyncTypeHS0[2:0] signals.

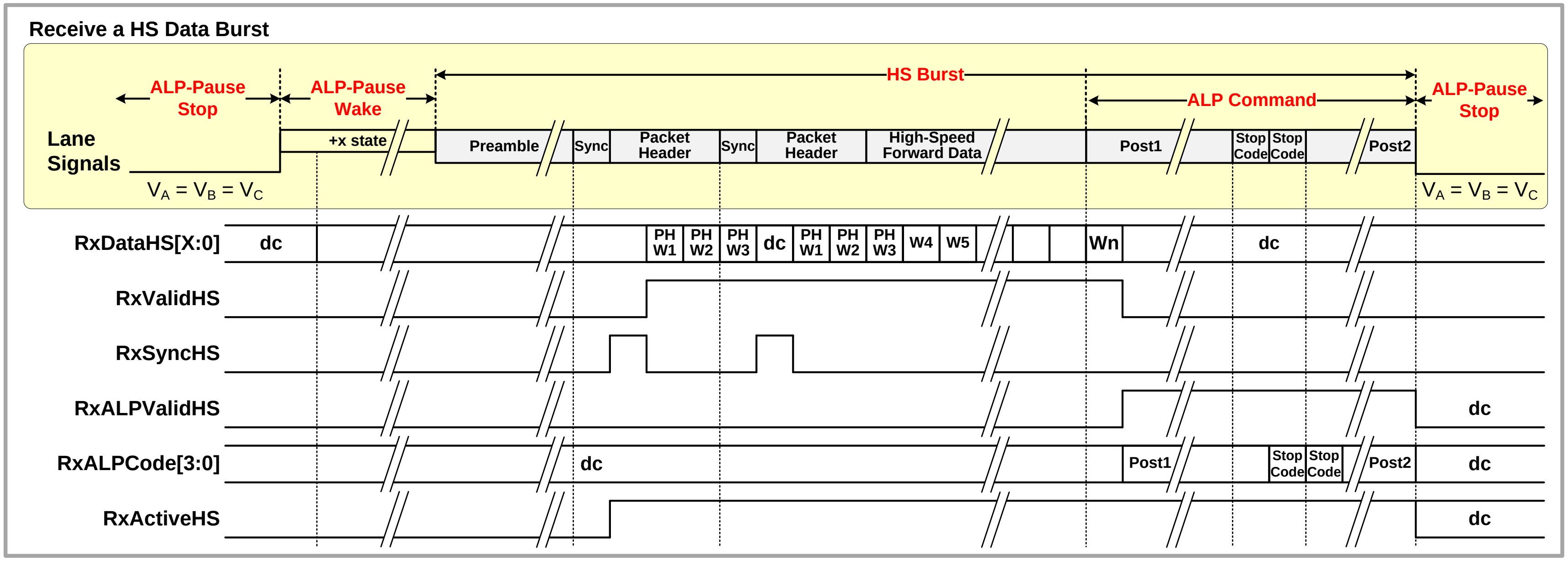
The end of burst in the Transmitter functions differently for ALP mode compared to the non-ALP high-speed mode. In the non-ALP high-speed mode, the end of burst is signaled to the PHY by pulling TxRequestHS low, as described in Annex A of the C-PHY specification. After TxRequestHS goes low, the C-PHY Tx will generate the Post sequence of length determined by a PHY configuration parameter that sets the length of Post.

In ALP mode, the protocol transmit unit generates all fields of the burst after the first sync word, including the packet headers, data burst, Stop Code, ULPS Code, Post1, and Post2. The burst is ended by pulling TxRequestHS low, and no additional data is transmitted on the Lane after this time.



3914

Figure 202 PPI Example Transmit Side Timing for an HS Data Burst



3915

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.

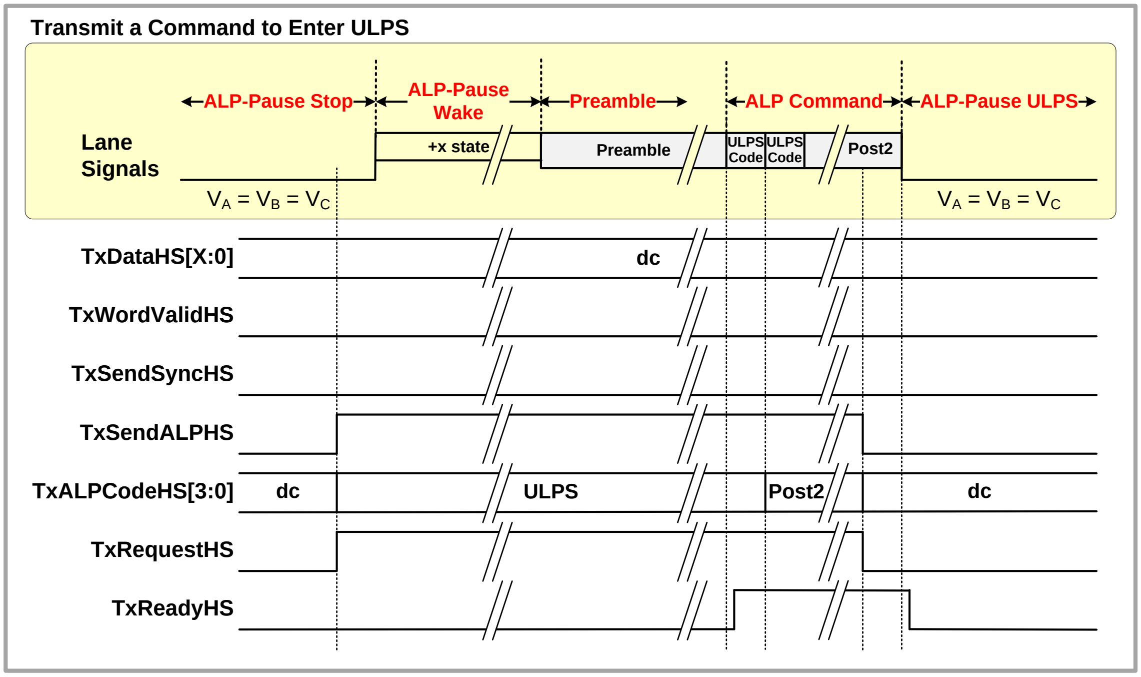


Figure 204 PPI Example Transmit Side Timing to Enter the ULPS State

3924

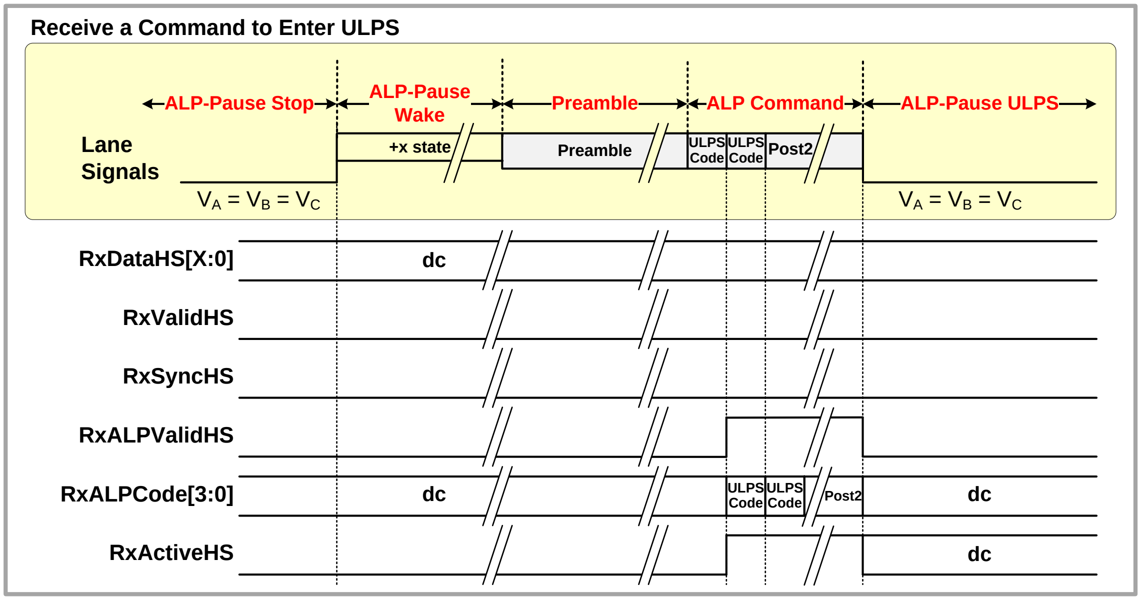


Figure 205 PPI Example Receive Side Timing to Enter the ULPS State

3925

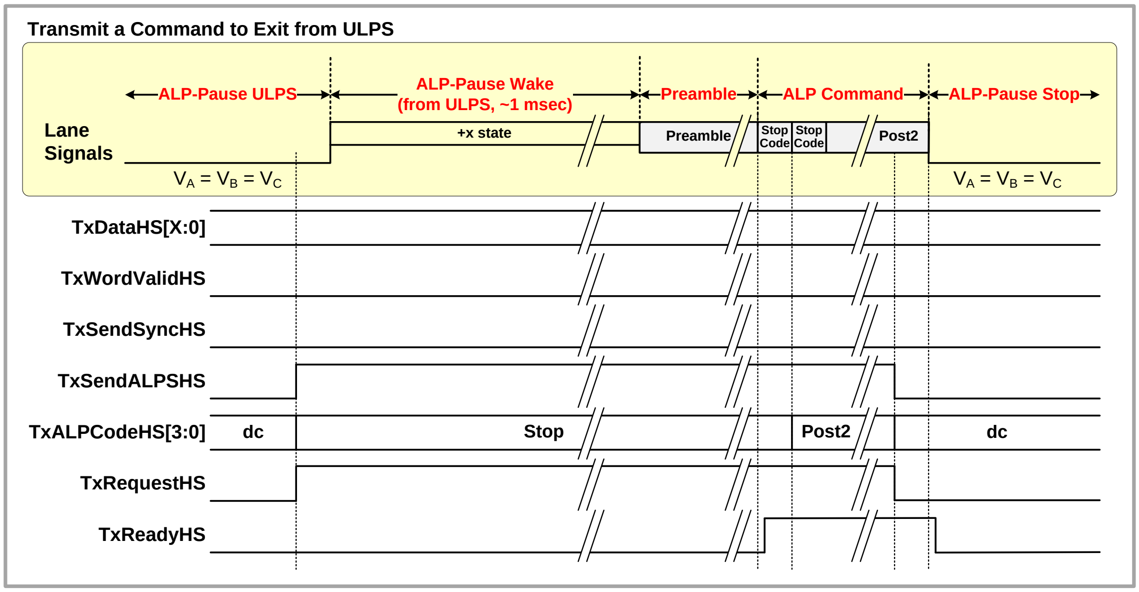
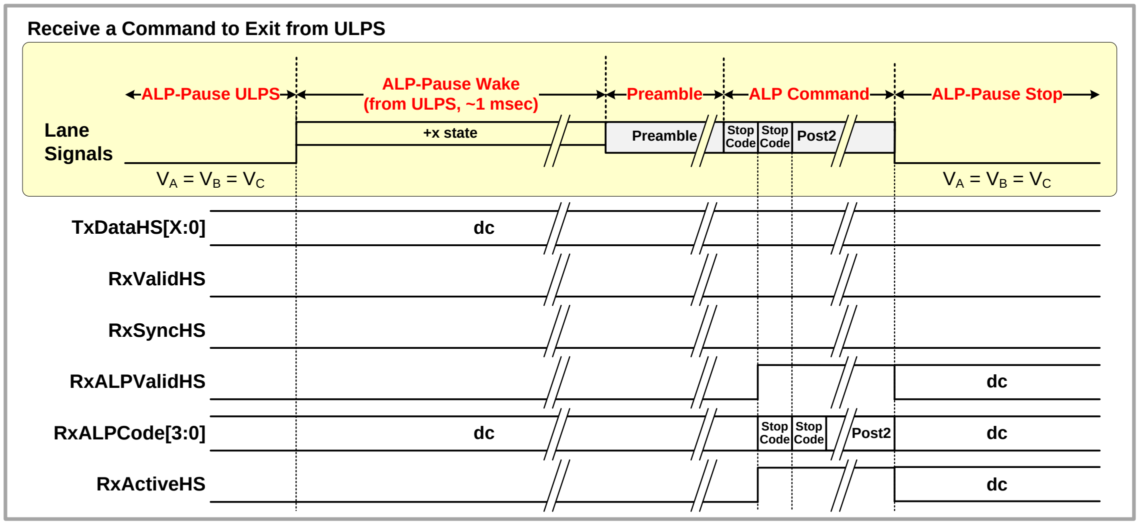


Figure 206 PPI Example Transmit Side Timing to Exit from the ULPS State



3926

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.

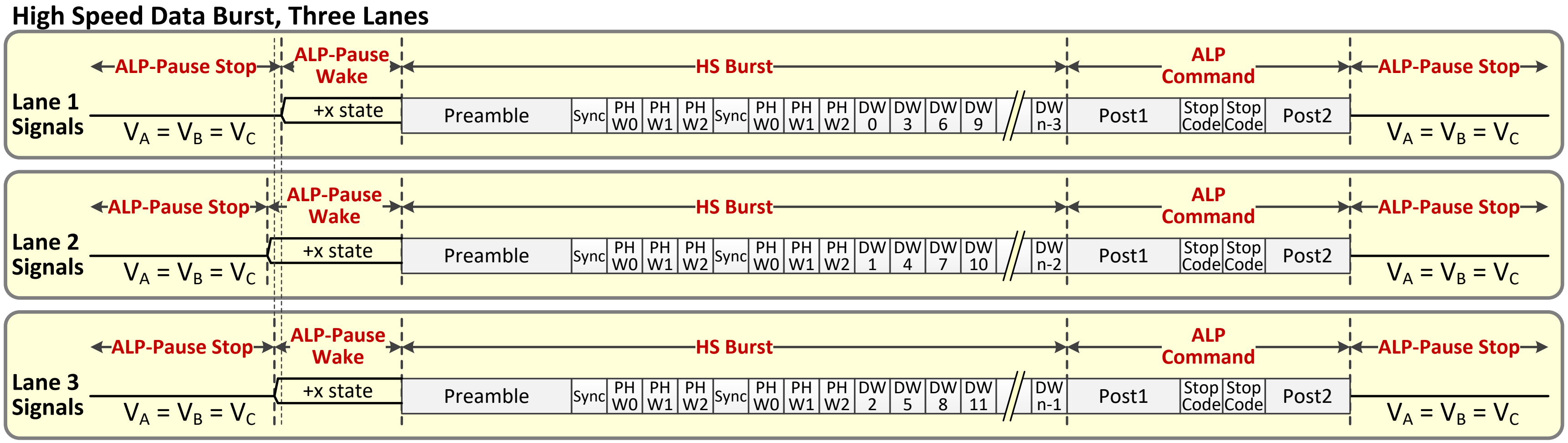


Figure 208 Example Showing a Data Transmission Burst using Three Lanes

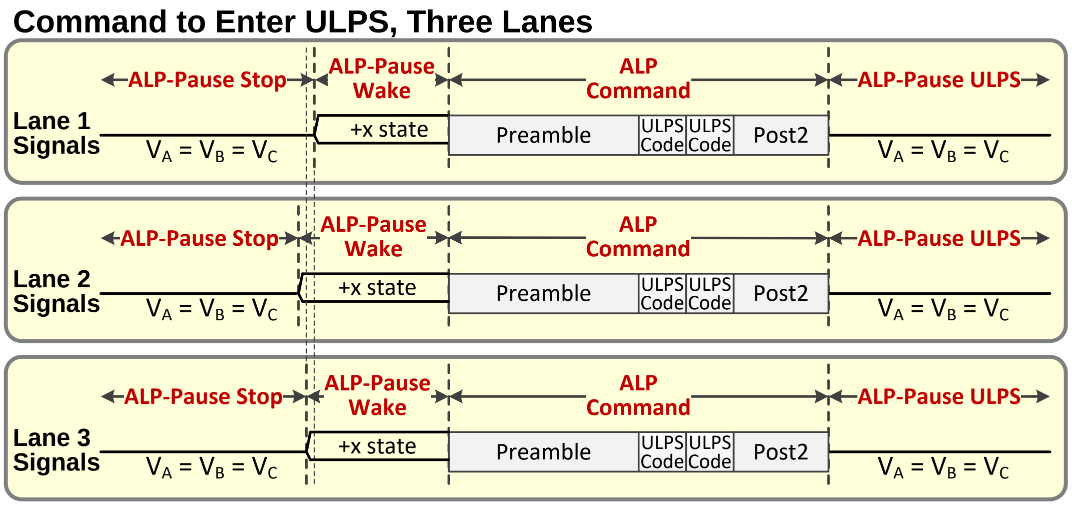


Figure 209 Example Showing an ALP Command Burst using Three Lanes

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***.

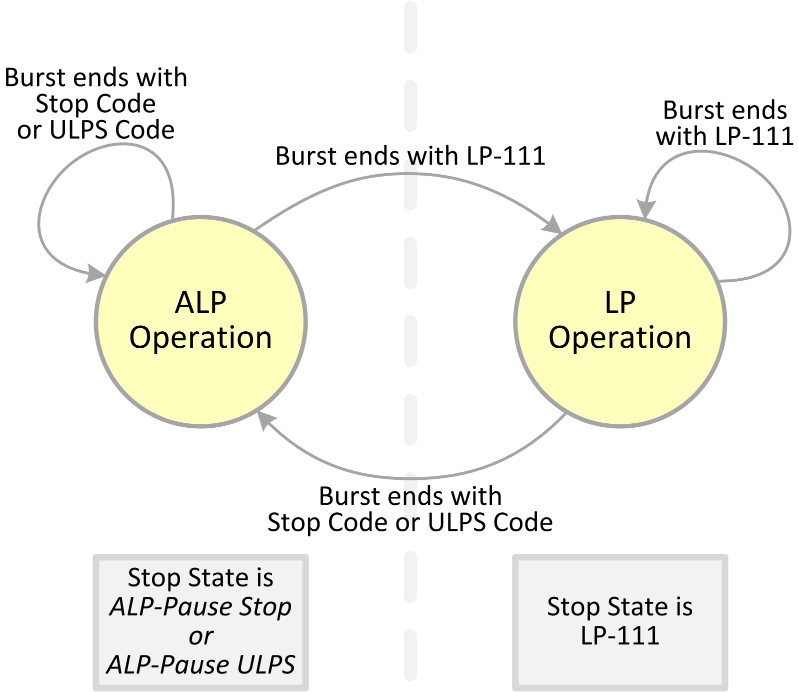


Figure 210 Automatic Selection of ALP Operation or LP Operation

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.