



**DRAFT Specification for
Camera Serial Interface 2
(CSI-2SM)**

Version 2.0

Revision 08

10 November 2016

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

This document is subject to further editorial and technical development.

Copyright © 2005-2016 MIPI Alliance, Inc.

All rights reserved.

Confidential

NOTICE OF DISCLAIMER

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®. The material contained herein is provided on an “AS IS” basis and 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 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.

All materials contained herein are 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, tradenames, and other intellectual property are the exclusive property of MIPI Alliance and cannot be used without its express prior written permission.

ALSO, THERE IS NO WARRANTY OF CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THIS MATERIAL OR THE CONTENTS OF THIS DOCUMENT. IN NO EVENT WILL ANY AUTHOR OR DEVELOPER OF THIS MATERIAL OR THE CONTENTS OF THIS DOCUMENT 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, SPECIFICATION OR DOCUMENT RELATING TO THIS MATERIAL, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Without limiting the generality of this Disclaimer stated above, the user of the contents of this Document is further notified that MIPI: (a) does not evaluate, test or verify the accuracy, soundness or credibility of the contents of this Document; (b) does not monitor or enforce compliance with the contents of this Document; and (c) does not certify, test, or in any manner investigate products or services or any claims of compliance with the contents of this Document. The use or implementation of the contents of this Document 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 Document or otherwise.

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

MIPI Alliance, Inc.
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08854
Attn: Board Secretary

10-Nov-2016

Contents

Figures.....	viii
Tables.....	xiv
Release History	xvi
Development History	xvii
1 Introduction	1
1.1 Scope	1
1.2 Purpose	1
2 Terminology	2
2.1 Use of Special Terms	2
2.2 Definitions	2
2.3 Abbreviations.....	3
2.4 Acronyms.....	3
3 References	5
4 Overview of CSI-2	7
5 CSI-2 Layer Definitions	9
6 Camera Control Interface (CCI)	11
6.1 Data Transfer Protocol.....	12
6.1.1 Message Type	12
6.1.2 Read/Write Operations	13
6.2 CCI Slave Addresses.....	16
6.3 CCI Multi-Byte Registers	16
6.3.1 Overview	16
6.3.2 The Transmission Byte Order for Multi-byte Register Values	18
6.3.3 Multi-Byte Register Protocol.....	18
6.4 Electrical Specifications and Timing for I/O Stages.....	23
7 Physical Layer	27
7.1 D-PHY Physical Layer Option	27
7.2 C-PHY Physical Layer Option.....	28
8 Multi-Lane Distribution and Merging	29
8.1 Lane Distribution for the D-PHY Physical Layer Option.....	33
8.2 Lane Distribution for the C-PHY Physical Layer Option	37
8.3 Multi-Lane Interoperability	39
8.3.1 C-PHY Lane De-Skew	41

9	Low Level Protocol.....	43
9.1	Low Level Protocol Packet Format	44
9.1.1	Low Level Protocol Long Packet Format.....	44
9.1.2	Low Level Protocol Short Packet Format	49
9.2	Data Identifier (DI)	50
9.3	Virtual Channel Identifier	51
9.4	Data Type (DT).....	53
9.5	Packet Header Error Correction Code for D-PHY Physical Layer Option.....	54
9.5.1	General Hamming Code Applied to Packet Header	55
9.5.2	Hamming-Modified Code	56
9.5.3	ECC Generation on TX Side	59
9.5.4	Applying ECC on RX Side (Informative)	60
9.6	Checksum Generation.....	62
9.7	Packet Spacing.....	64
9.8	Synchronization Short Packet Data Type Codes.....	65
9.8.1	Frame Synchronization Packets.....	65
9.8.2	Line Synchronization Packets	66
9.9	Generic Short Packet Data Type Codes	68
9.10	Packet Spacing Examples	69
9.11	Latency Reduction and Transport Efficiency (LRTE)	72
9.11.1	Interpacket Latency Reduction (ILR).....	72
9.11.2	Using ILR and Enhanced Transport Efficiency Together.....	77
9.11.3	LRTE Register Tables.....	78
9.12	Data Scrambling	80
9.13	Packet Data Payload Size Rules	90
9.14	Frame Format Examples	91
9.15	Data Interleaving	94
9.15.1	Data Type Interleaving	94
9.15.2	Virtual Channel Identifier Interleaving.....	97
10	Color Spaces	99
10.1	RGB Color Space Definition	99
10.2	YUV Color Space Definition.....	99

10-Nov-2016

11	Data Formats	101
11.1	Generic 8-bit Long Packet Data Types	103
11.1.1	Null and Blanking Data	103
11.1.2	Embedded Information	104
11.2	YUV Image Data	105
11.2.1	Legacy YUV420 8-bit	106
11.2.2	YUV420 8-bit	109
11.2.3	YUV420 10-bit	112
11.2.4	YUV422 8-bit	114
11.2.5	YUV422 10-bit	116
11.3	RGB Image Data	118
11.3.1	RGB888	119
11.3.2	RGB666	121
11.3.3	RGB565	123
11.3.4	RGB555	125
11.3.5	RGB444	126
11.4	RAW Image Data	127
11.4.1	RAW6	128
11.4.2	RAW7	129
11.4.3	RAW8	130
11.4.4	RAW10	131
11.4.5	RAW12	133
11.4.6	RAW14	134
11.4.7	RAW16	136
11.4.8	RAW20	137
11.5	User Defined Data Formats	139
12	Recommended Memory Storage	141
12.1	General/Arbitrary Data Reception	141
12.2	RGB888 Data Reception	142
12.3	RGB666 Data Reception	142
12.4	RGB565 Data Reception	143
12.5	RGB555 Data Reception	143
12.6	RGB444 Data Reception	144
12.7	YUV422 8-bit Data Reception	144
12.8	YUV422 10-bit Data Reception	145
12.9	YUV420 8-bit (Legacy) Data Reception	146
12.10	YUV420 8-bit Data Reception	147
12.11	YUV420 10-bit Data Reception	148
12.12	RAW6 Data Reception	149
12.13	RAW7 Data Reception	149
12.14	RAW8 Data Reception	150
12.15	RAW10 Data Reception	150
12.16	RAW12 Data Reception	151
12.17	RAW14 Data Reception	151
12.18	RAW16 Data Reception	152
12.19	RAW20 Data Reception	152

Annex A	JPEG8 Data Format (informative).....	155
A.1	Introduction.....	155
A.2	JPEG Data Definition	156
A.3	Image Status Information	157
A.4	Embedded Images.....	159
A.5	JPEG8 Non-standard Markers	160
A.6	JPEG8 Data Reception	160
Annex B	CSI-2 Implementation Example (informative)	161
B.1	Overview.....	161
B.2	CSI-2 Transmitter Detailed Block Diagram	162
B.3	CSI-2 Receiver Detailed Block Diagram.....	163
B.4	Details on the D-PHY Implementation.....	164
B.4.1	CSI-2 Clock Lane Transmitter.....	165
B.4.2	CSI-2 Clock Lane Receiver.....	166
B.4.3	CSI-2 Data Lane Transmitter.....	167
B.4.4	CSI-2 Data Lane Receiver.....	169
Annex C	CSI-2 Recommended Receiver Error Behavior (informative)	171
C.1	Overview.....	171
C.2	D-PHY Level Error.....	172
C.3	Packet Level Error	173
C.4	Protocol Decoding Level Error.....	174
Annex D	CSI-2 Sleep Mode (informative)	175
D.1	Overview.....	175
D.2	SLM Command Phase	175
D.3	SLM Entry Phase.....	176
D.4	SLM Exit Phase	176
Annex E	Data Compression for RAW Data Types (normative)	177
E.1	Predictors	179
E.1.1	Predictor1	179
E.1.2	Predictor2	180
E.2	Encoders	181
E.2.1	Coder for 10–8–10 Data Compression	181
E.2.2	Coder for 10–7–10 Data Compression	183
E.2.3	Coder for 10–6–10 Data Compression	186
E.2.4	Coder for 12–10–12 Data Compression	189
E.2.5	Coder for 12–8–12 Data Compression	191
E.2.6	Coder for 12–7–12 Data Compression	194
E.2.7	Coder for 12–6–12 Data Compression	197
E.3	Decoders	200
E.3.1	Decoder for 10–8–10 Data Compression	200
E.3.2	Decoder for 10–7–10 Data Compression	203
E.3.3	Decoder for 10–6–10 Data Compression	206
E.3.4	Decoder for 12–10–12 Data Compression	209
E.3.5	Decoder for 12–8–12 Data Compression	212
E.3.6	Decoder for 12–7–12 Data Compression	215
E.3.7	Decoder for 12–6–12 Data Compression	219

Annex F JPEG Interleaving (informative)223
Annex G Scrambler Seeds for Lanes 9 through 32227

Figures

Figure 1 CSI-2 and CCI Transmitter and Receiver Interface for D-PHY	7
Figure 2 CSI-2 and CCI Transmitter and Receiver Interface for C-PHY.....	8
Figure 3 CSI-2 Layer Definitions.....	9
Figure 4 CCI Message Types.....	12
Figure 5 CCI Single Read from Random Location	13
Figure 6 CCI Single Read from Current Location	13
Figure 7 CCI Sequential Read Starting from a Random Location	14
Figure 8 CCI Sequential Read Starting from the Current Location	14
Figure 9 CCI Single Write to a Random Location	15
Figure 10 CCI Sequential Write Starting from a Random Location	15
Figure 11 Corruption of a 32-bit Wide Register during a Read Message.....	17
Figure 12 Corruption of a 32-bit Wide Register during a Write Message.....	17
Figure 13 Example 16-bit Register Write.....	18
Figure 14 Example 32-bit Register Write (address not shown).....	18
Figure 15 Example 64-bit Register Write (address not shown).....	18
Figure 16 Example 16-bit Register Read	19
Figure 17 Example 32-bit Register Read	20
Figure 18 Example 16-bit Register Write.....	21
Figure 19 Example 32-bit Register Write.....	22
Figure 20 CCI Timing	25
Figure 21 Conceptual Overview of the Lane Distributor Function for D-PHY	29
Figure 22 Conceptual Overview of the Lane Distributor Function for C-PHY	30
Figure 23 Conceptual Overview of the Lane Merging Function for D-PHY	31
Figure 24 Conceptual Overview of the Lane Merging Function for C-PHY	32
Figure 25 Two Lane Multi-Lane Example for D-PHY	33
Figure 26 Three Lane Multi-Lane Example for D-PHY	34
Figure 27 N-Lane Multi-Lane Example for D-PHY	35
Figure 28 N-Lane Multi-Lane Example for D-PHY Short Packet Transmission.....	36
Figure 29 Two Lane Multi-Lane Example for C-PHY.....	37
Figure 30 Three Lane Multi-Lane Example for C-PHY	38
Figure 31 General N-Lane Multi-Lane Distribution for C-PHY.....	38

10-Nov-2016

Figure 32 One Lane Transmitter and N-Lane Receiver Example for D-PHY	39
Figure 33 M-Lane Transmitter and N-Lane Receiver Example ($M < N$) for D-PHY	39
Figure 34 M-Lane Transmitter and One Lane Receiver Example for D-PHY	40
Figure 35 M-Lane Transmitter and N-Lane Receiver Example ($N < M$) for D-PHY	40
Figure 36 Example of Digital Logic to Align All RxDataHS	41
Figure 37 Low Level Protocol Packet Overview	43
Figure 38 Long Packet Structure for D-PHY Physical Layer Option	44
Figure 39 Long Packet Structure for C-PHY Physical Layer Option	45
Figure 40 Packet Header Lane Distribution for C-PHY Physical Layer Option	46
Figure 41 Minimal Filler Byte Insertion Requirements for Three Lane C-PHY	48
Figure 42 Short Packet Structure for D-PHY Physical Layer Option	49
Figure 43 Short Packet Structure for C-PHY Physical Layer Option	49
Figure 44 Data Identifier Byte	50
Figure 45 Logical Channel Block Diagram (Receiver)	51
Figure 46 Interleaved Video Data Streams Examples	52
Figure 47 26-bit ECC Generation Example	54
Figure 48 64-bit ECC Generation on TX Side	59
Figure 49 26-bit ECC Generation on TX Side	59
Figure 50 64-bit ECC on RX Side Including Error Correction	60
Figure 51 26-bit ECC on RX Side Including Error Correction	61
Figure 52 Checksum Transmission Byte Order	62
Figure 53 Checksum Generation for Long Packet Payload Data	62
Figure 54 Definition of 16-bit CRC Shift Register	63
Figure 55 16-bit CRC Software Implementation Example	63
Figure 56 Packet Spacing	64
Figure 57 Example Interlaced Frame Using LS/SE Short Packet and Line Counting	67
Figure 58 Multiple Packet Example	69
Figure 59 Single Packet Example	69
Figure 60 Line and Frame Blanking Definitions	70
Figure 61 Vertical Sync Example	71
Figure 62 Horizontal Sync Example	71
Figure 63 Interpacket Latency Reduction Using LRTE EPD	72
Figure 64 LRTE Efficient Packet Delimiter Example for CSI-2 over C-PHY (2 Lanes)	74

Figure 65 Example of LRTE EPD for CSI-2 Over D-PHY – Option 1.....	75
Figure 66 Example of LRTE EPD for CSI-2 Over D-PHY – Option 2.....	76
Figure 67 Using EPD and ALPS Together	77
Figure 68 System Diagram Showing Per-Lane Scrambling.....	80
Figure 69 Example of Data Bursts in Two Lanes Using the D-PHY Physical Layer.....	81
Figure 70 Example of Data Bursts in Two Lanes Using the C-PHY Physical Layer.....	82
Figure 71 Generating Tx Sync Type as Seed Index (Single Lane View)	83
Figure 72 Generating Tx Sync Type Using the C-PHY Physical Layer.....	84
Figure 73 PRBS LFSR Serial Implementation Example.....	87
Figure 74 General Frame Format Example	91
Figure 75 Digital Interlaced Video Example	92
Figure 76 Digital Interlaced Video with Accurate Synchronization Timing Information	93
Figure 77 Interleaved Data Transmission using Data Type Value.....	94
Figure 78 Packet Level Interleaved Data Transmission	95
Figure 79 Frame Level Interleaved Data Transmission.....	96
Figure 80 Interleaved Data Transmission using Virtual Channels	97
Figure 81 Byte Packing Pixel Data to C-PHY Symbol Illustration.....	102
Figure 82 Frame Structure with Embedded Data at the Beginning and End of the Frame	104
Figure 83 Legacy YUV420 8-bit Transmission.....	106
Figure 84 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration.....	107
Figure 85 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1	107
Figure 86 Legacy YUV420 8-bit Frame Format	108
Figure 87 YUV420 8-bit Data Transmission Sequence	109
Figure 88 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration	110
Figure 89 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1.....	110
Figure 90 YUV420 Spatial Sampling for MPEG 2 and MPEG 4	111
Figure 91 YUV420 8-bit Frame Format.....	111
Figure 92 YUV420 10-bit Transmission	112
Figure 93 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration	113
Figure 94 YUV420 10-bit Frame Format.....	113
Figure 95 YUV422 8-bit Transmission	114
Figure 96 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration	114
Figure 97 YUV422 Co-sited Spatial Sampling	115

10-Nov-2016

Figure 98 YUV422 8-bit Frame Format.....	115
Figure 99 YUV422 10-bit Transmitted Bytes	116
Figure 100 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration	116
Figure 101 YUV422 10-bit Frame Format.....	117
Figure 102 RGB888 Transmission	119
Figure 103 RGB888 Transmission in CSI-2 Bus Bitwise Illustration.....	119
Figure 104 RGB888 Frame Format.....	120
Figure 105 RGB666 Transmission with 18-bit BGR Words	121
Figure 106 RGB666 Transmission on CSI-2 Bus Bitwise Illustration.....	121
Figure 107 RGB666 Frame Format.....	122
Figure 108 RGB565 Transmission with 16-bit BGR Words	123
Figure 109 RGB565 Transmission on CSI-2 Bus Bitwise Illustration.....	123
Figure 110 RGB565 Frame Format.....	124
Figure 111 RGB555 Transmission on CSI-2 Bus Bitwise Illustration	125
Figure 112 RGB444 Transmission on CSI-2 Bus Bitwise Illustration.....	126
Figure 113 RAW6 Transmission	128
Figure 114 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration.....	128
Figure 115 RAW6 Frame Format	128
Figure 116 RAW7 Transmission	129
Figure 117 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration.....	129
Figure 118 RAW7 Frame Format	129
Figure 119 RAW8 Transmission	130
Figure 120 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration	130
Figure 121 RAW8 Frame Format.....	130
Figure 122 RAW10 Transmission	131
Figure 123 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration	131
Figure 124 RAW10 Frame Format.....	132
Figure 125 RAW12 Transmission	133
Figure 126 RAW12 Transmission on CSI-2 Bus Bitwise Illustration.....	133
Figure 127 RAW12 Frame Format.....	133
Figure 128 RAW14 Transmission	134
Figure 129 RAW14 Transmission on CSI-2 Bus Bitwise Illustration.....	134
Figure 130 RAW14 Frame Format.....	135

Figure 131 RAW16 Transmission	136
Figure 132 RAW16 Transmission on CSI-2 Bus Bitwise Illustration	136
Figure 133 RAW16 Frame Format	136
Figure 134 RAW20 Transmission	137
Figure 135 RAW20 Transmission on CSI-2 Bus Bitwise Illustration	137
Figure 136 RAW20 Frame Format	138
Figure 137 User Defined 8-bit Data (128 Byte Packet)	139
Figure 138 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration	139
Figure 139 Transmission of User Defined 8-bit Data	139
Figure 140 General/Arbitrary Data Reception	141
Figure 141 RGB888 Data Format Reception	142
Figure 142 RGB666 Data Format Reception	142
Figure 143 RGB565 Data Format Reception	143
Figure 144 RGB555 Data Format Reception	143
Figure 145 RGB444 Data Format Reception	144
Figure 146 YUV422 8-bit Data Format Reception	144
Figure 147 YUV422 10-bit Data Format Reception	145
Figure 148 YUV420 8-bit Legacy Data Format Reception	146
Figure 149 YUV420 8-bit Data Format Reception	147
Figure 150 YUV420 10-bit Data Format Reception	148
Figure 151 RAW6 Data Format Reception	149
Figure 152 RAW7 Data Format Reception	149
Figure 153 RAW8 Data Format Reception	150
Figure 154 RAW10 Data Format Reception	150
Figure 155 RAW12 Data Format Reception	151
Figure 156 RAW 14 Data Format Reception	151
Figure 157 RAW16 Data Format Reception	152
Figure 158 RAW20 Data Format Reception	153
Figure 159 JPEG8 Data Flow in the Encoder	155
Figure 160 JPEG8 Data Flow in the Decoder	155
Figure 161 EXIF Compatible Baseline JPEG DCT Format	156
Figure 162 Status Information Field in the End of Baseline JPEG Frame	158
Figure 163 Example of TN Image Embedding Inside the Compressed JPEG Data Block	159

10-Nov-2016

Figure 164 JPEG8 Data Format Reception	160
Figure 165 Implementation Example Block Diagram and Coverage.....	161
Figure 166 CSI-2 Transmitter Block Diagram	162
Figure 167 CSI-2 Receiver Block Diagram	163
Figure 168 D-PHY Level Block Diagram.....	164
Figure 169 CSI-2 Clock Lane Transmitter	165
Figure 170 CSI-2 Clock Lane Receiver	166
Figure 171 CSI-2 Data Lane Transmitter	167
Figure 172 CSI-2 Data Lane Receiver	169
Figure 173 SLM Synchronization	176
Figure 174 Data Compression System Block Diagram.....	178
Figure 175 Pixel Order of the Original Image.....	179
Figure 176 Example Pixel Order of the Original Image	179
Figure 177 Data Type Interleaving: Concurrent JPEG and YUV Image Data	223
Figure 178 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data	224
Figure 179 Example JPEG and YUV Interleaving Use Cases	225

Tables

Table 1 CCI I/O Characteristics	23
Table 2 CCI Timing Specification	24
Table 3 Data Type Classes	53
Table 4 ECC Syndrome Association Matrix	56
Table 5 ECC Parity Generation Rules	57
Table 6 Synchronization Short Packet Data Type Codes	65
Table 7 Generic Short Packet Data Type Codes	68
Table 8 LRTE Transmitter Registers for CSI-2 Over C-PHY	78
Table 9 LRTE Transmitter Registers for CSI-2 Over D-PHY	79
Table 10 Symbol Sequence Values Per Sync Type	82
Table 11 Fields That Are Not Scrambled	85
Table 12 D-PHY Scrambler PRBS Initial Seed Values for Lanes 1 Through 8	85
Table 13 C-PHY Scrambler PRBS Initial Seed Values for Lanes 1 Through 8	86
Table 14 Example of the PRBS Bit-at-a-Time Shift Sequence	88
Table 15 Example PRBS LFSR Byte Sequence for D-PHY Physical Layer	88
Table 16 Example PRBS LFSR Byte Sequence for C-PHY Physical Layer	89
Table 17 Primary and Secondary Data Formats Definitions	101
Table 18 Generic 8-bit Long Packet Data Types	103
Table 19 YUV Image Data Types	105
Table 20 Legacy YUV420 8-bit Packet Data Size Constraints	106
Table 21 YUV420 8-bit Packet Data Size Constraints	109
Table 22 YUV420 10-bit Packet Data Size Constraints	112
Table 23 YUV422 8-bit Packet Data Size Constraints	114
Table 24 YUV422 10-bit Packet Data Size Constraints	116
Table 25 RGB Image Data Types	118
Table 26 RGB888 Packet Data Size Constraints	119
Table 27 RGB666 Packet Data Size Constraints	121
Table 28 RGB565 Packet Data Size Constraints	123
Table 29 RAW Image Data Types	127
Table 30 RAW6 Packet Data Size Constraints	128
Table 31 RAW7 Packet Data Size Constraints	129

Table 32 RAW8 Packet Data Size Constraints	130
Table 33 RAW10 Packet Data Size Constraints	131
Table 34 RAW12 Packet Data Size Constraints	133
Table 35 RAW14 Packet Data Size Constraints	134
Table 36 RAW16 Packet Data Size Constraints	136
Table 37 RAW20 Packet Data Size Constraints	137
Table 38 User Defined 8-bit Data Types	140
Table 39 Status Data Padding	157
Table 40 JPEG8 Additional Marker Codes Listing	160
Table 41 Initial Seed Values for Lanes 9 through 32	227

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.
YYYY-MM-DD	v2.0	Target version for next release.

10-Nov-2016

Development History

Date	Version	Description
2005-11-29	v1.00	Initial Board-approved release.
2010-11-09	v1.01.00	Board-approved release.
2012-07-18	v1.1	Updated D-PHY citation reference to v1.1. Applied latest paragraph and character styles. Corrected typographical and formatting issues. Added Release History and Development History tables.
2014-01-16	v1.2 r05	Updated D-PHY citation reference to v1.2 and reflected possibility to implement CSI-2 configurations with a number of Lanes higher than 4.
2014-01-23	v1.2 r06	Addressed 18 CR requests, scrubbed document of highlighting, re-linked broken figure links, made several editorial changes.
2014-01-29	v1.3 r03	Final version merging v1.3 r02 with v1.2 r06. x
2014-02-15	v1.3 r06	Addressed BZ items 4714, 4710, 4700, 4702, 4703, 4704, 4711, 4716, 4721, 4723, 4729. Merged r03, r04, and r05. Migrated to the latest template. Migrated from Word .doc format to .docx format. Made all embedded figures linked figures.
2014-02-16	v1.3 r07	Addressed BZ items 4717, 4725, 4728.
2014-02-22	v1.3 r08	Addressed BZ items 4762, 4763, 4765, 4775.
2014-05-29	v1.3 r09	Addressed BZ items 4861. Made minor editorial changes to the document.
2015-05-06	v2.0 r01	Applied all edits between v1.3 r09 and v1.3 Board Adopted. Updated version to v2.0. Addressed BZ items 5158, 5277, 4727, and 4899. Applied 'Cross Reference' character style to all document citation cross-references.
2015-09-09	v2.0 r02	Addressed BZ items 4718, 4724, 4726, 4895, 5459, and 5480. Added page breaks to keep more related material together.
2015-10-07	v2.0 r03	Addressed BZ items 5661 and 5665. Additional pagination and line number clean-up.
2016-06-02	v2.0 r04	Addressed BZ items 5495 and 5516. Updated date of referenced MIPI D-PHY v2.0 Specification. Applied Cross-Reference style to cross references.
2016-10-05	v2.0 r05	Addressed BZ items 5126, 5516, 5854, 5917, and 5971. Made hyphens in 'C-PHY' and 'D-PHY' non-breaking.
2016-10-19	v2.0 r06	Addressed BZ items 6008, 6030, and 6039.
2016-11-09	v2.0 r07	Addressed BZ item 6065.
2016-11-10	v2.0 r08	Addressed BZ item 6090.

This page intentionally left blank.

10-Nov-2016

1 Introduction

1.1 Scope

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

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

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

1.2 Purpose

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

21 CSI-2 provides the mobile industry a standard, robust, scalable, low-power, high-speed, cost-effective
22 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

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.

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.

10-Nov-2016

2.3 Abbreviations

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

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

2.4 Acronyms

63 BER Bit Error Rate

64 CCI Camera Control Interface

65 CIL Control and Interface Logic

66 CRC Cyclic Redundancy Check

67 CSI Camera Serial Interface

68 CSPS Chroma Shifted Pixel Sampling

69 DDR Dual Data Rate

70 DI Data Identifier

71 DT Data Type

72 ECC Error Correction Code

73 EoT End of Transmission

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

75 EXIF Exchangeable Image File Format

76 FE Frame End

77 FS Frame Start

78 HS High Speed; identifier for operation mode

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

81 HS-RX High-Speed Receiver

82 HS-TX High-Speed Transmitter

83 I2C Inter-Integrated Circuit

84 ILR Interpacket Latency Reduction

85 JFIF JPEG File Interchange Format

86 JPEG Joint Photographic Expert Group

87 LE Line End

88 LFSR Linear Feedback Shift Register

89 LLP Low Level Protocol

90 LS Line Start

91 LSB Least Significant Bit

92 LSS Least Significant Symbol

93 LP Low-Power; identifier for operation mode

94	LP-RX	Low-Power Receiver (Large-Swing Single Ended)
95	LP-TX	Low-Power Transmitter (Large-Swing Single Ended)
96	LRTE	Latency Reduction Transport Efficiency
97	MSB	Most Significant Bit
98	MSS	Most Significant Symbol
99	PDQ	Packet Delimiter Quick (PHY generated and consumed signaling used in LRTE)
100	PF	Packet Footer
101	PH	Packet Header
102	PI	Packet Identifier
103	PT	Packet Type
104	PHY	Physical Layer
105	PPI	PHY Protocol Interface
106	PRBS	Pseudo-Random Binary Sequence
107	RGB	Color representation (Red, Green, Blue)
108	RX	Receiver
109	SCL	Serial Clock (for CCI)
110	SDA	Serial Data (for CCI)
111	SLM	Sleep Mode
112	SoT	Start of Transmission
113	TX	Transmitter
114	ULPS	Ultra Low Power State
115	VGA	Video Graphics Array
116	YUV	Color representation (Y for luminance, U & V for chrominance)

3 References

- 117 [NXP01] UM10204, *I2C-bus specification and user manual*, Revision 117 03, NXP B.V., 19 June
118 2007.
- 119 [MIPI01] *MIPI Alliance Specification for D-PHY*, version 2.1, MIPI Alliance, Inc., In press.
- 120 [MIPI02] *MIPI Alliance Specification for C-PHY*, version 1.2, MIPI Alliance, Inc., In press.

This page intentionally left blank.

10-Nov-2016

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].

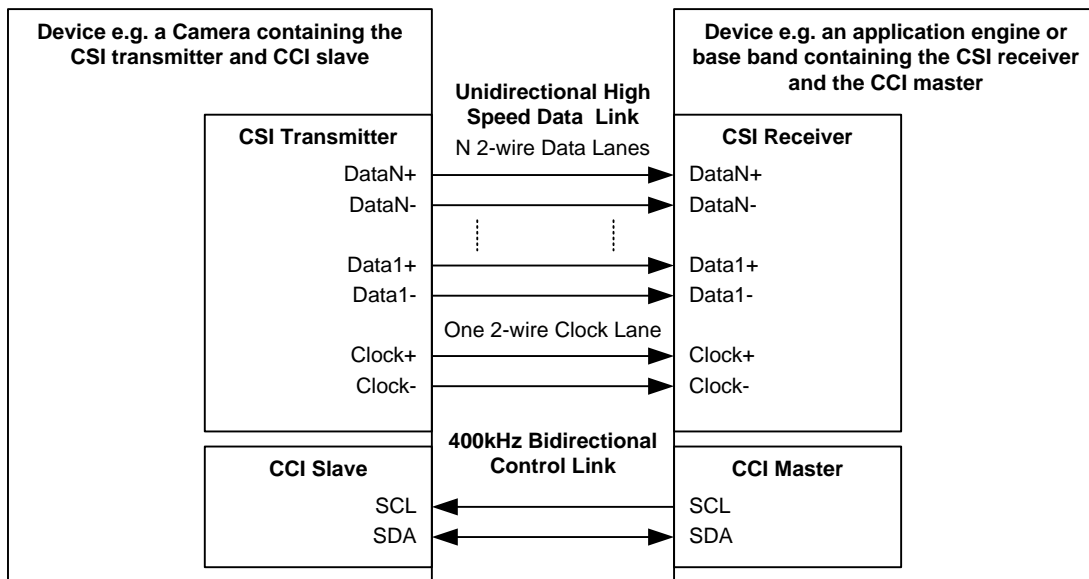
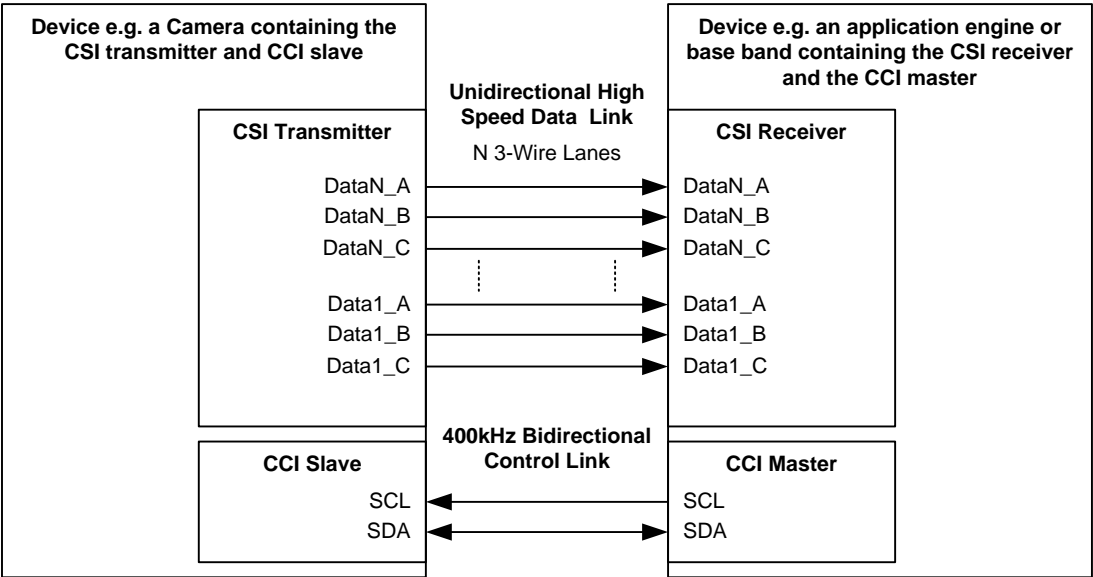


Figure 1 CSI-2 and CCI Transmitter and Receiver Interface for D-PHY



136

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

10-Nov-2016

5 CSI-2 Layer Definitions

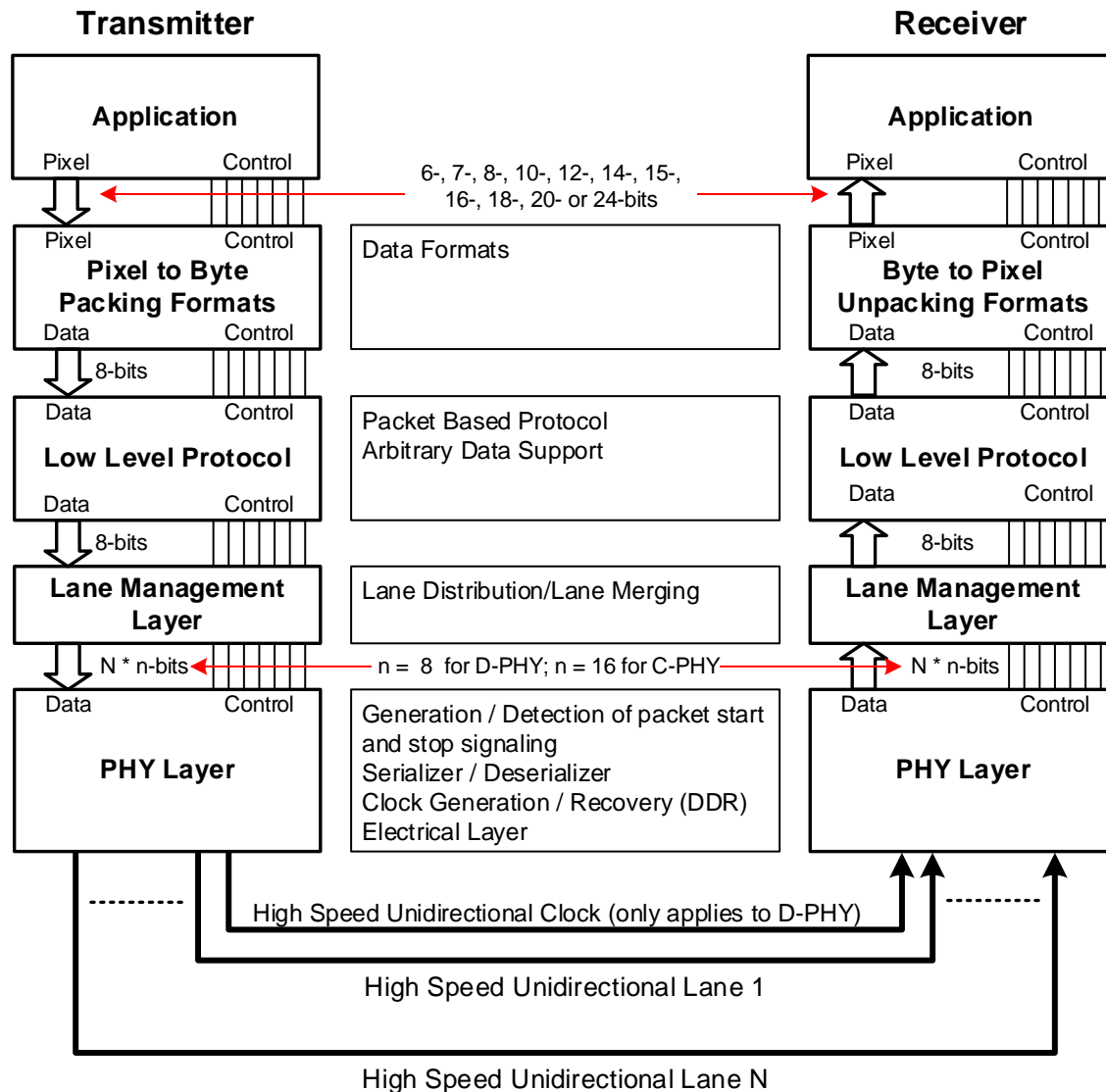


Figure 3 CSI-2 Layer Definitions

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.

10-Nov-2016

6 Camera Control Interface (CCI)

CCI is a two-wire, bi-directional, half duplex, serial interface for controlling the transmitter. CCI is compatible with the fast mode variant of the I2C interface. CCI shall support 400kHz operation and 7-bit Slave Addressing.

A CSI-2 receiver shall be configured as a master and a CSI-2 transmitter shall be configured as a slave on the CCI bus. CCI is capable of handling multiple slaves on the bus. However, multi-master mode is not supported by CCI. Any I2C commands that are not described in this section shall be ignored and shall not cause unintended device operation. Note that the terms master and slave, when referring to CCI, should not be confused with similar terminology used for D-PHY's operation; they are not related.

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

CCI is a subset of the I2C protocol, including the minimum combination of obligatory features for I2C slave devices specified in the I2C specification. Therefore, transmitters complying with the CCI specification can also be connected to the system I2C bus. However, care must be taken so that I2C masters do not try to utilize those I2C features that are not supported by CCI masters and CCI slaves

Each CCI transmitter may have additional features to support I2C, but that is dependent on implementation. 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. As such, it is the responsibility of the CSI-2 implementer to define a set of control messages and corresponding frame timing and I2C latency requirements, if any, that must be met by the CCI master when sending such control messages to the CCI slave.

The CCI defines an additional data protocol layer on top of I2C. The data protocol is presented in the following sections.

205 **6.1 Data Transfer Protocol**

206 The data transfer protocol is according to I2C standard. The START, REPEATED START and STOP
207 conditions as well as data transfer protocol are specified in *The I²C Specification [NXP01]*.

208 **6.1.1 Message Type**

209 A basic CCI message consists of START condition, slave address with read/write bit, acknowledge from
210 slave, sub address (index) for pointing at a register inside the slave device, acknowledge signal from slave,
211 in write operation data byte from master, acknowledge/negative acknowledge from slave and STOP
212 condition. In read operation data byte comes from slave and acknowledge/negative acknowledge from
213 master. This is illustrated in *Figure 4*.

214 The slave address in the CCI is 7-bit.

215 The CCI supports 8-bit index with 8-bit data or 16-bit index with 8-bit data. The slave device in question
216 defines what message type is used.

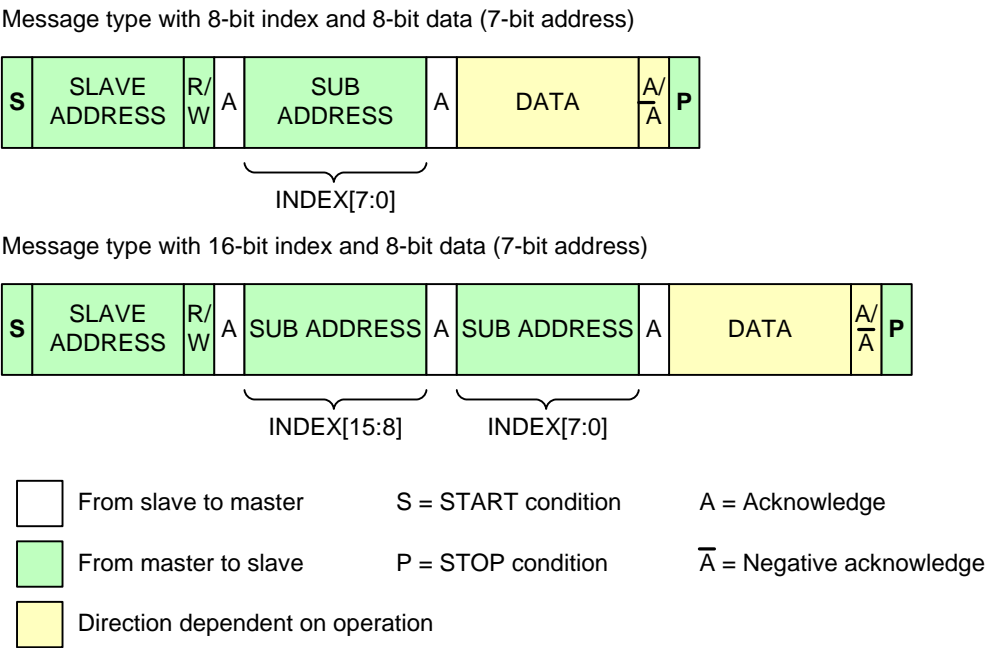


Figure 4 CCI Message Types

10-Nov-2016

6.1.2 Read/Write Operations

The CCI compatible device shall be able to support four different read operations and two different write operations; single read from random location, sequential read from random location, single read from current location, sequential read from current location, single write to random location and sequential write starting from random location. The read/write operations are presented in the following sections.

The index in the slave device has to be auto incremented after each read/write operation. This is also explained in the following sections.

6.1.2.1 Single Read from Random Location

In single read from random location the master does a dummy write operation to desired index, issues a repeated start condition and then addresses the slave again with read operation. After acknowledging its slave address, the slave starts to output data onto SDA line. This is illustrated in **Figure 5**. The master terminates the read operation by setting a negative acknowledge and stop condition.

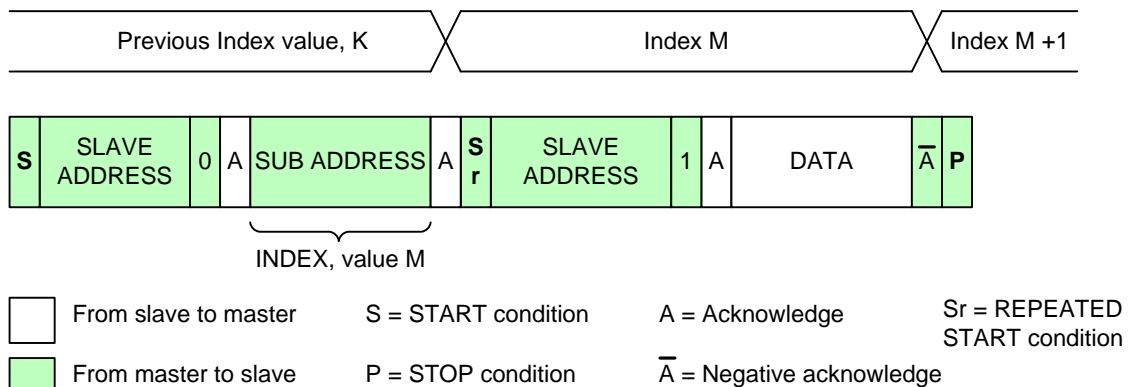


Figure 5 CCI Single Read from Random Location

6.1.2.2 Single Read from the Current Location

It is also possible to read from last used index by addressing the slave with read operation. The slave responds by setting the data from last used index to SDA line. This is illustrated in **Figure 6**. The master terminates the read operation by setting a negative acknowledge and stop condition.

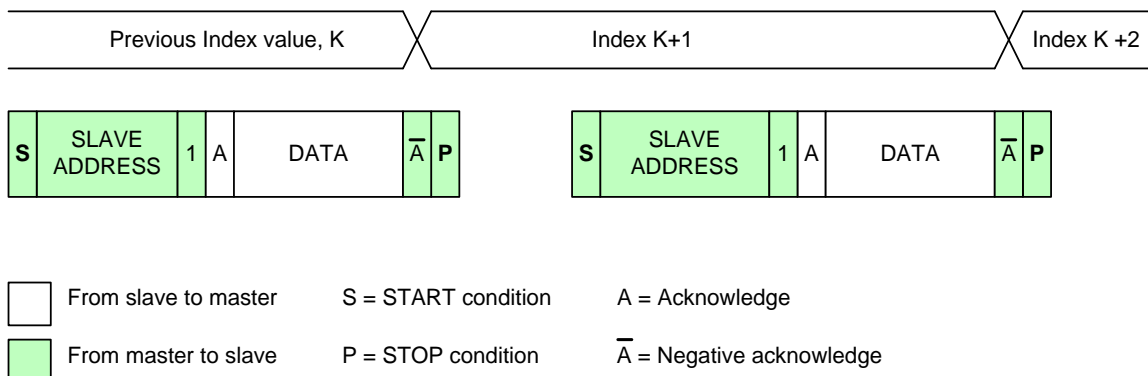


Figure 6 CCI Single Read from Current Location

6.1.2.3 Sequential Read Starting from a Random Location

The sequential read starting from a random location is illustrated in **Figure 7**. 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 received data it acts as a signal to the slave that the read operation continues from the next index. When the master has read the last data byte it issues a negative acknowledge and stop condition.

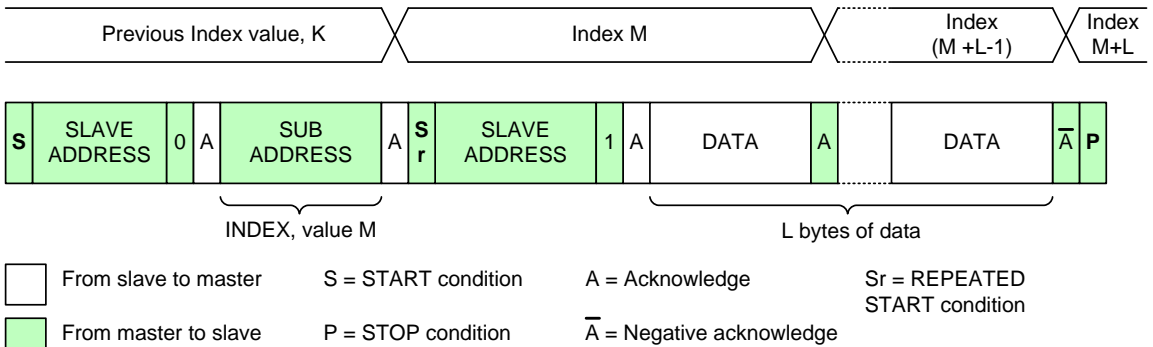


Figure 7 CCI Sequential Read Starting from a Random Location

6.1.2.4 Sequential Read Starting from the Current Location

A sequential read starting from the current location is similar to a sequential read from a random location. The only exception is there is no dummy write operation. The command sequence is illustrated in **Figure 8**. The master terminates the read operation by issuing a negative acknowledge and stop condition.

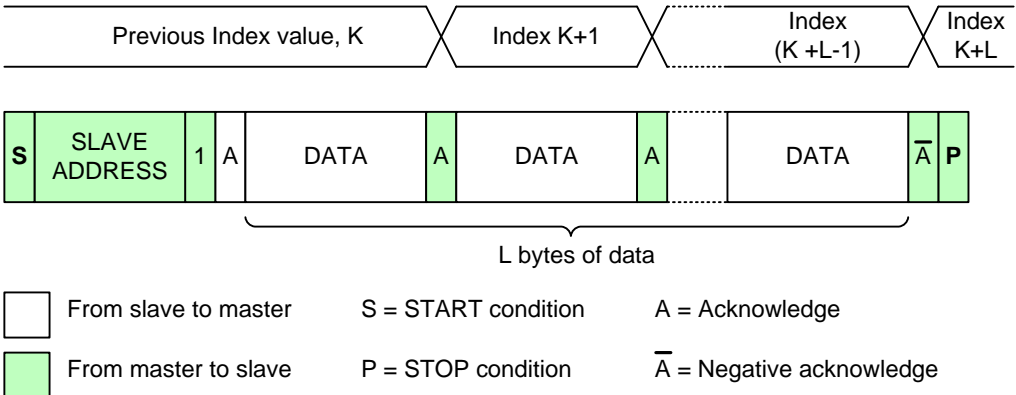
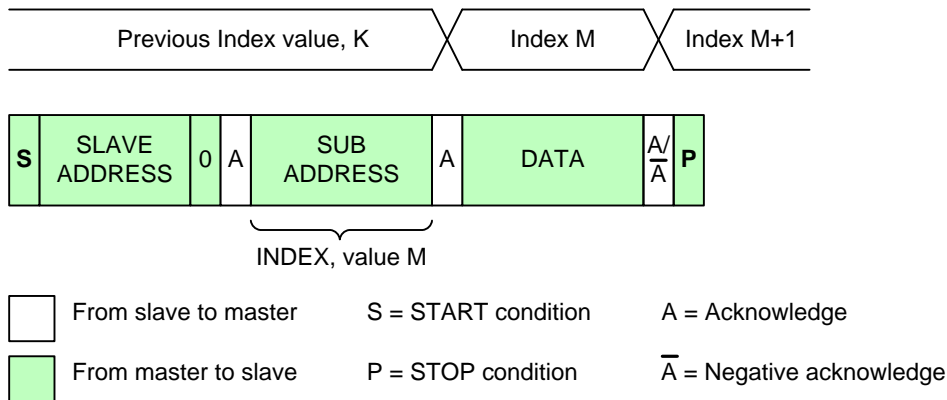


Figure 8 CCI Sequential Read Starting from the Current Location

248 **6.1.2.5 Single Write to a Random Location**

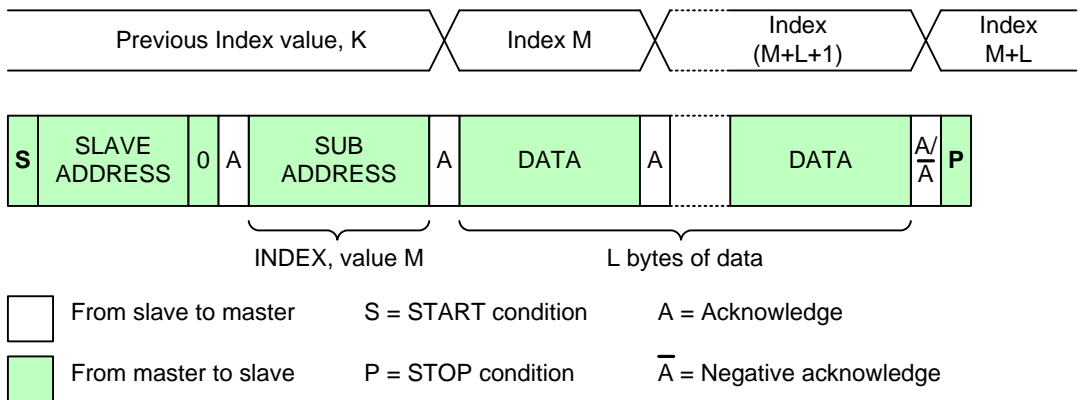
249 A write operation to a random location is illustrated in **Figure 9**. The master issues a write operation to the
250 slave then issues the index and data after the slave has acknowledged the write operation. The write
251 operation is terminated with a stop condition from the master.



252 **Figure 9 CCI Single Write to a Random Location**

253 **6.1.2.6 Sequential Write**

254 The sequential write operation is illustrated in **Figure 10**. The slave auto-increments the index after each
255 data byte is received. The sequential write operation is terminated with a stop condition from the master.



256 **Figure 10 CCI Sequential Write Starting from a Random Location**

6.2 CCI Slave Addresses

For camera modules having only raw Bayer output the 7-bit slave address should be 011011Xb, where X = 0 or 1. For all other camera modules the 7-bit slave address should be 011110Xb.

6.3 CCI Multi-Byte Registers

6.3.1 Overview

Peripherals contain a wide range of different register widths for various control and setup purposes. The CSI-2 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, it could then be accessed using a four-byte Sequential Read from the Current Location protocol.

The motivation for a general multi-byte protocol rather than fixing the registers at 16-bits width is flexibility. The protocol described in the following paragraphs 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 at the highest address.

The address of the first byte of a multi-byte register may, or may not be, aligned to the size of the register; i.e., a multiple of the number of register bytes. The register alignment is an implementation choice between processing optimized and 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 that rules for the valid locations for the MS bytes and LS bytes of registers are followed.

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 first byte is accessed first; its second byte is accessed second, etc.

When a multi-byte register is accessed, the following re-timing rules must 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 has been read

Section 6.3.3 describes example behavior for the re-timing of multi-byte register accesses.

Without re-timing, data may be corrupted as illustrated in **Figure 11** and **Figure 12**.

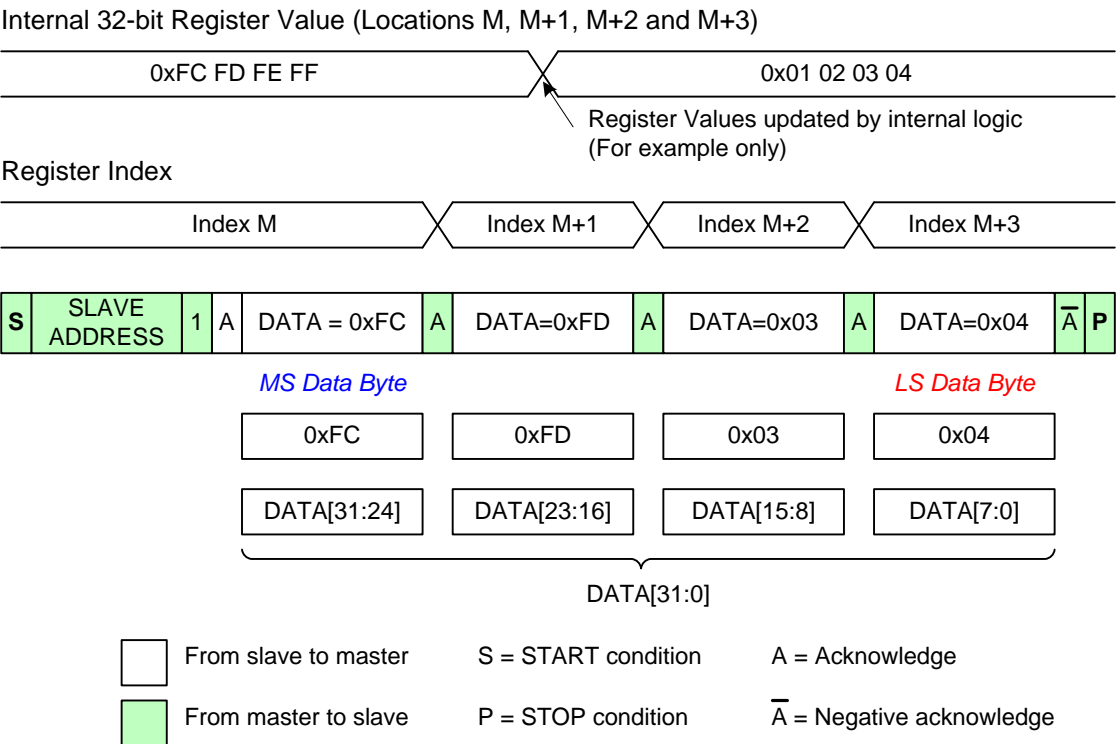


Figure 11 Corruption of a 32-bit Wide Register during a Read Message

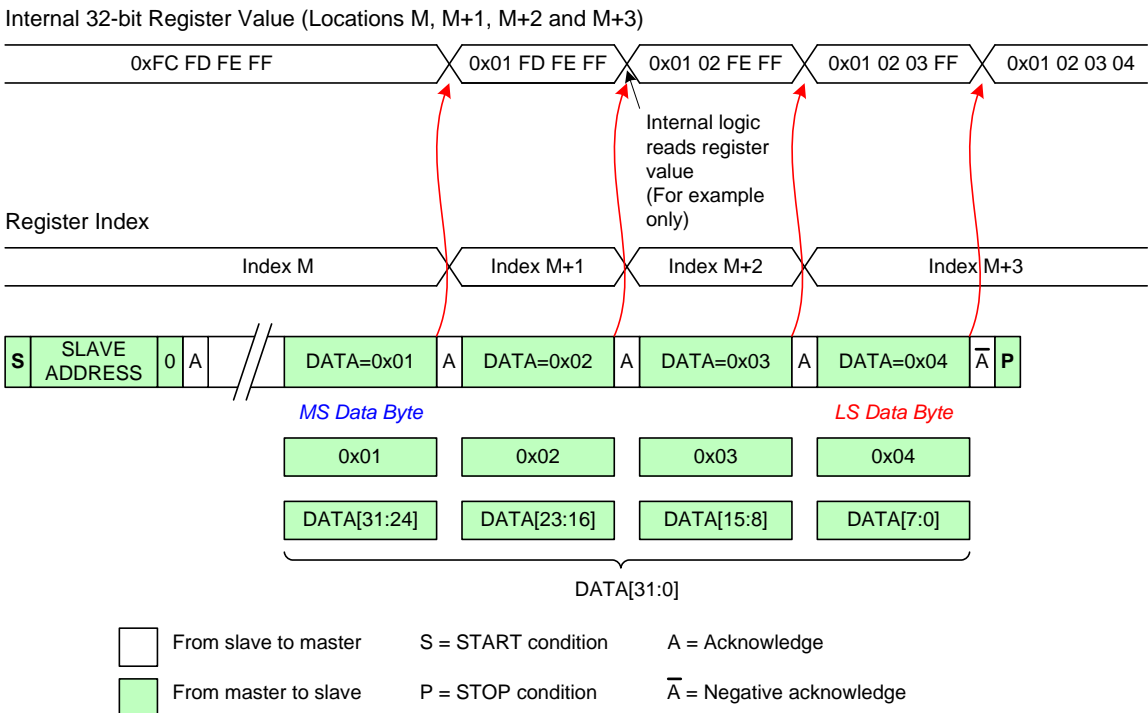


Figure 12 Corruption of a 32-bit Wide Register during a Write Message

6.3.2 The Transmission Byte Order for Multi-byte Register Values

This is a normative section.

The first byte of a CCI message is always the MS byte of a multi-byte register and the last byte is always the LS byte.

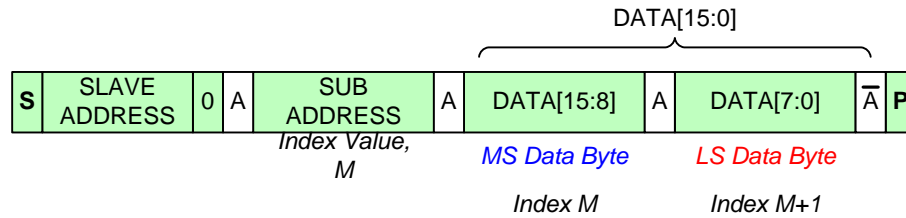


Figure 13 Example 16-bit Register Write

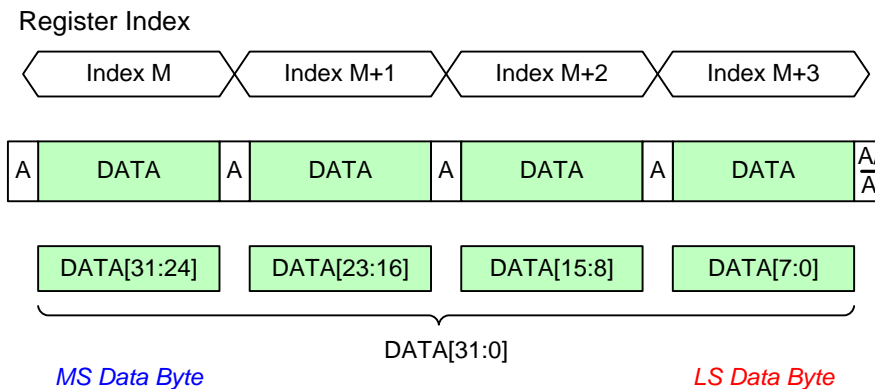


Figure 14 Example 32-bit Register Write (address not shown)

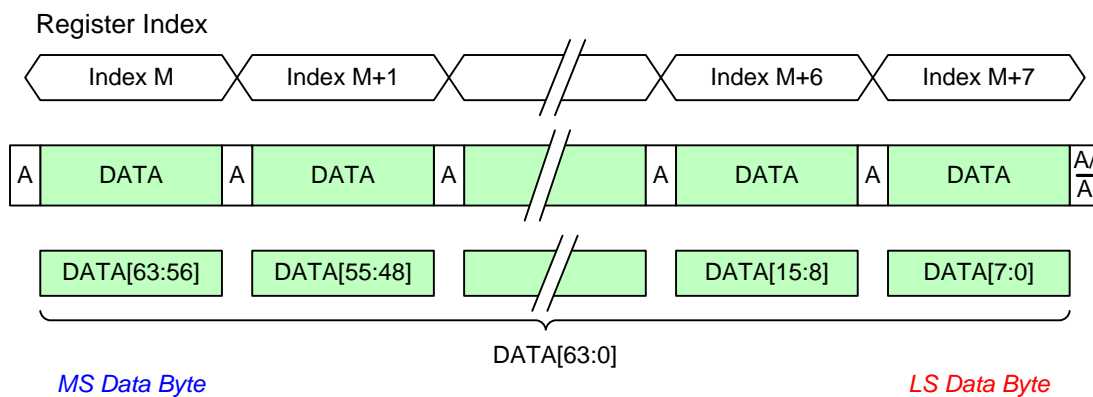


Figure 15 Example 64-bit Register Write (address not shown)

6.3.3 Multi-Byte Register Protocol

This is an informative section.

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

10-Nov-2016

6.3.3.1 Reading Multi-byte Registers

To ensure that the value read from a multi-byte register is consistent (i.e. all bytes are temporally coherent), the device internally transfers the contents of the register into a temporary buffer when the MS byte of the register is read. The contents of the temporary buffer are then output as a sequence of bytes on the SDA line. **Figure 16** and **Figure 17** illustrate multi-byte register read operations.

The temporary buffer is always updated unless the read operation is incremental within the same multi-byte register.

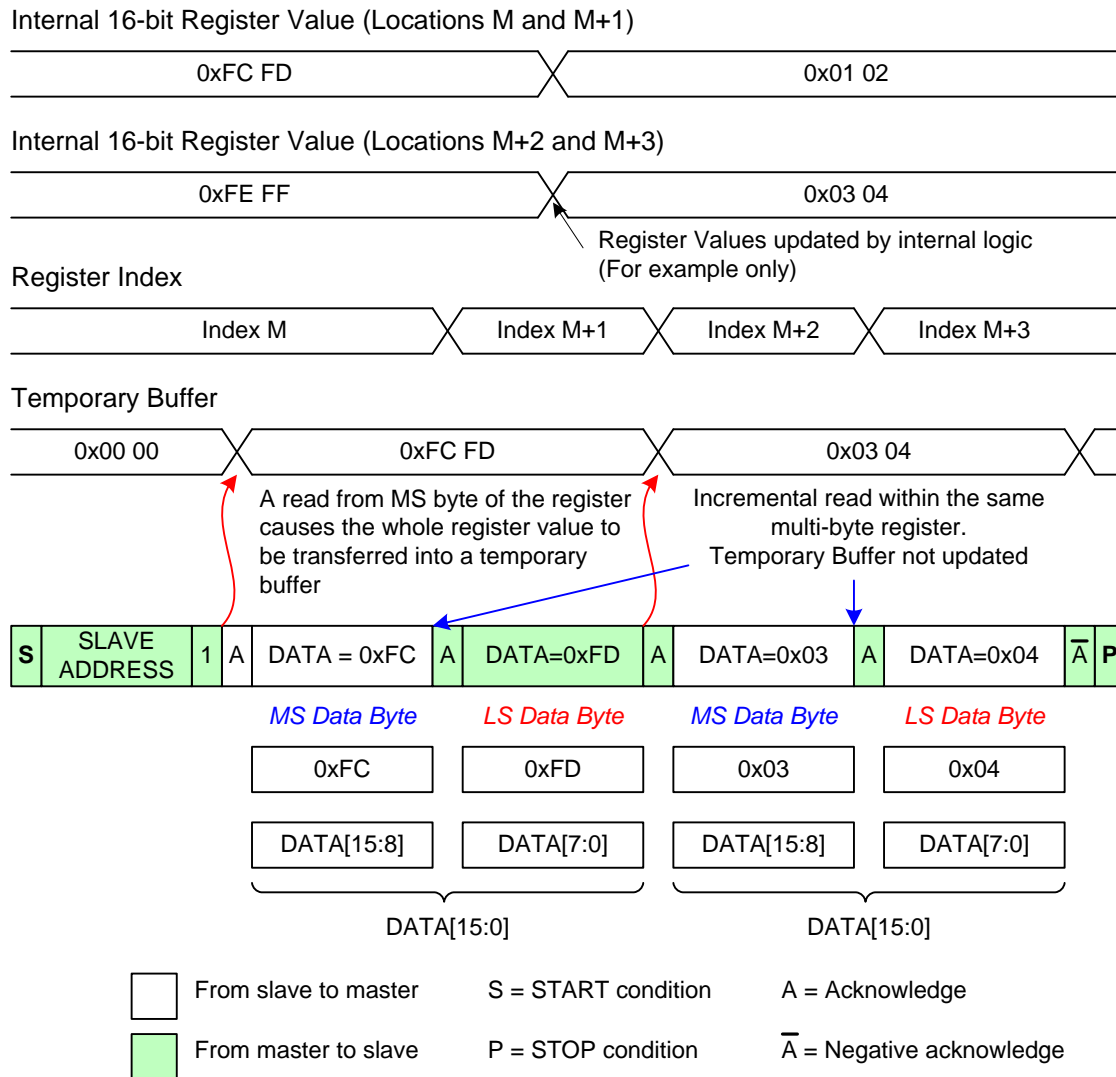


Figure 16 Example 16-bit Register Read

In this definition there is no distinction made between whether the register is accessed incrementally via separate, single byte read messages with no intervening data writes or 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 are as follows:

- The MS byte of a register is accessed
- The index has crossed a multi-byte register boundary
- Successive single byte reads from the same index location
- The index value for the byte about to be read is the same or less than the previous index

Unless the contents of a multi-byte register are accessed in an incremental manner the values read back are not guaranteed to be consistent.

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

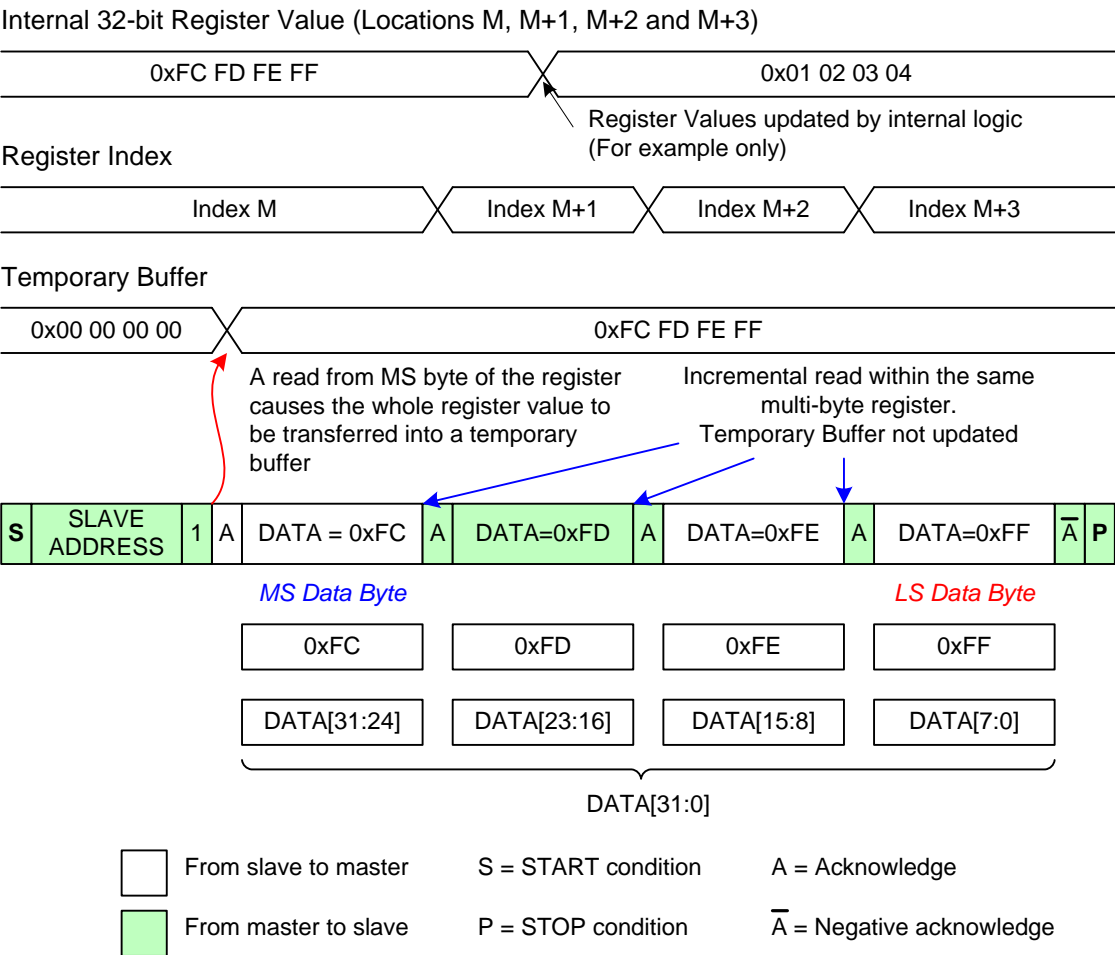


Figure 17 Example 32-bit Register Read

10-Nov-2016

6.3.3.2 Writing Multi-byte Registers

To ensure that the value written is consistent, the bytes of data of 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 18 and **Figure 19** 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.

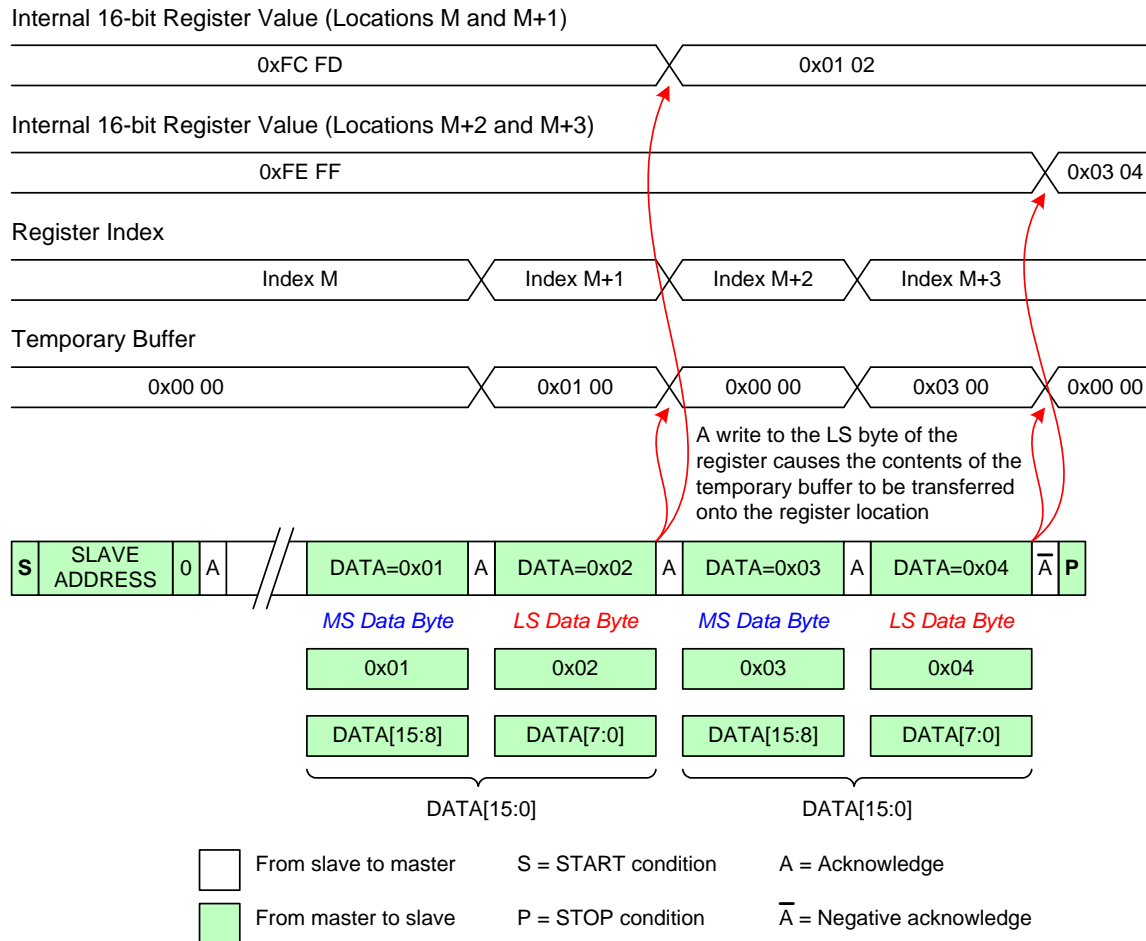


Figure 18 Example 16-bit Register Write

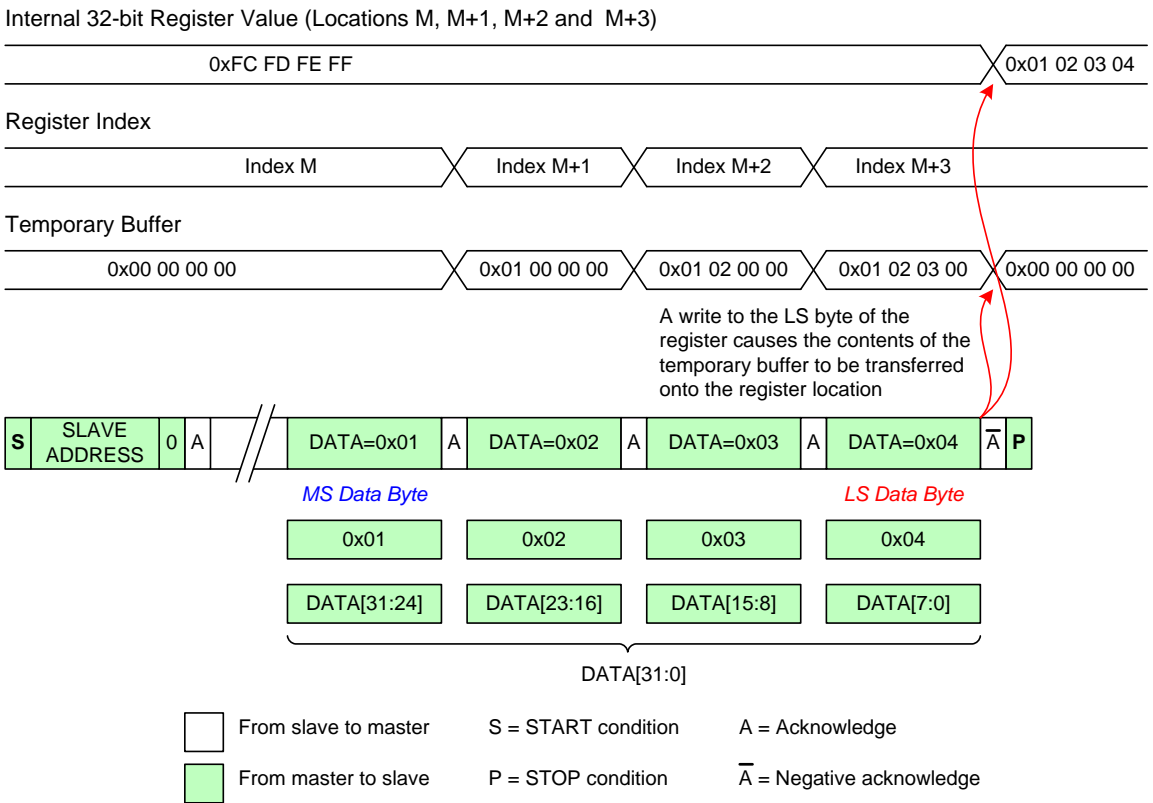


Figure 19 Example 32-bit Register Write

10-Nov-2016

6.4 Electrical Specifications and Timing for I/O Stages

The electrical specification and timing for I/O stages conform to I²C Standard- and Fast-mode devices. Information presented in **Table 1** is from [NXP01].

Table 1 CCI I/O Characteristics

Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
LOW level input voltage	V _{IL}	-0.5	0.3V _{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 _{DD} > 2V V _{DD} < 2V	V _{HYS}	N/A N/A	N/A N/A	0.05V _{DD} 0.1V _{DD}	- -	V
LOW level output voltage (open drain) at 3mA sink current V _{DD} > 2V V _{DD} < 2V	V _{OL1} V _{OL3}	0 N/A	0.4 N/A	0 0	0.4 0.2V _{DD}	V
HIGH level output voltage	V _{OH}	N/A	N/A	0.8V _{DD}		V
Output fall time from V _{IHmin} to V _{ILmax} with bus capacitance from 10 pF to 400 pF	t _{OF}	-	250	20+0.1C _B Note 2	250	ns
Pulse width of spikes which shall be suppressed by the input filter	t _{SP}	N/A	N/A	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	10	-10 Note 3	10 Note 3	μA
Input/Output capacitance (SDA)	C _{I/O}	-	8	-	8	pF
Input capacitance (SCL)	C _I	-	6	-	6	pF

Note:

1. Maximum V_{IH} = V_{DDmax} + 0.5V
2. C_B = capacitance of one bus line in pF
3. I/O pins of Fast-mode devices shall not obstruct the SDA and SCL line if V_{DD} is switched off

344

Table 2 CCI Timing Specification

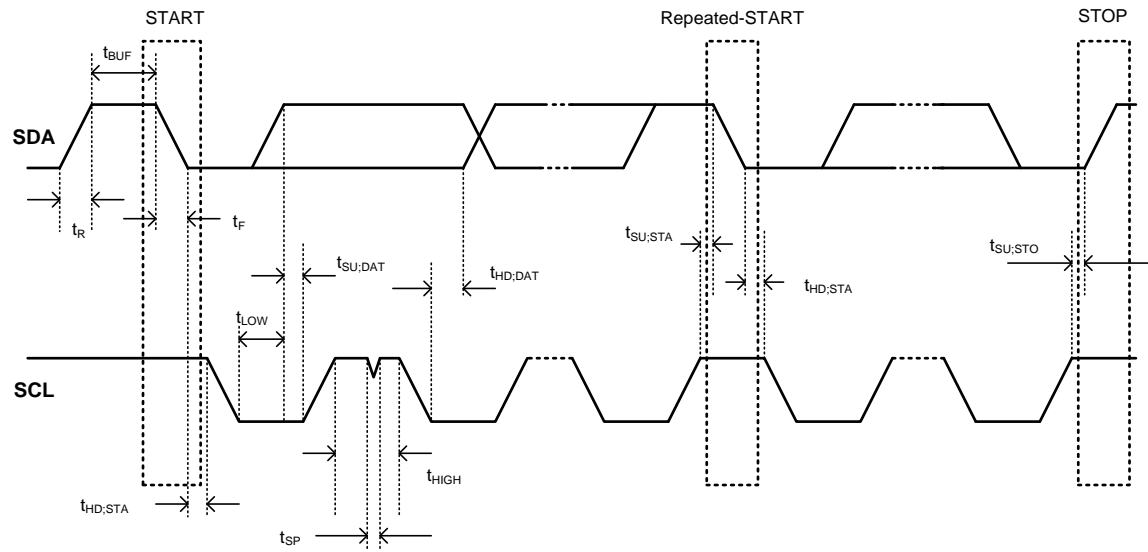
Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	0.4	-	0.6	-	μs
LOW period of the SCL clock	t_{LOW}	4.7	-	1.3	-	μs
HIGH period of the SCL clock	t_{HIGH}	4.0	-	0.6	-	μs
Setup time for a repeated START condition	$t_{SU;STA}$	4.7	-	0.6	-	μs
Data hold time	$t_{HD;DAT}$	0 Note 2	3.45 Note 3	0 Note 2	0.9 Note 3	μs
Data set-up time	$t_{SU;DAT}$	250	-	100 Note 4	-	ns
Rise time of both SDA and SCL signals	t_R	-	1000	$20+0.1C_B$ Note 5	300	ns
Fall time of both SDA and SCL signals	t_F	-	300	$20+0.1C_B$ Note 5	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	-	0.6	-	μs
Bus free time between a STOP and START condition	t_{BUF}	4.7	-	1.3	-	μs
Capacitive load for each bus line	C_B	-	400	-	400	pF
Noise margin at the LOW level for each connected device (including hysteresis)	V_{nL}	$0.1V_{DD}$	-	$0.1V_{DD}$	-	V
Noise margin at the HIGH level for each connected device (including hysteresis)	V_{nH}	$0.2V_{DD}$	-	$0.2V_{DD}$	-	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 $t_{HD;DAT}$ has only to be met if the device does not the LOW period (t_{LOW}) of the SCL signal
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 $t_{MAX} + t_{SU;DAT} = 1000 + 250 = 1250$ ns (according to the Standard-mode I2C bus specification) before the SCL line is released.
5. C_B = total capacitance of one bus line in pF.

10-Nov-2016

345 The CCI timing is illustrated in *Figure 20*.



346

Figure 20 CCI Timing

This page intentionally left blank

10-Nov-2016

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 receiver 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 Physical Layer provides 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. LVLP can also help achieve longer reach for CSI-2 imaging interface channels before re-drivers and re-timers become necessary.

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, 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 can also help achieve longer reach for CSI-2 imaging interface channels before re-drivers and re-timers become necessary.

10-Nov-2016

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 21** and **Figure 22** 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 23** and **Figure 24** 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.

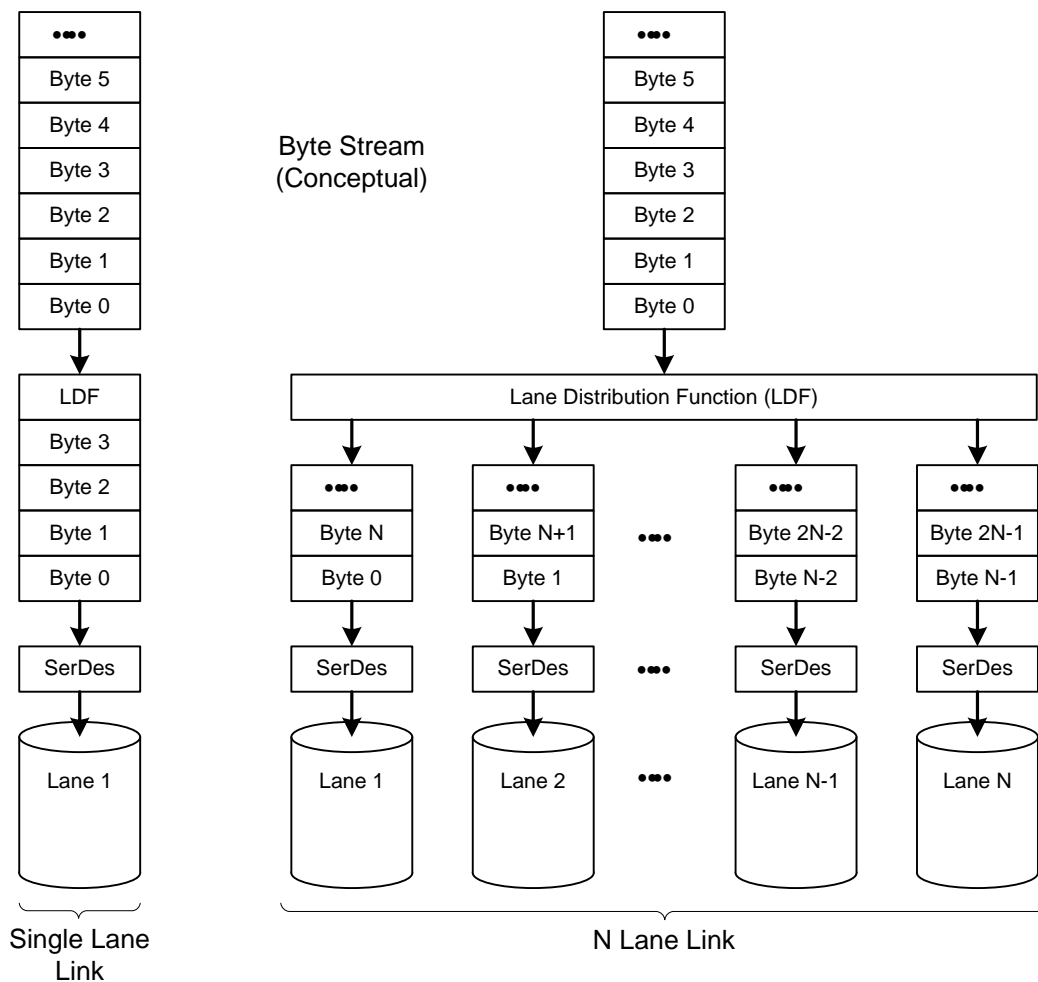


Figure 21 Conceptual Overview of the Lane Distributor Function for D-PHY

416

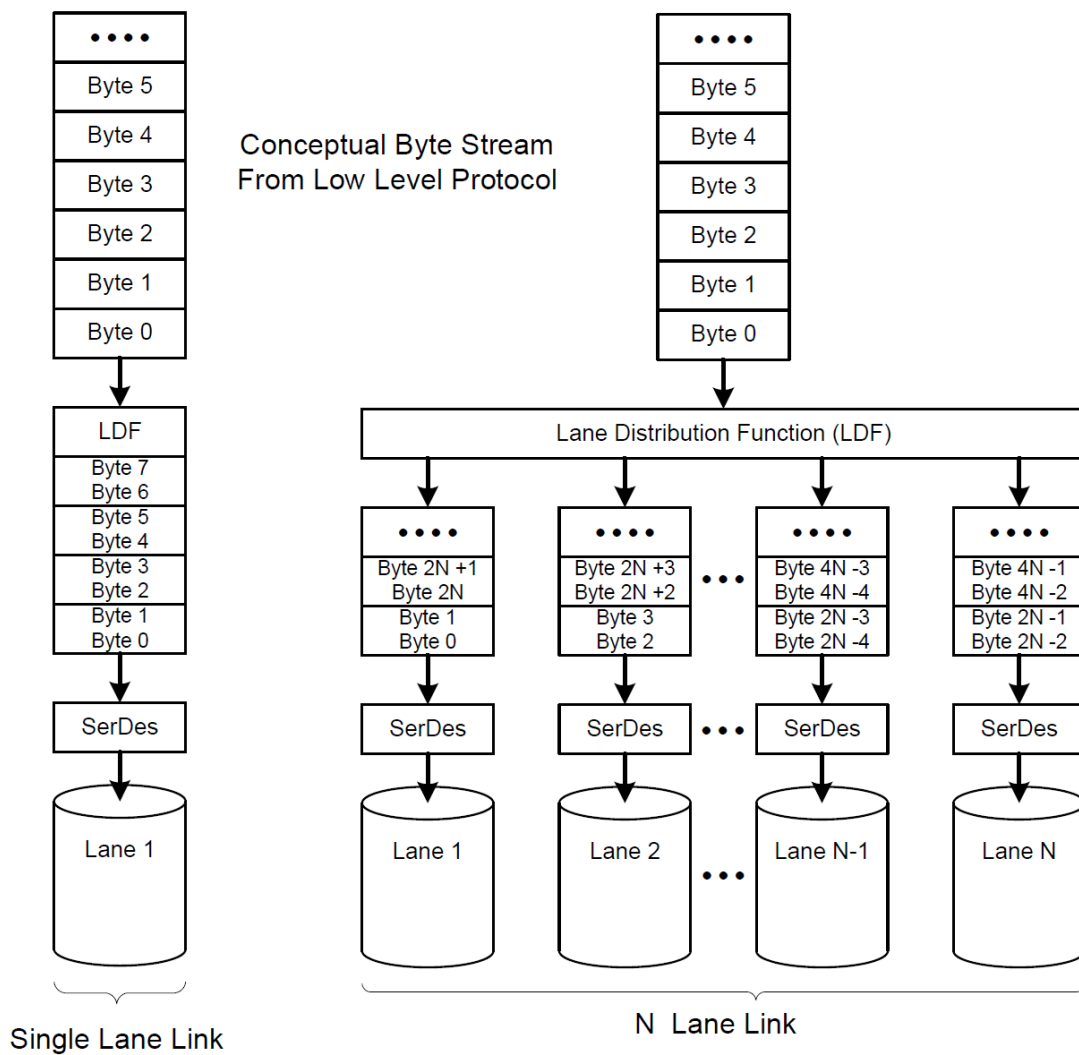


Figure 22 Conceptual Overview of the Lane Distributor Function for C-PHY

10-Nov-2016

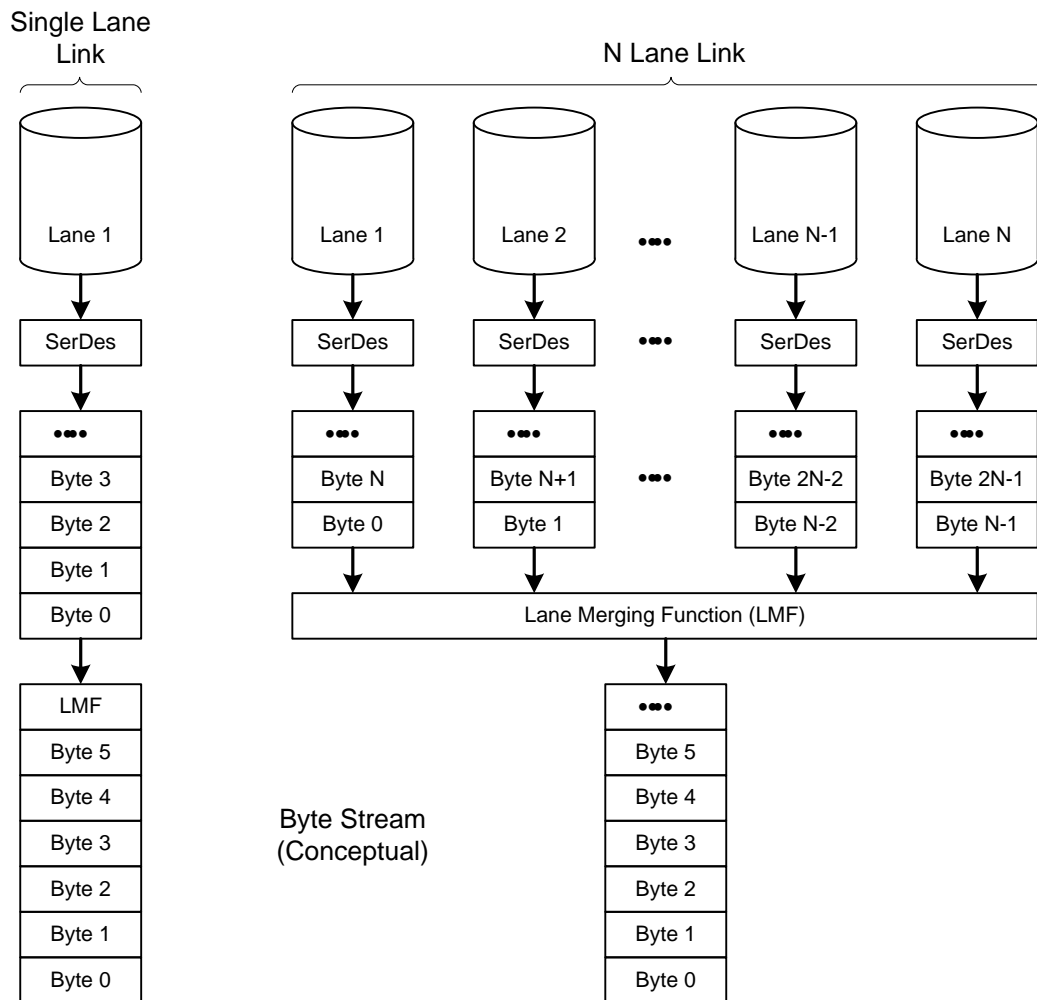


Figure 23 Conceptual Overview of the Lane Merging Function for D-PHY

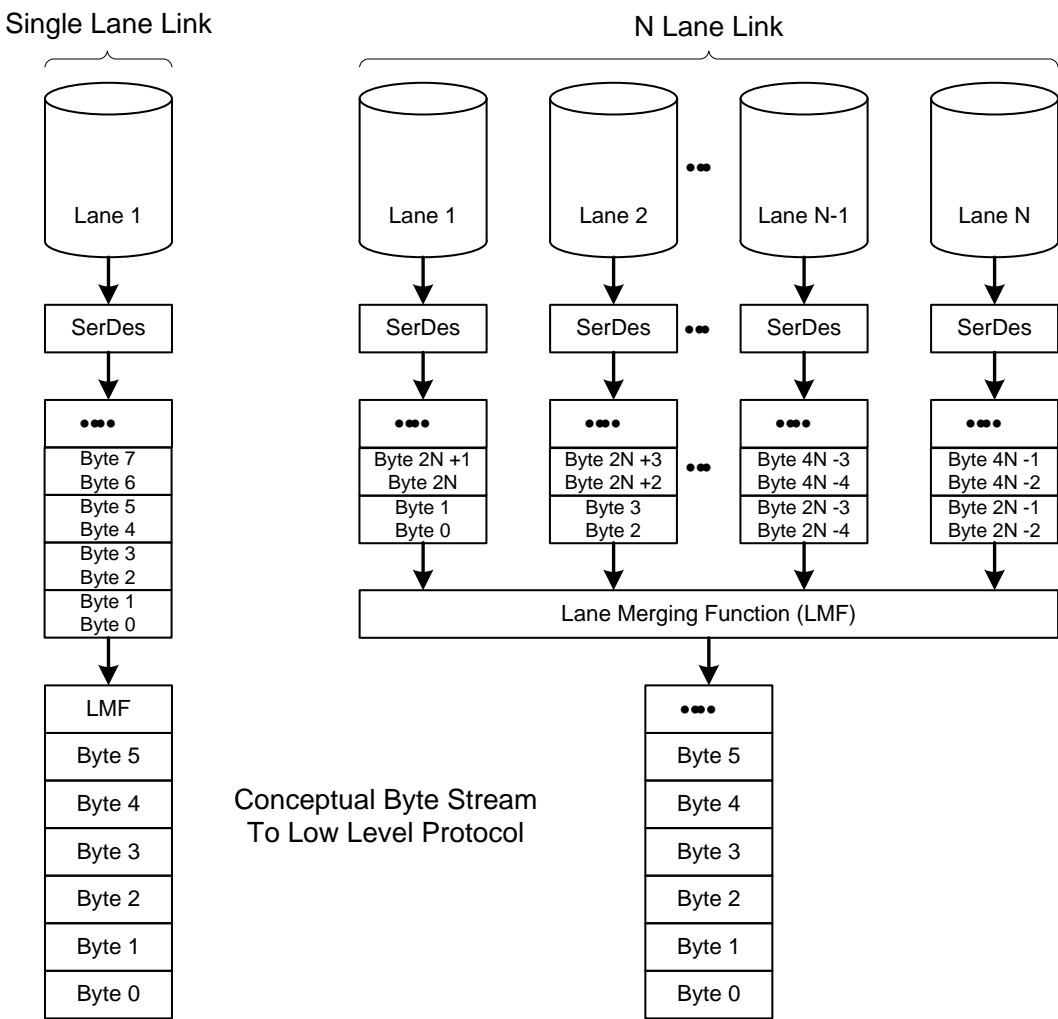


Figure 24 Conceptual Overview of the Lane Merging Function for C-PHY

The Lane distributor takes a transmission of arbitrary byte length, buffers up $N \cdot b$ 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 $N \cdot b$ 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.

10-Nov-2016

8.1 Lane Distribution for the D-PHY Physical Layer Option

Examples are shown in *Figure 25*, *Figure 26*, *Figure 27*, and *Figure 28*:

- 2-Lane system (*Figure 25*): 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 26*): 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 27*): 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 28*) 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.

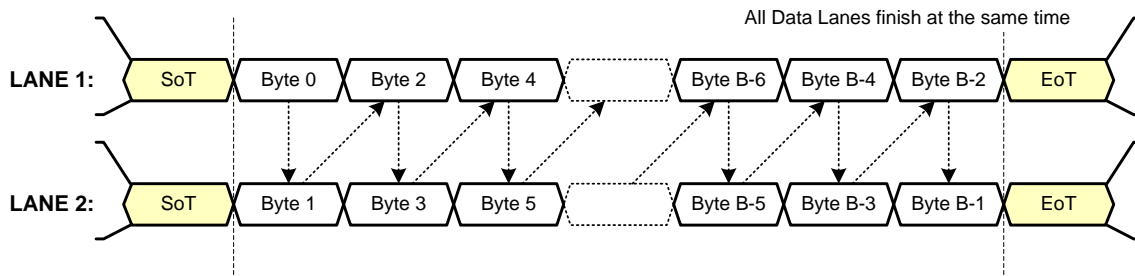
At the end of the transmission, there may be “extra” bytes since the total byte count may not be an integer multiple of the number of Lanes, N. One or more Lanes may send their last bytes before the others. The Lane distributor, as it buffers up the final set of less-than-N bytes in parallel for sending 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.

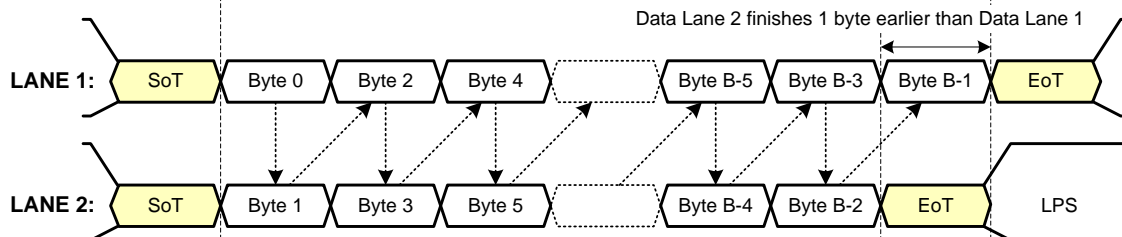
Although multiple Lanes all start simultaneously with parallel “start packet” codes, they may complete the transaction at different times, sending “end packet” codes one cycle (byte) apart.

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:



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



KEY:

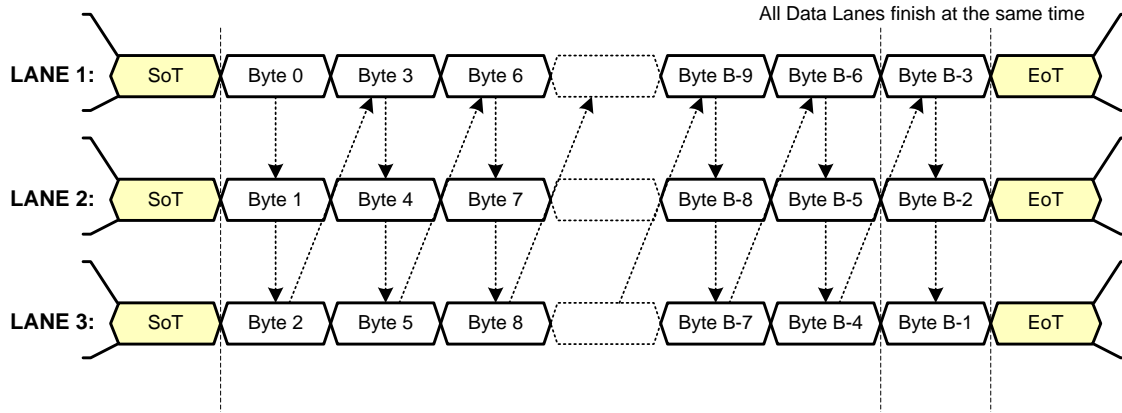
LPS – Low Power State

SoT – Start of Transmission

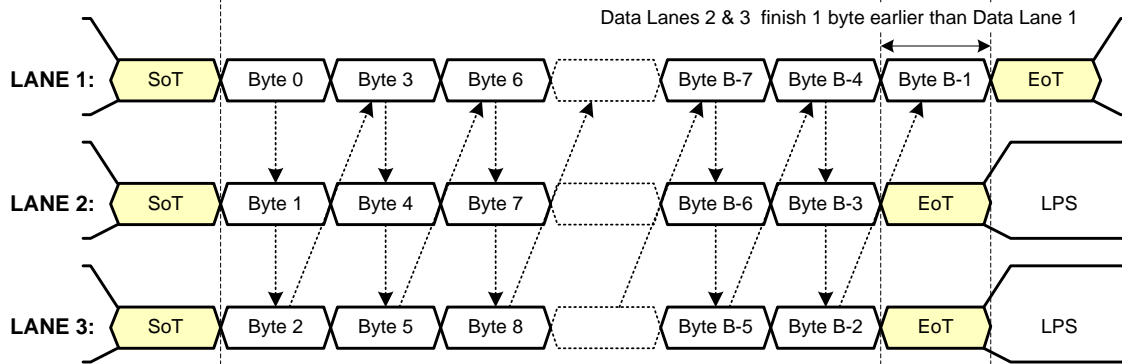
EoT – End of Transmission

Figure 25 Two Lane Multi-Lane Example for D-PHY

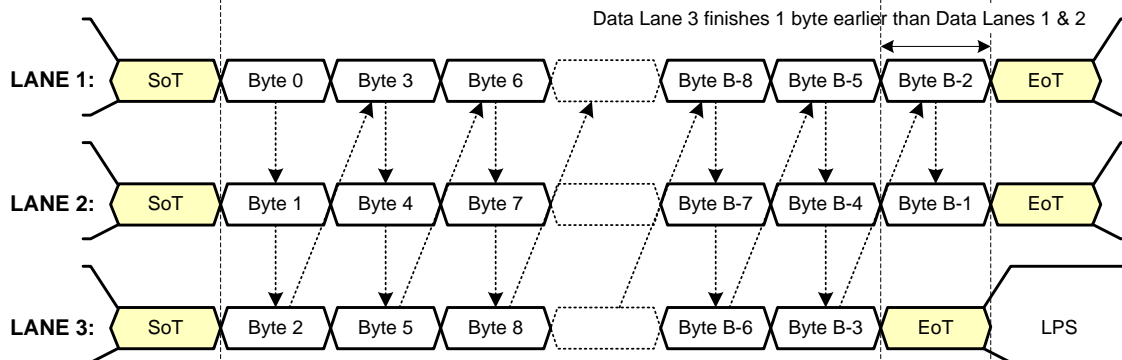
Number of Bytes, B, transmitted is an integer multiple of the number of lanes:



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):



KEY:

LPS – Low Power State

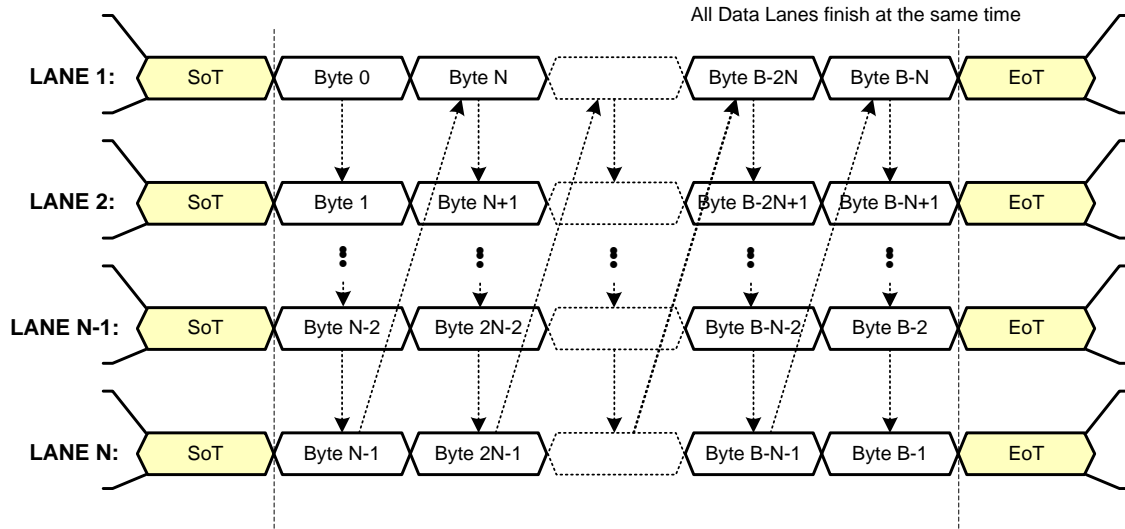
SoT – Start of Transmission

EoT – End of Transmission

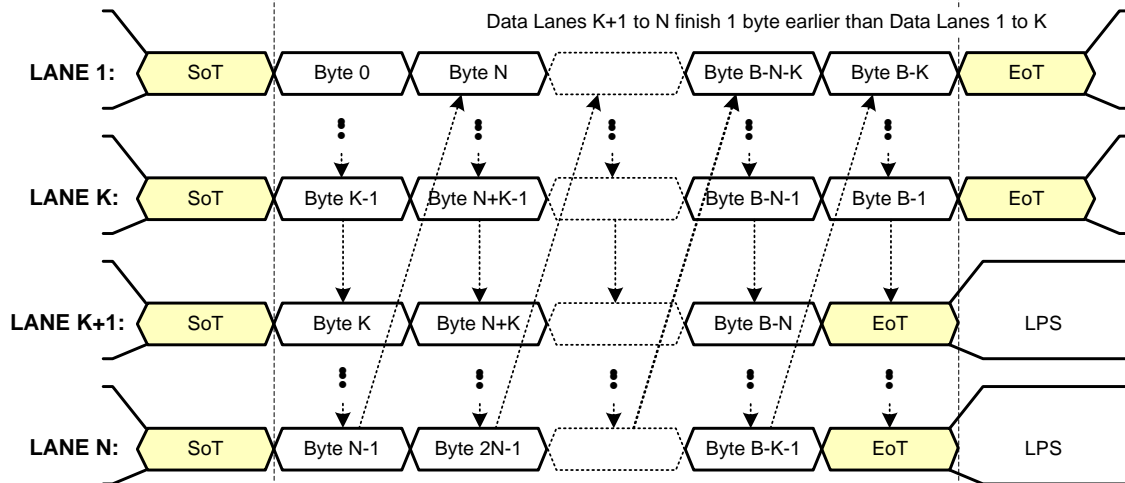
Figure 26 Three Lane Multi-Lane Example for D-PHY

10-Nov-2016

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



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



KEY:

LPS – Low Power State

SoT – Start of Transmission

EoT – End of Transmission

Figure 27 N-Lane Multi-Lane Example for D-PHY

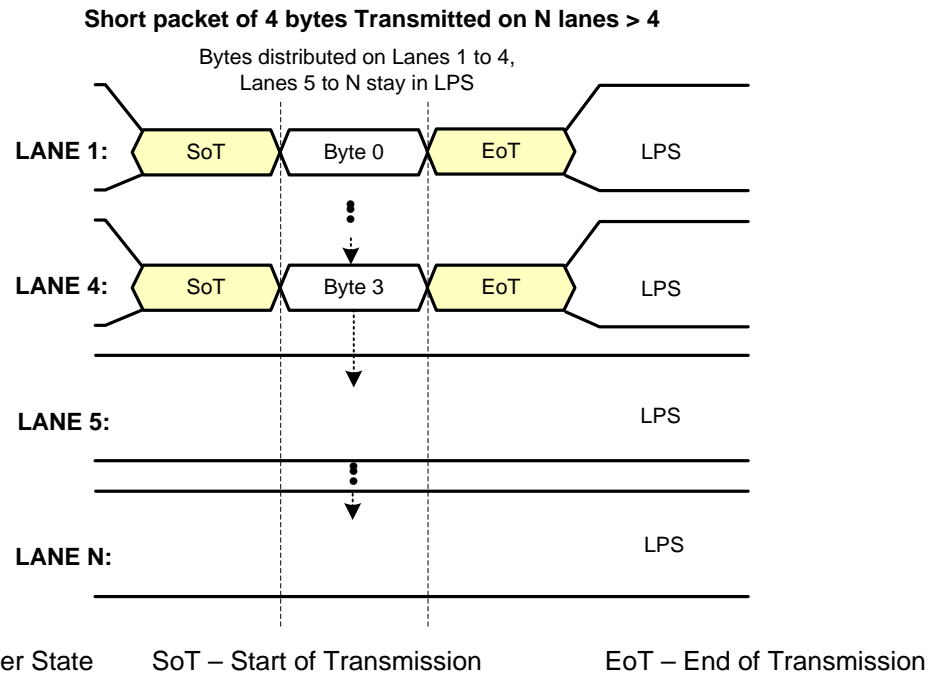


Figure 28 N-Lane Multi-Lane Example for D-PHY Short Packet Transmission

10-Nov-2016

8.2 Lane Distribution for the C-PHY Physical Layer Option

Examples are shown in *Figure 29* and *Figure 30*:

- 2-Lane system (*Figure 29*): 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 30*): 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 31 illustrates normative behavior for an N-Lane system where $N \geq 1$: bytes 1 and 0 of the packet are sent as a 16-bit word to the Lane 1 C-PHY module, bytes 3 and 2 are sent to Lane 2, bytes $2N-1$ and $2N-2$ are sent to Lane N, bytes $2N+1$ and $2N$ are sent to Lane 1, and so on. The last two bytes B-1 and B-2 are sent to Lane N, where B is the total number of bytes in the packet.

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

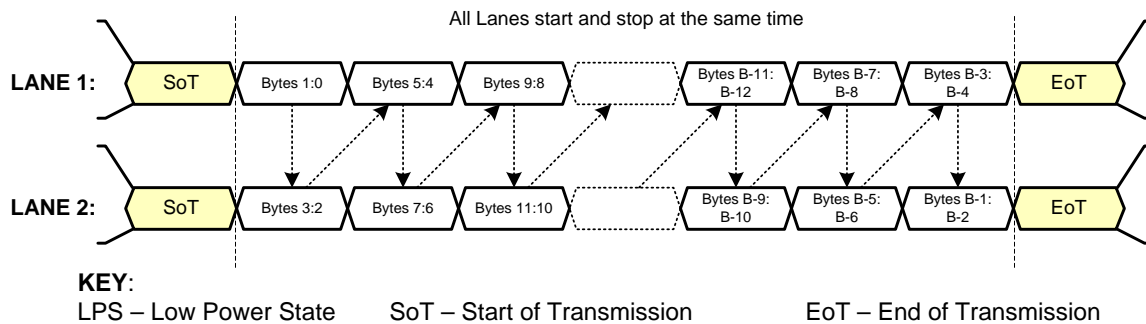


Figure 29 Two Lane Multi-Lane Example for C-PHY

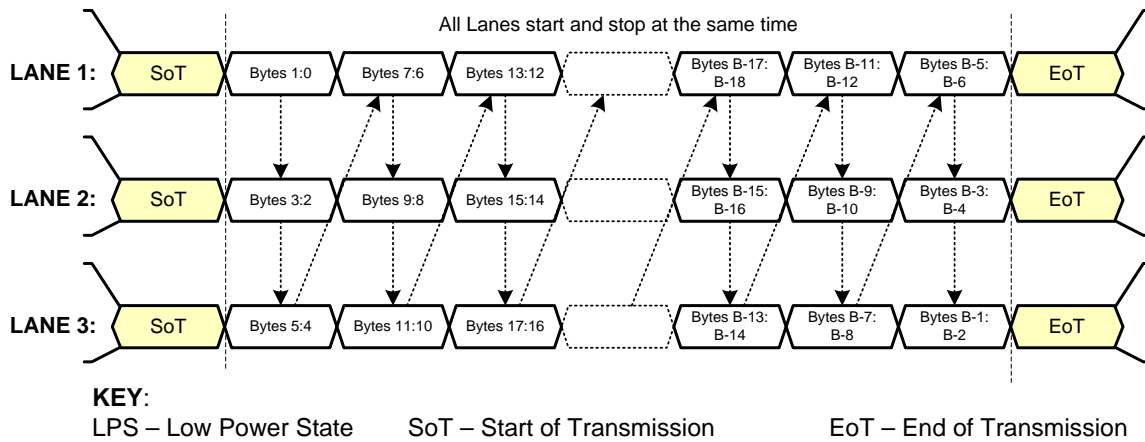


Figure 30 Three Lane Multi-Lane Example for C-PHY

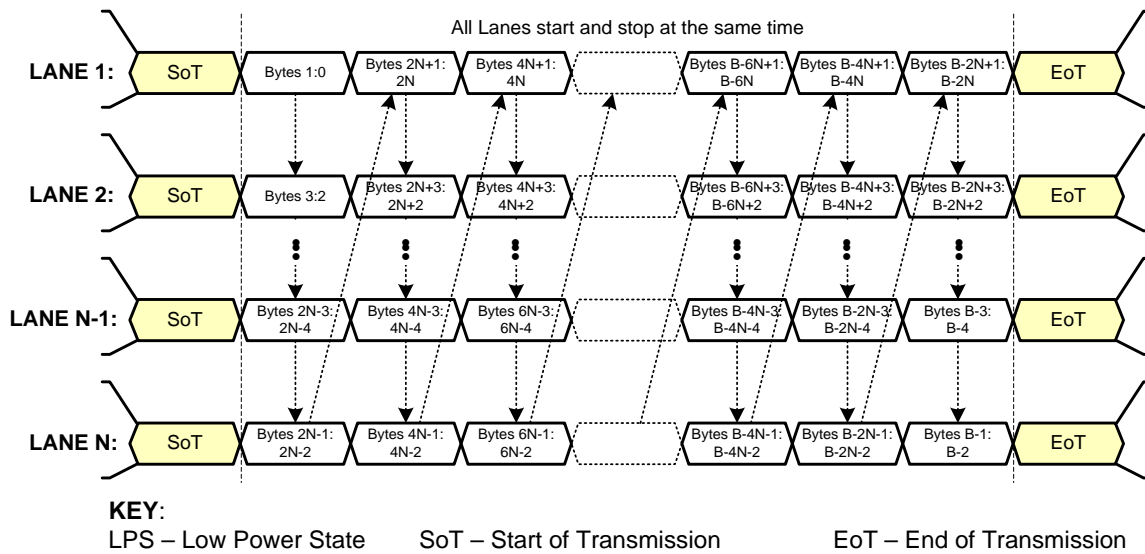


Figure 31 General N-Lane Multi-Lane Distribution for C-PHY

10-Nov-2016

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 $M \leq N$ 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 $M \leq N$ then there is no loss of performance – the receiver has sufficient data Lanes to match the transmitter (**Figure 32** and **Figure 33**).
- 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 34** and **Figure 35**).
- 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.

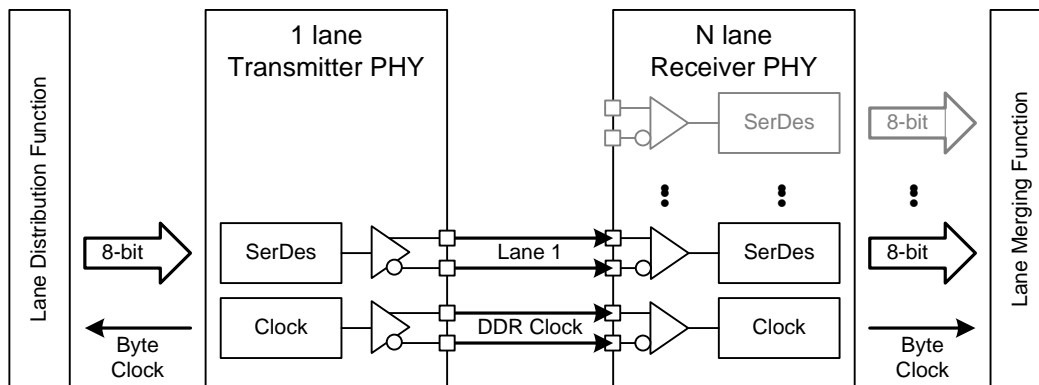


Figure 32 One Lane Transmitter and N-Lane Receiver Example for D-PHY

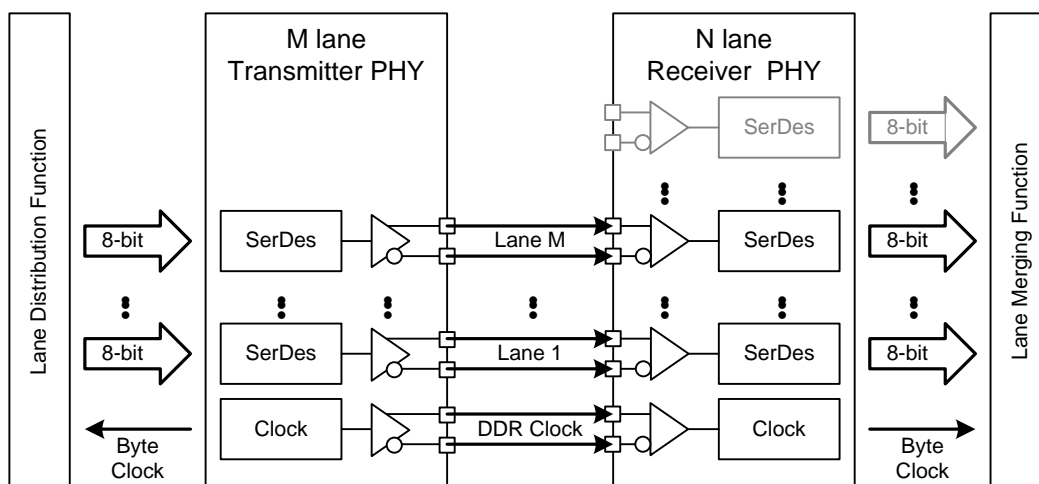


Figure 33 M-Lane Transmitter and N-Lane Receiver Example ($M < N$) for D-PHY

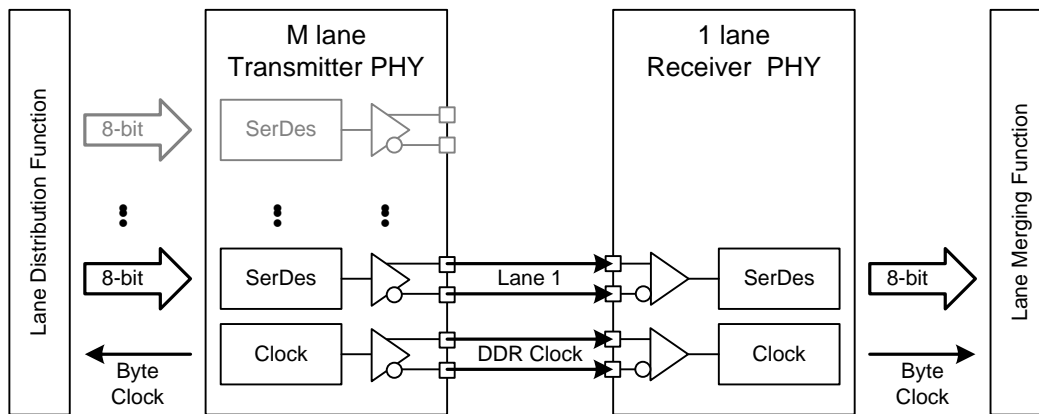


Figure 34 M-Lane Transmitter and One Lane Receiver Example for D-PHY

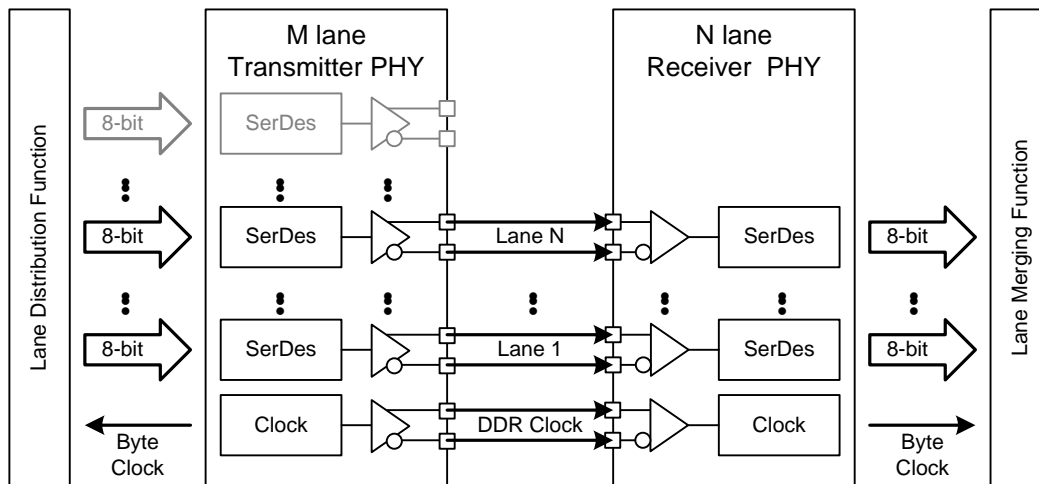


Figure 35 M-Lane Transmitter and N-Lane Receiver Example (N<M) for D-PHY

10-Nov-2016

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 36** shows a mechanism for clocking data from the elastic buffers, in order to align (De-Skew) all RxDataHS to one RxWordClkHS.

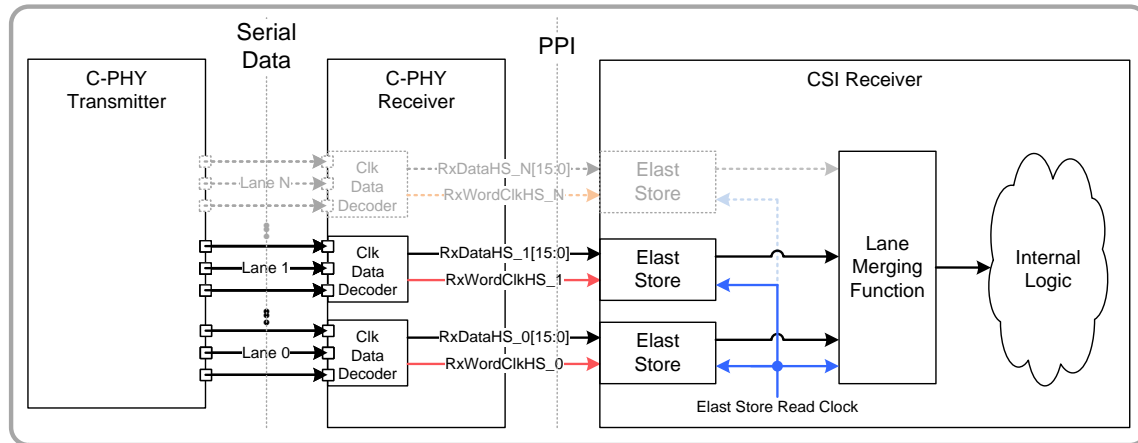


Figure 36 Example of Digital Logic to Align All RxDataHS

This page intentionally left blank

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)

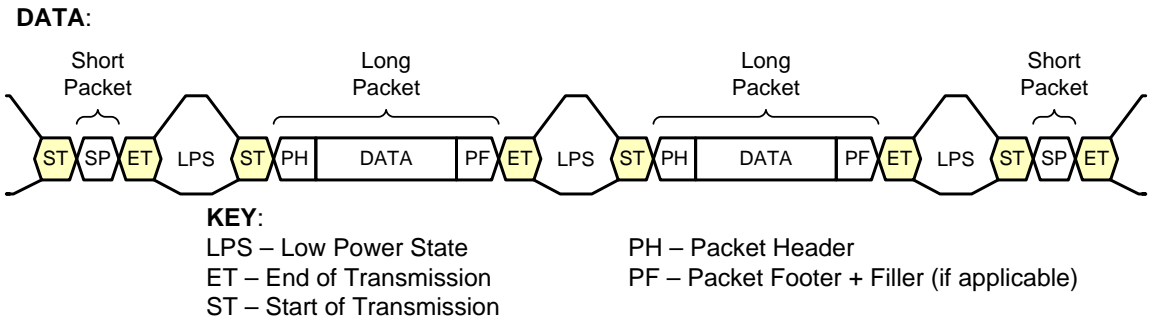


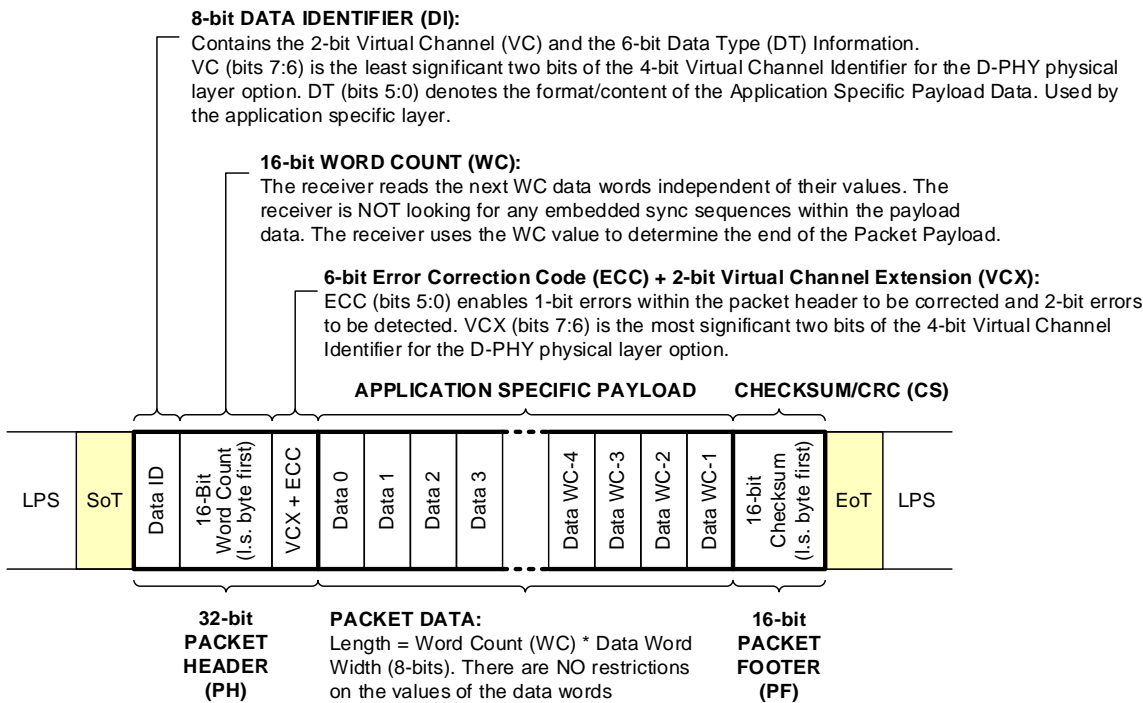
Figure 37 Low Level Protocol Packet Overview

518 **9.1 Low Level Protocol Packet Format**

519 As shown in **Figure 37**, two packet structures are defined for low-level protocol communication: Long
520 packets and Short packets. The format and length of Short and Long Packets depends on the choice of
521 physical layer. For each packet structure, exit from the low power state followed by the Start of
522 Transmission (SoT) sequence indicates the start of the packet. The End of Transmission (EoT) sequence
523 followed by the low power state indicates the end of the packet.

524 **9.1.1 Low Level Protocol Long Packet Format**

525 **Figure 38** shows the structure of the Low Level Protocol Long Packet for the D-PHY physical layer option.
526 A Long Packet shall be identified by Data Types 0x10 to 0x37. See **Table 3** for a description of the Data
527 Types. A Long Packet for the D-PHY physical layer option shall consist of three elements: a 32-bit Packet
528 Header (PH), an application specific Data Payload with a variable number of 8-bit data words, and a 16-bit
529 Packet Footer (PF). The Packet Header is further composed of four elements: an 8-bit Data Identifier, a 16-bit
530 Word Count field, a 2-bit Virtual Channel Extension field, and a 6-bit ECC. The Packet footer has one
531 element, a 16-bit checksum (CRC). See **Section 9.2** through **Section 9.5** for further descriptions of the
532 packet elements.



533 **Figure 38 Long Packet Structure for D-PHY Physical Layer Option**

Figure 39 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 39, 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).

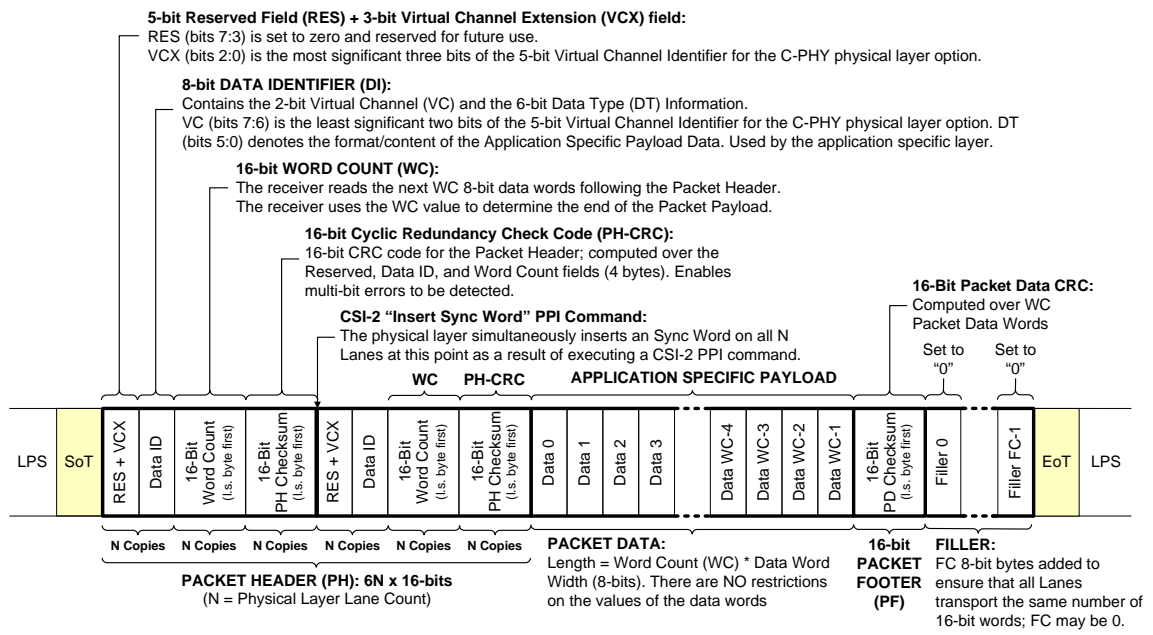


Figure 39 Long Packet Structure for C-PHY Physical Layer Option

As shown in **Figure 40**, the Packet Header structure depicted in **Figure 39** 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.

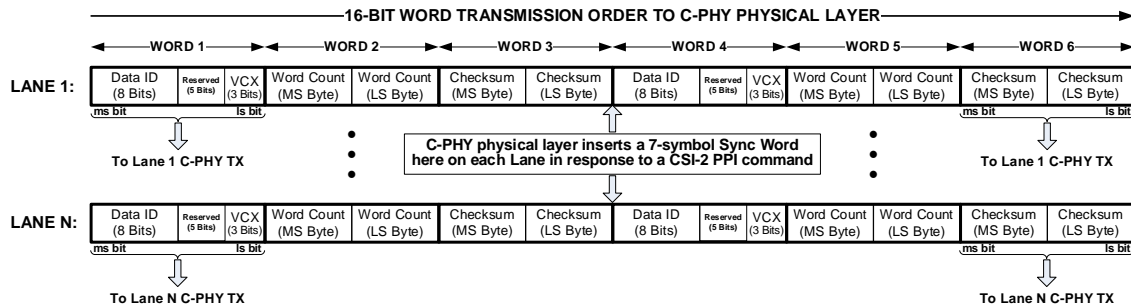


Figure 40 Packet Header Lane Distribution for C-PHY Physical Layer Option

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 40**) 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

10-Nov-2016

(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 $(WC \bmod 2) + \{ \{N - ((WC + 2 + (WC \bmod 2)) / 2) \bmod N\} \bmod N \} * 2$, where N is the number of Lanes. Note that it is possible for FC to be zero.

Figure 41 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 $n \geq 0$) 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.

**Figure 41 Minimal Filler Byte Insertion Requirements for Three Lane C-PHY**

10-Nov-2016

9.1.2 Low Level Protocol Short Packet Format

Figure 42 and **Figure 43** 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 3** 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 6** 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.

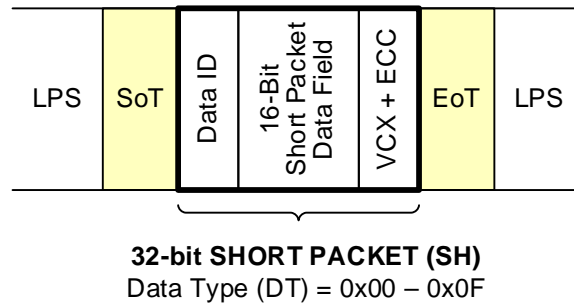


Figure 42 Short Packet Structure for D-PHY Physical Layer Option

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.

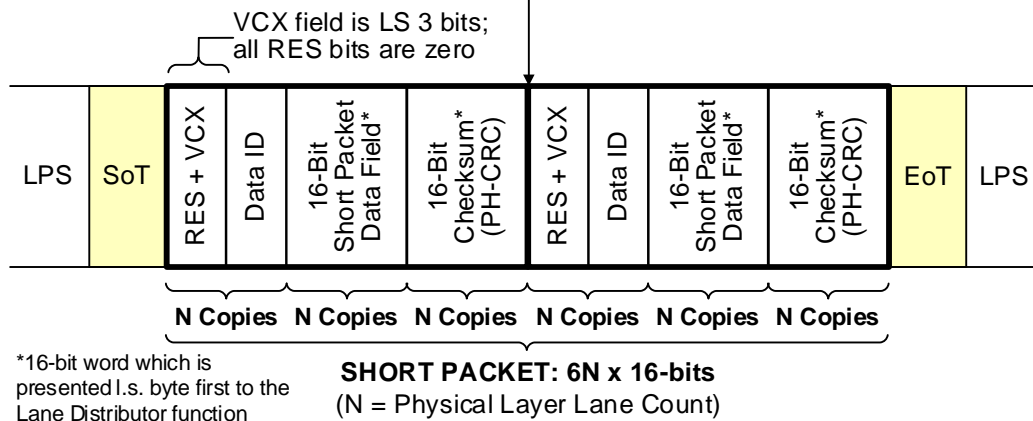


Figure 43 Short Packet Structure for C-PHY Physical Layer Option

9.2 Data Identifier (DI)

The Data Identifier byte contains the Virtual Channel (VC) and Data Type (DT) fields as illustrated in *Figure 44*. 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.

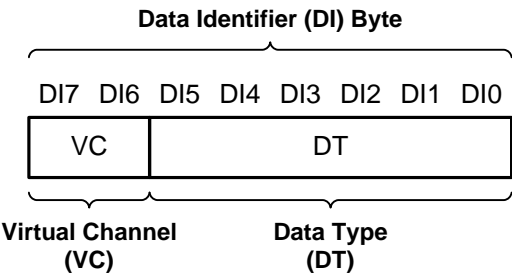


Figure 44 Data Identifier Byte

10-Nov-2016

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 45**, 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 38** and **Figure 39**, 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.

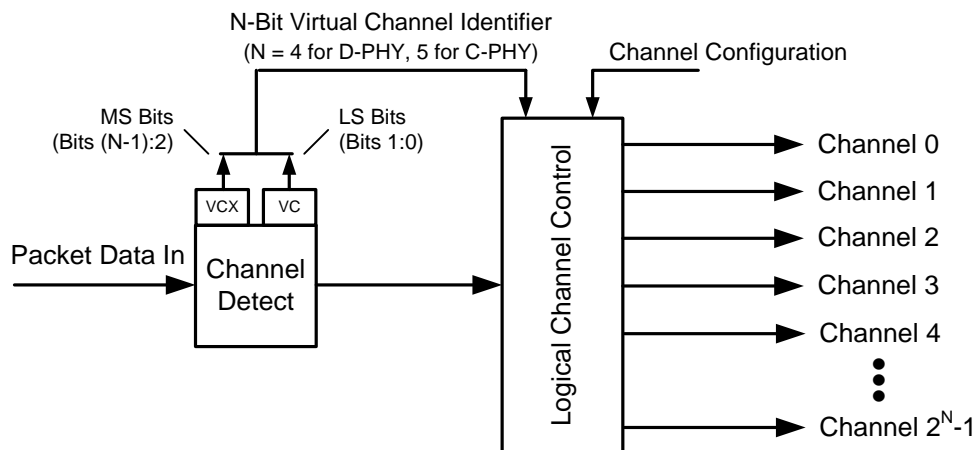


Figure 45 Logical Channel Block Diagram (Receiver)

Figure 46 illustrates an example of data streams utilizing virtual channel support.

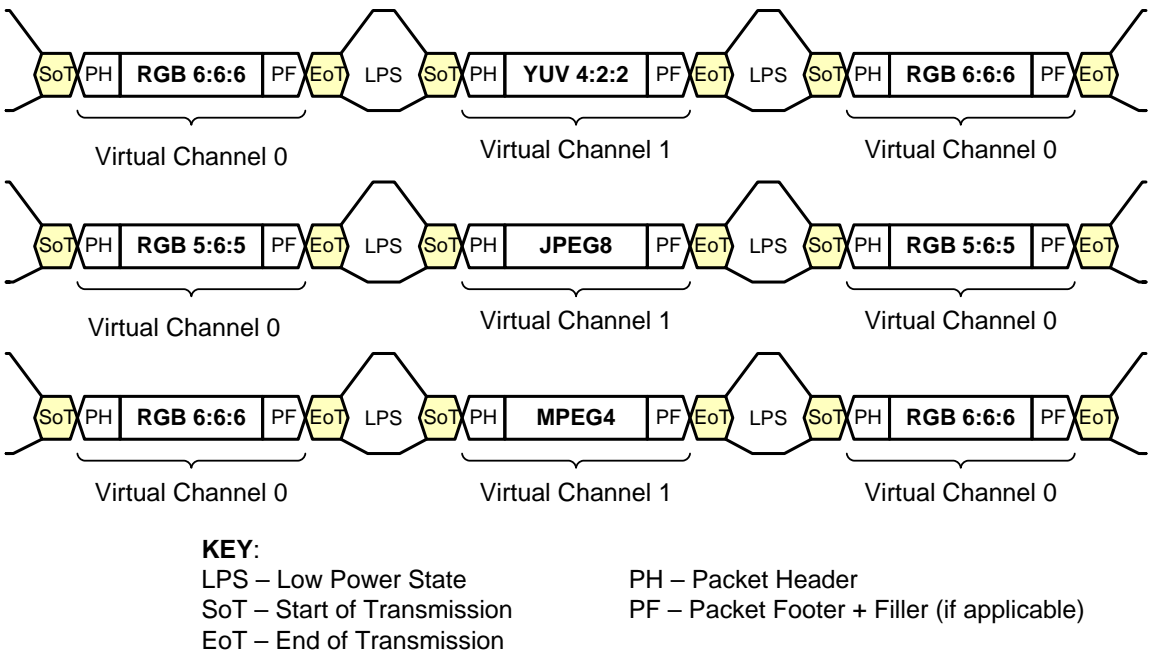


Figure 46 Interleaved Video Data Streams Examples

10-Nov-2016

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 3*. 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 3 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

661

662

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

663

664

665

666

667

668

669

670

671

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 results 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 47*, which also serves as an ECC calculation example.

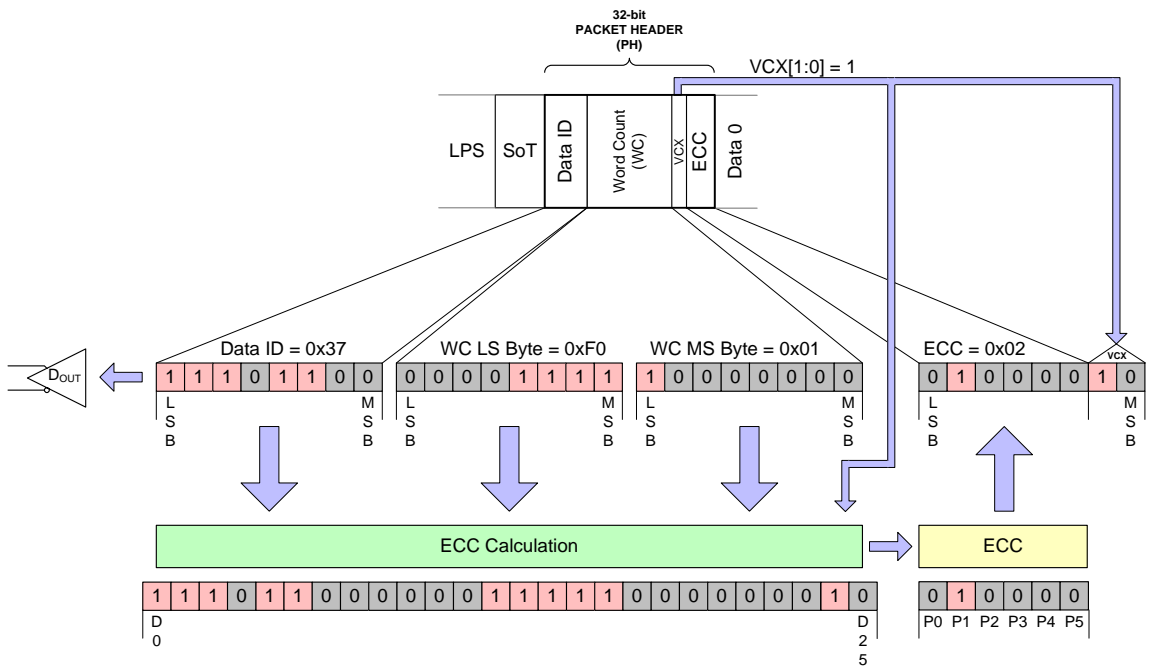


Figure 47 26-bit ECC Generation Example

10-Nov-2016

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:

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

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 \mathbf{G} . The resulting product is the code-word vector $(c_1, c_2, c_3 \dots c_n)$, consisting of the original data bits and the calculated parity bits. The generator matrix \mathbf{G} used in constructing Hamming codes consists of \mathbf{I} (the identity matrix) and a parity generation matrix \mathbf{A} :

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

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

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

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

If all elements of \mathbf{s} are zero, the code word was received correctly. If \mathbf{s} contains non-zero elements, then at least one error is present. If a single bit error is encountered then the syndrome \mathbf{s} is one of the elements of \mathbf{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 \mathbf{I} , else it will be the data bit identified by the position of the syndrome in \mathbf{A}^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 4** is used:

Table 4 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. $P_{0,26\text{-bits}} = D0 \wedge D1 \wedge D2 \wedge D4 \wedge D5 \wedge D7 \wedge D10 \wedge D11 \wedge D13 \wedge D16 \wedge D20 \wedge D21 \wedge D22 \wedge D23 \wedge D24$

Similarly, to derive the parity P0 for 64-bits, all P0's in **Table 5** 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 4**. These syndromes identify the bit position in error. The syndrome is calculated as:

$S = P_{\text{SEND}} \wedge P_{\text{RECEIVED}}$, 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 5 represents the same information as the matrix in **Table 4**, 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.

10-Nov-2016

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 5** [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 5 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

10-Nov-2016

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 48** for a 64-bit header.

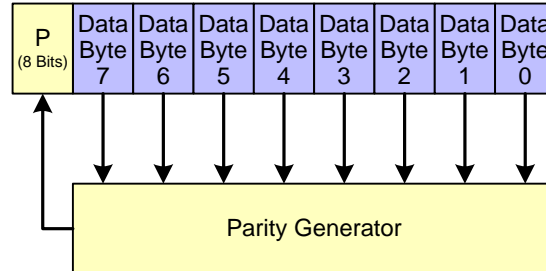


Figure 48 64-bit ECC Generation on TX Side

And **Figure 49** for a 26-bit header:

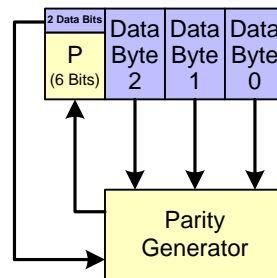


Figure 49 26-bit ECC Generation on TX Side

The parity generators are based on **Table 5**.

$$\text{e.g. } P_{32\text{-bit}} = D1 \wedge D2 \wedge D3 \wedge D7 \wedge D8 \wedge D9 \wedge D13 \wedge D14 \wedge D15 \wedge D19 \wedge D20 \wedge D21 \wedge D23 \wedge D24 \wedge D25$$

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 50** depicts ECC processing for 64 received Packet Header data bits, using 8 parity bits.

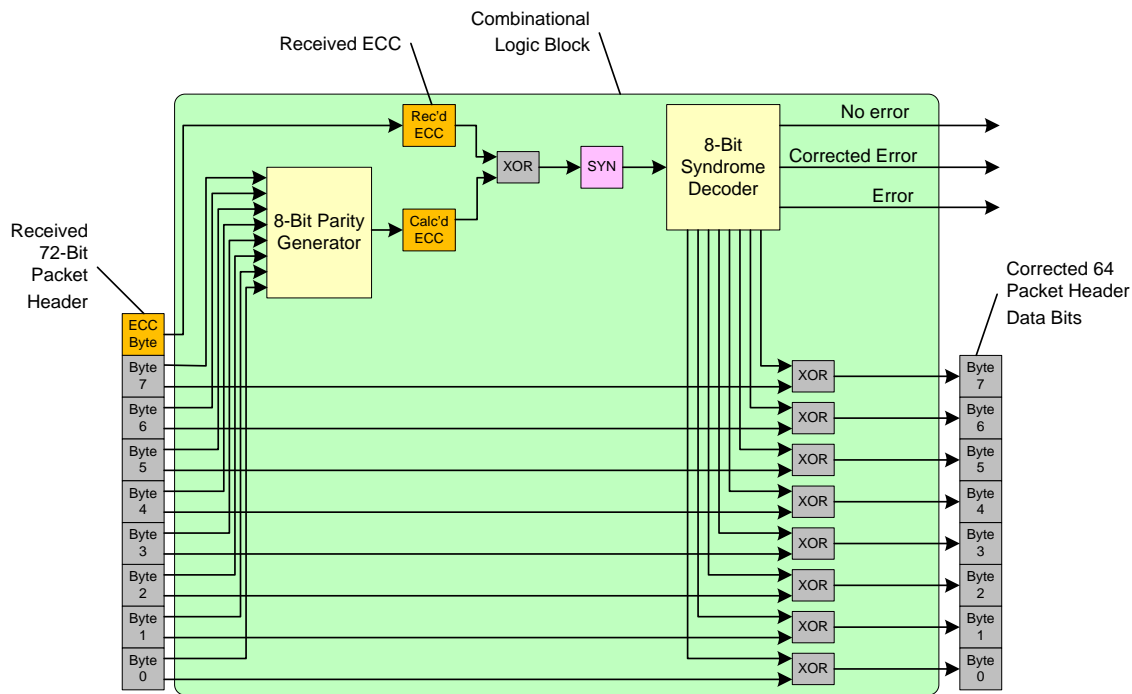


Figure 50 64-bit ECC on RX Side Including Error Correction

Decoding the syndrome has four possible outcomes:

1. If the syndrome is 0, no errors are present.
2. If the syndrome matches one of the matrix entries in the **Table 4**, then a single bit error has occurred and the corresponding bit position may be corrected by inverting it (e.g. by XORing with '1').
3. 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.
4. 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 51** 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 51**). 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.

10-Nov-2016

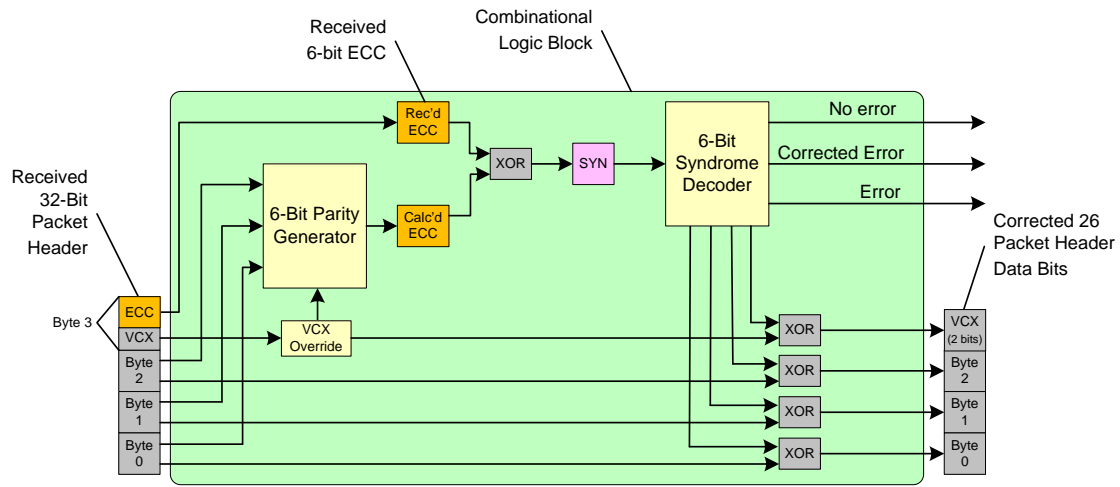


Figure 51 26-bit ECC on RX Side Including Error Correction

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 38*, *Figure 39*, and *Figure 43*.

The order in which the checksum bytes are presented to the Lane Distributor function is illustrated in *Figure 52*.

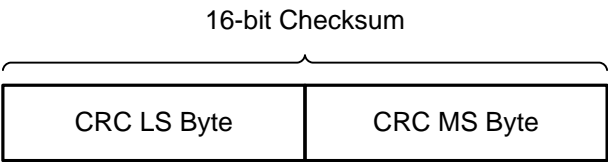


Figure 52 Checksum Transmission Byte Order

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.

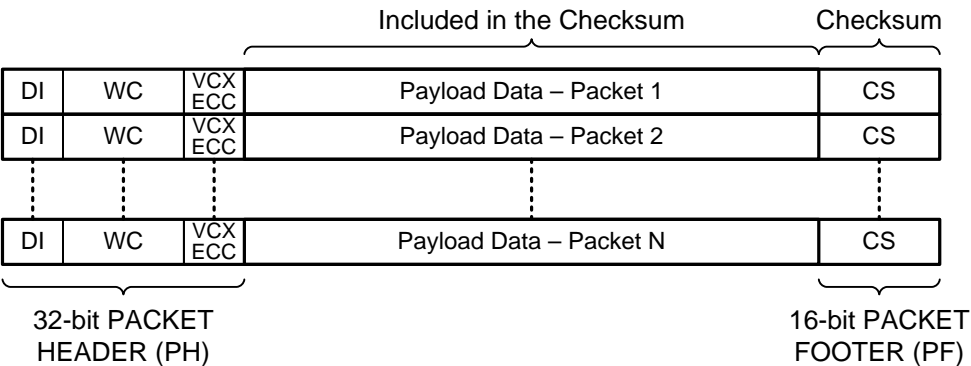
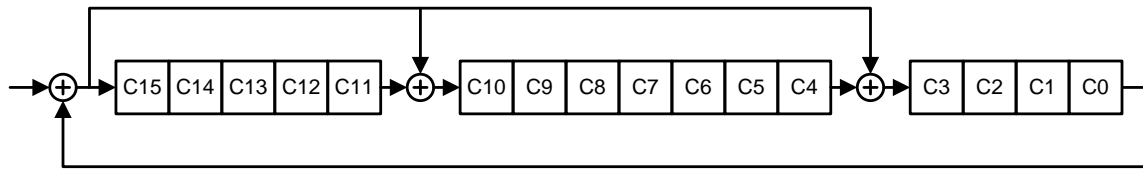


Figure 53 Checksum Generation for Long Packet Payload Data

The definition of a serial CRC implementation is presented in *Figure 54*. The CRC implementation shall be functionally equivalent with the C code presented in *Figure 55*. 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 55* equals the final contents of the C[15:0] shift register shown in *Figure 54*. 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.

10-Nov-2016



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

Note: C15 represents x^0 , C0 represents x^{15}

Figure 54 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 55 16-bit CRC Software Implementation Example

Beginning with index 0, the contents of the input data array in **Figure 55** 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 56 illustrates the packet spacing with the LPS.

The packet spacing illustrated in Figure 56 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.

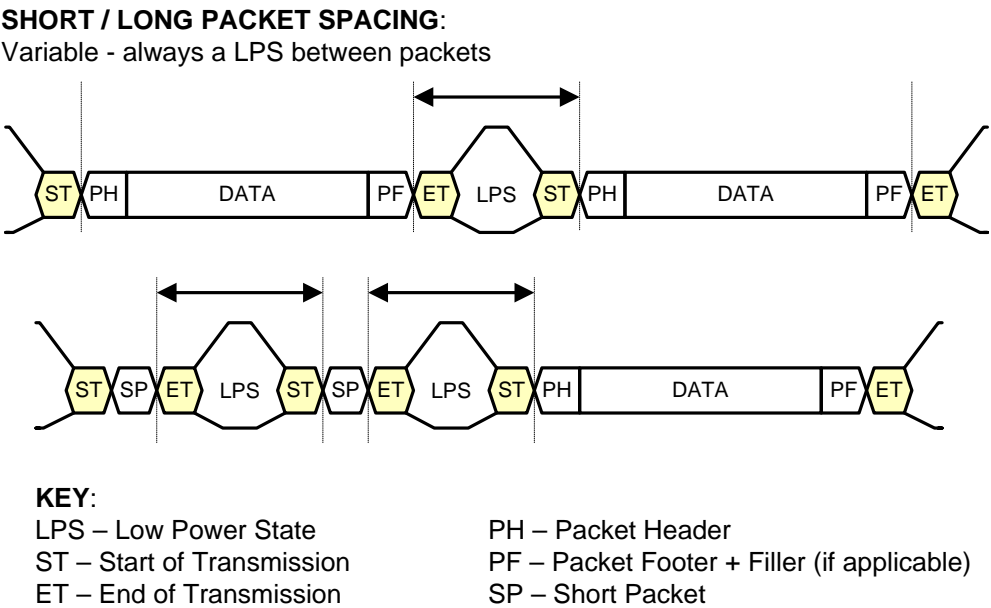


Figure 56 Packet Spacing

10-Nov-2016

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 6 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 6* 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 as one of the following

- Frame number is always zero – frame number is inoperative.
- Frame number increments by 1 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

The frame number must be a non-zero value.

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.

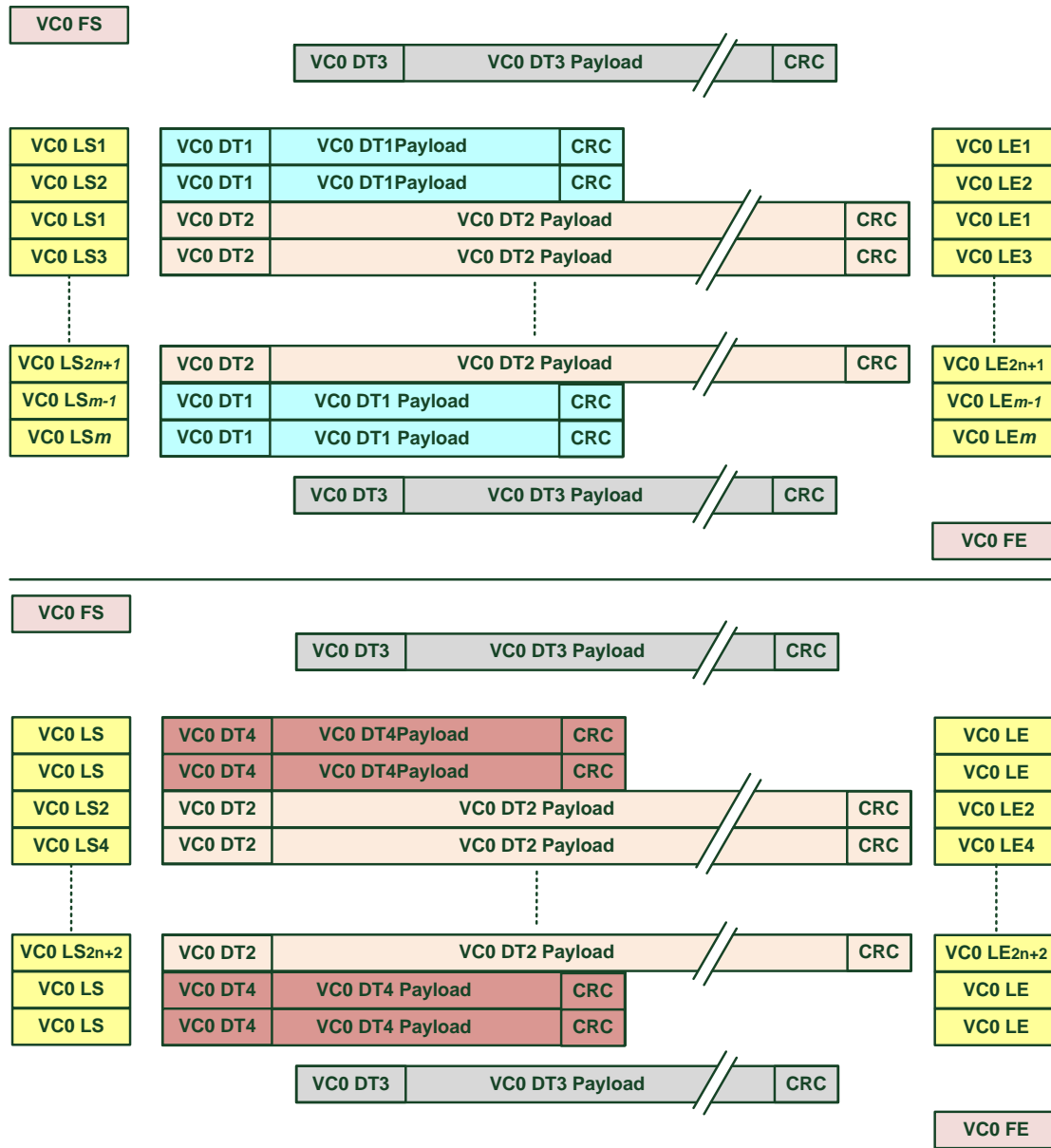
Or:

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 57 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.

10-Nov-2016

**Note:**

- For VC0 DT2 Odd Frames LS_{2n+1} and Even Frames LS_{2n+2} (where n=0,1,2,3...) the first line n=0
- For VC0 DT1 LS_{m+1} (where m=0,1,2,3...) the first line m=0

861

Figure 57 Example Interlaced Frame Using LS/SE Short Packet and Line Counting

9.9 Generic Short Packet Data Type Codes

Table 7 lists the Generic Short Packet Data Types.

Table 7 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.

10-Nov-2016

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 58 and Figure 59 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.

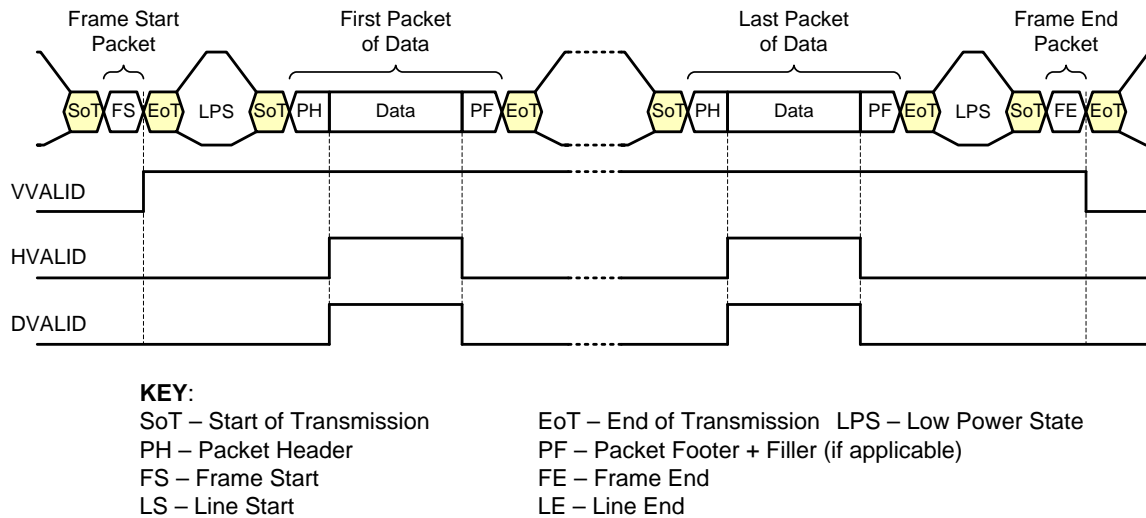


Figure 58 Multiple Packet Example

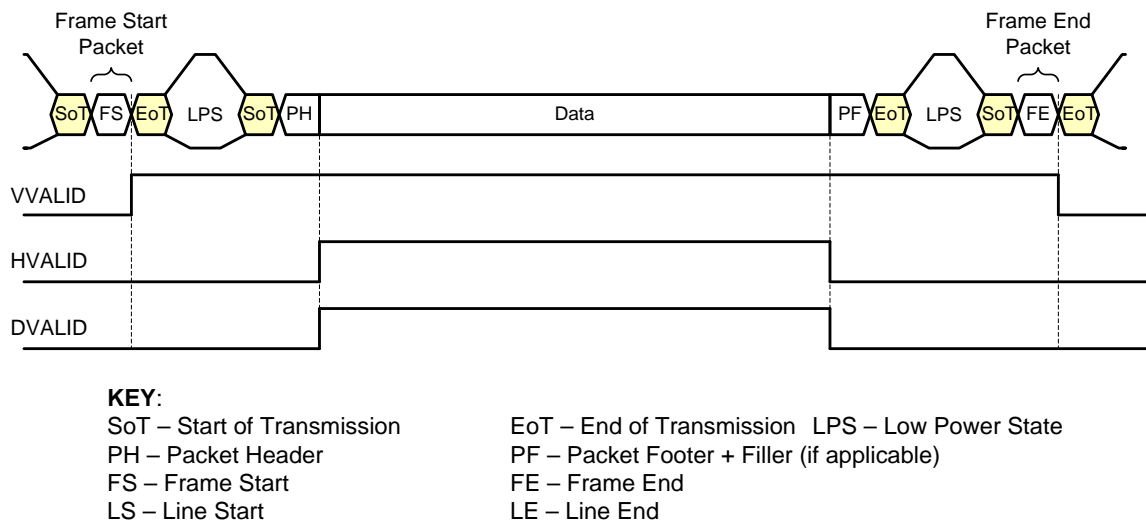


Figure 59 Single Packet Example

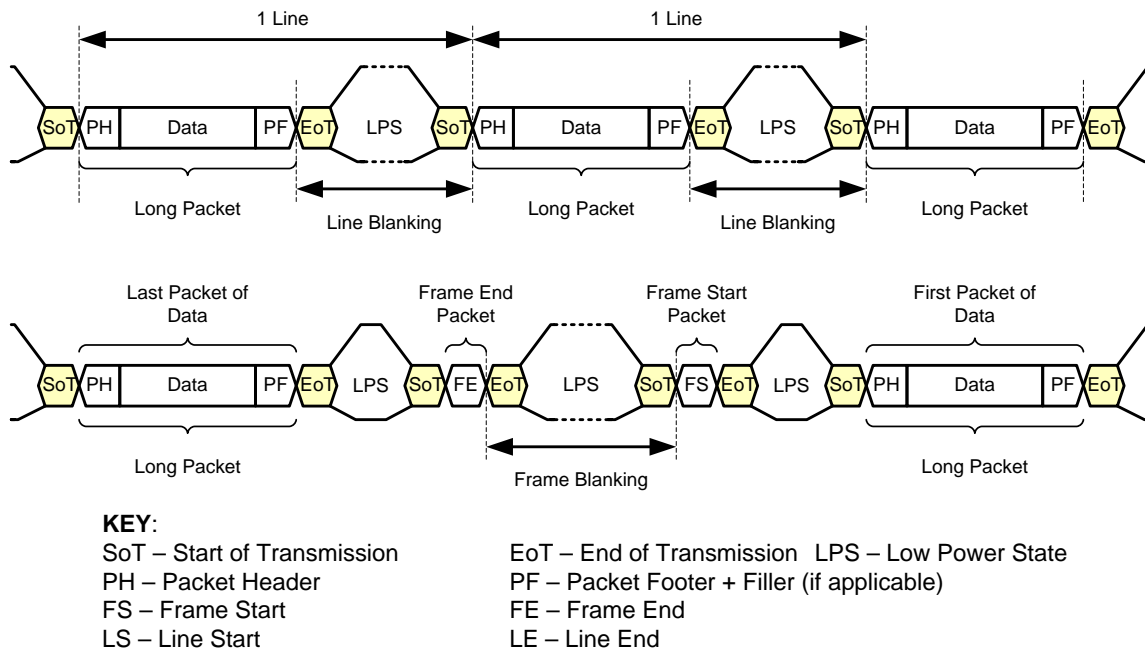


Figure 60 Line and Frame Blanking Definitions

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 60**).

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

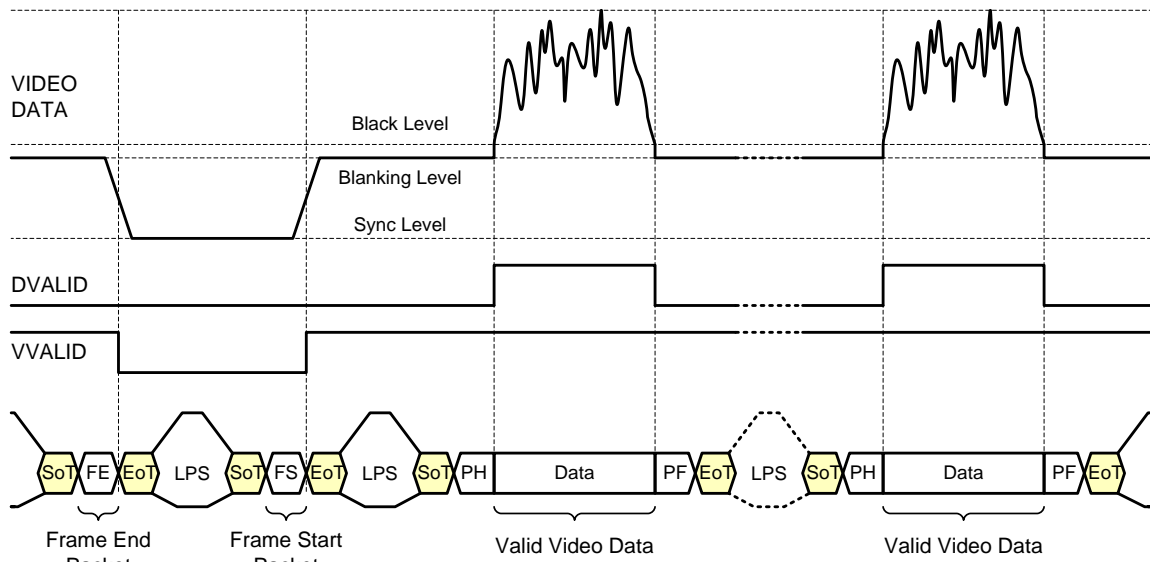
The intention is to ensure that the Frame Start and Frame End packets accurately denote the start and end of a frame of image data. A valid exception is when the positions of the Frame Start and Frame End packets are being used to convey pixel level accurate vertical synchronization timing information.

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 61**.

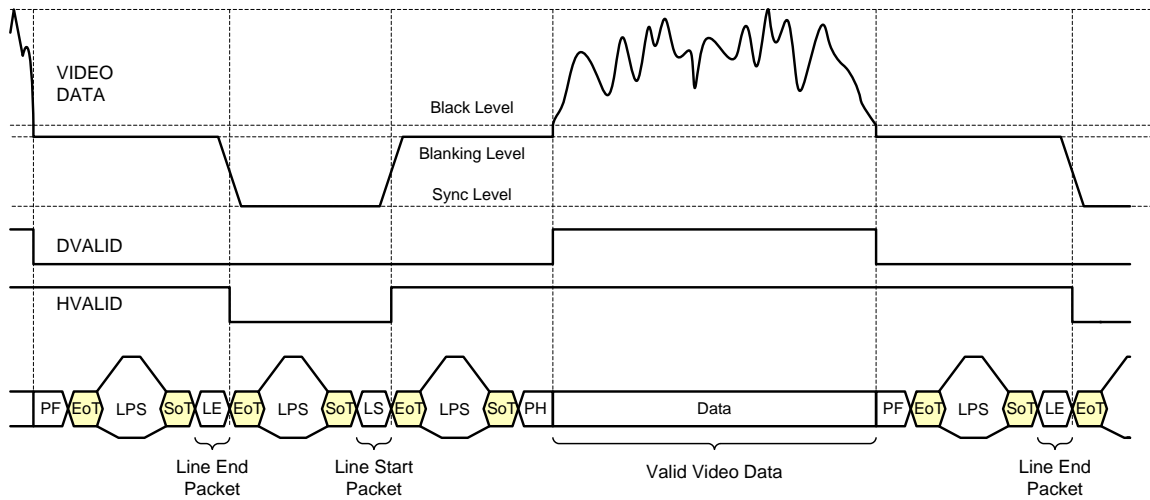
If pixel level accurate horizontal synchronization timing information is required, Line Start and Line End packets should be used to achieve it.

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 62**.

10-Nov-2016



904

Figure 61 Vertical Sync Example

905

Figure 62 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.

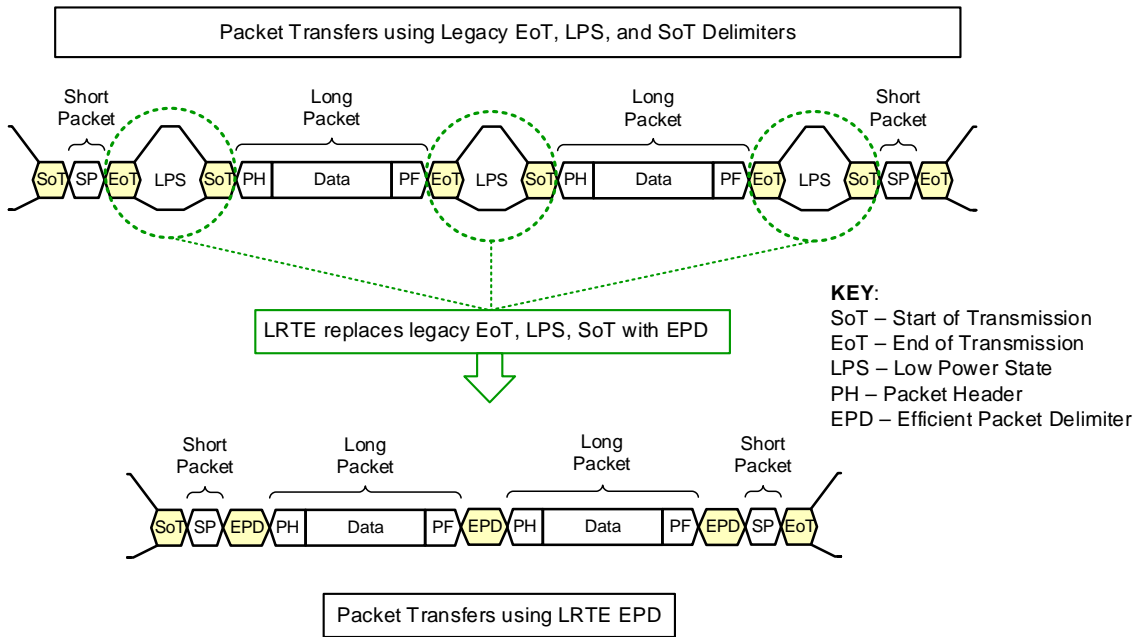


Figure 63 Interpacket Latency Reduction Using LRTE EPD

10-Nov-2016

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 control code (3444_443) 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 “RxDetectSyncHS” 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.

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 for 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 for 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 combination of the **TX_REG_CSI_EPD_EN_SSP** and **TX_REG_CSI_EPD_OP_SLP** registers.

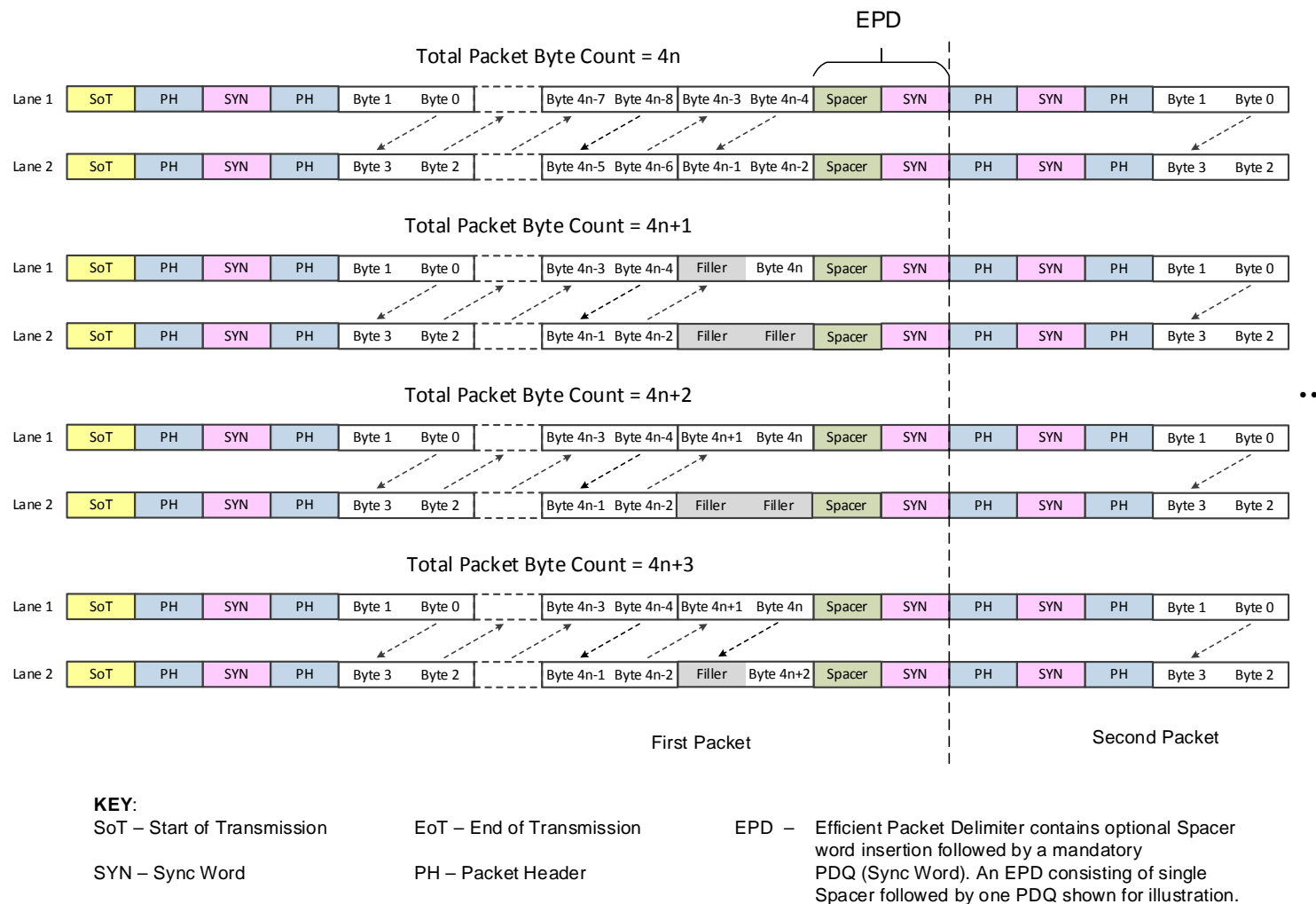


Figure 64 LRTE Efficient Packet Delimiter Example for CSI-2 over C-PHY (2 Lanes)

10-Nov-2016

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 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. 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 “TxSendPDQHS” 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 “RXDetectPDQHS” at the PPI in order to notify the CSI-2 protocol layer.

Number of Bytes, B, transmitted is NOT an integer multiple of the number of lanes, N with alignment using Filler bytes for packet transfers using PHY generated and consumed PDQ. One optional Spacer byte insertion included for illustration.

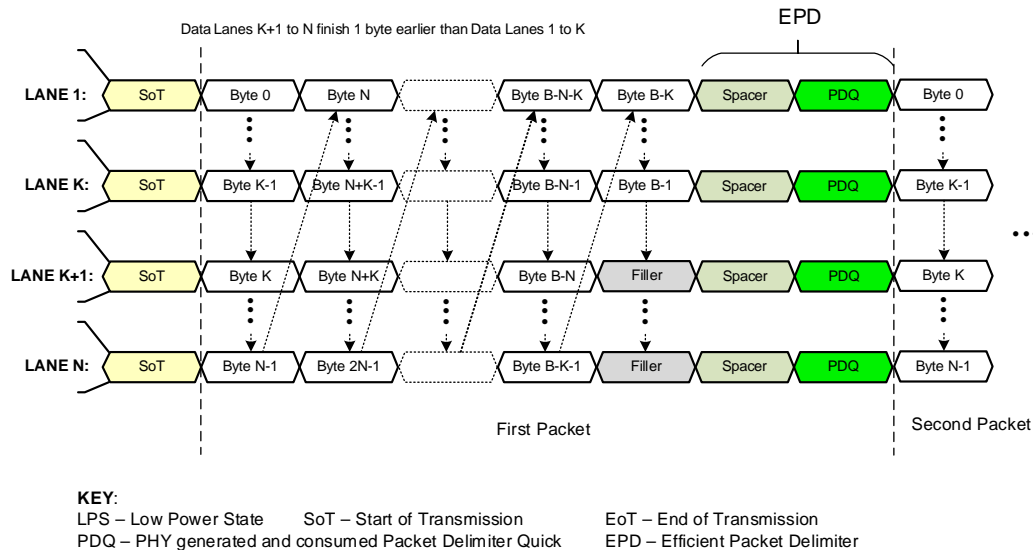


Figure 65 Example of LRTE EPD for CSI-2 Over D-PHY – Option 1

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.

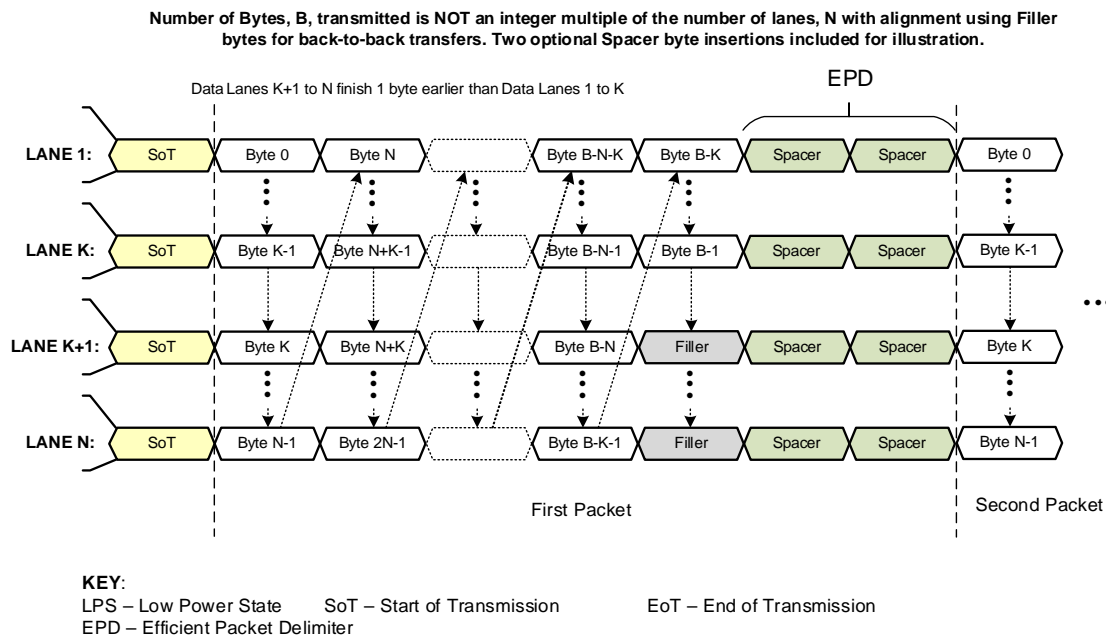


Figure 66 Example of LRTE EPD for CSI-2 Over D-PHY – Option 2

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: 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 for CSI-2 Short Packets. These Spacer insertions for CSI-2 Short Packets applies 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 not enabled, then the remaining fifteen bits of this register (bits [14:0]) shall be used to generate up to 32,767 optional Spacer insertions for CSI-2 Long Packets. These Spacer insertions for CSI-2 Long Packets apply to both D-PHY EPD options.

10-Nov-2016

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 combination 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 [MIP101] for the D-PHY physical layer option.

9.11.2 Using ILR and Enhanced Transport Efficiency Together

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

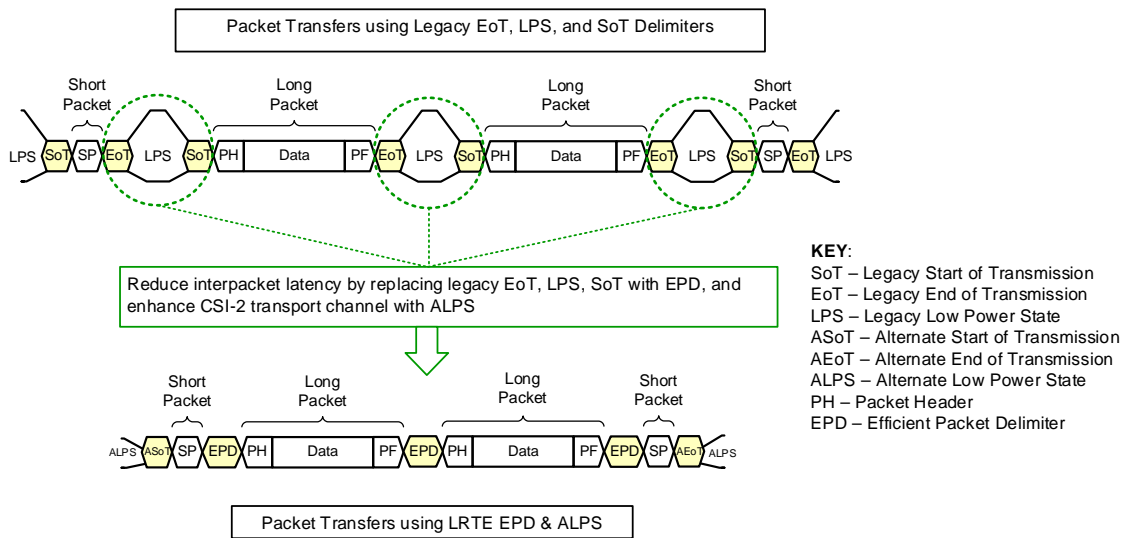


Figure 67 Using EPD and ALPS Together

9.11.3 LRTE Register Tables

Table 8 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 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 Spacers may be Inserted for Short Packets and Long Packets. See Figure 64 .
Bits [14:0]: EPD Short Packet Spacers	The number of Spacer Words 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 number of Spacer Words 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.

10-Nov-2016

1007

Table 9 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 65 . If EPD is enabled, the D-PHY EPD Options are determined by TX_REG_CSI_EDP_OP_SLP[15] .
Bits [14:0]: EPD Short Packet Spacers	The number of Spacer Bytes 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_EDP_EN_SSP[15]). The Short Packet Spacers may range from 0 to 32,767 Bytes. See Figure 65 and Figure 66 .
TX_REG_CSI_EDP_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 65 . D-PHY EPD Option 2: CSI-2 over D-PHY EPD operation using optional Spacer Insertion(s). See Figure 66 .
Bits [14:0]: Long Packet Spacers	The number of Spacer Bytes 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_EDP_EN_SSP[15]). The Long Packet Spacers may range from 0 to 32,767 Bytes. See Figure 65 and Figure 66 .

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 68*. 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.

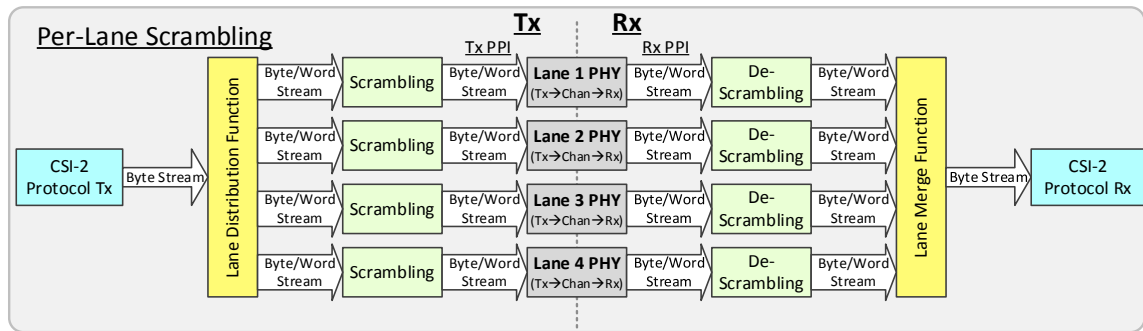


Figure 68 System Diagram Showing Per-Lane Scrambling

10-Nov-2016

9.12.1 CSI-2 Scrambling for D-PHY

Figure 69 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.
2. Whenever the optional D_PHY HS-Idle is transmitted.

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

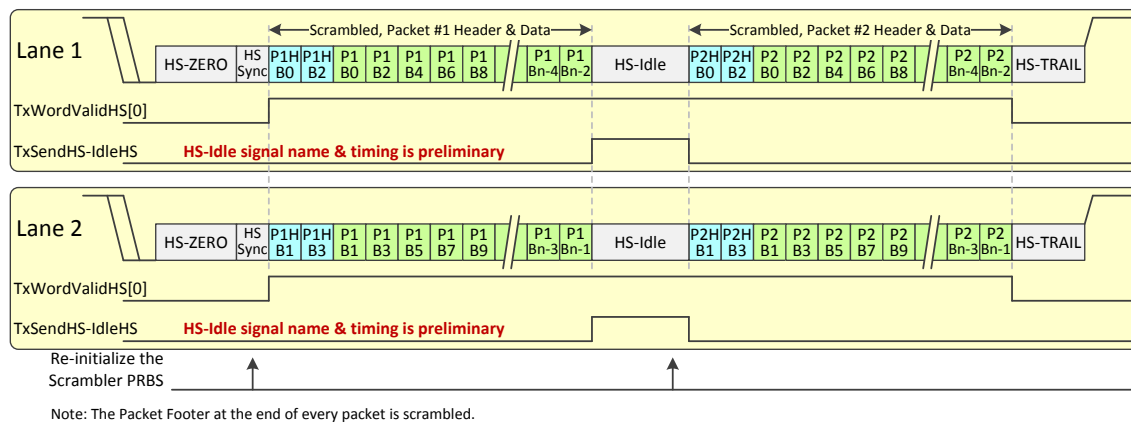


Figure 69 Example of Data Bursts in Two Lanes Using the D-PHY Physical Layer

9.12.2 CSI-2 Scrambling for C-PHY

Figure 70 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.

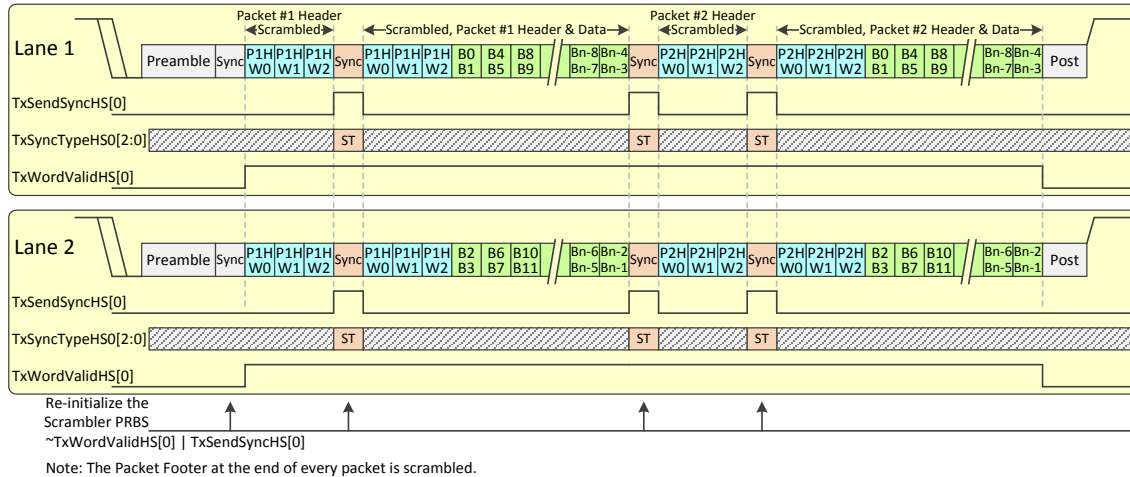


Figure 70 Example of Data Bursts in Two Lanes Using the C-PHY Physical Layer

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 pseudorandom 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 10 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 10** for convenience. The CSI-2 protocol uses only the first four out of the five possible Sync Types, which simplifies the implementation.

Table 10 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

10-Nov-2016

Note:

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

Figure 71 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.

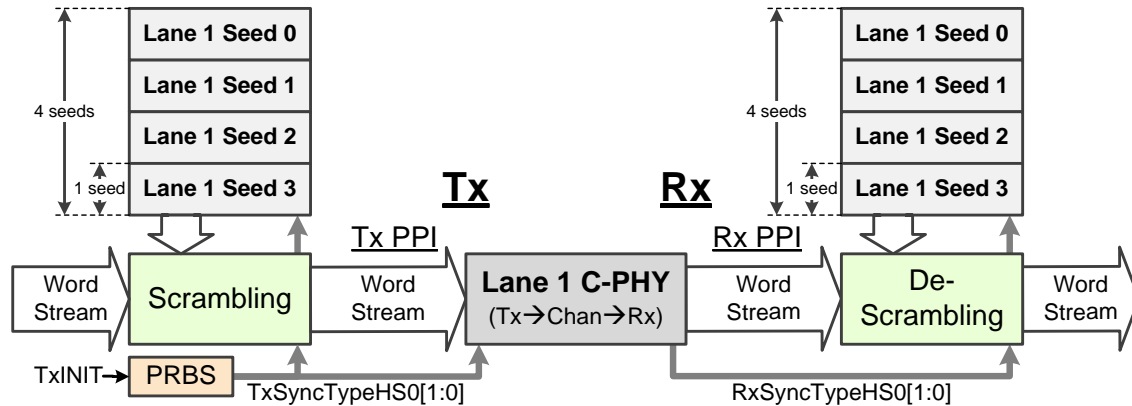


Figure 71 Generating Tx Sync Type as Seed Index (Single Lane View)

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 71. 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 binary sequence (PRBS), and shall be generated using the Galois form of an LFSR implementing the generator polynomial:

$$G(x) = x^9 + x^5 + 1$$

The least significant two bits of the generator shall be used as the seed index. 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 71. Figure 68 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.

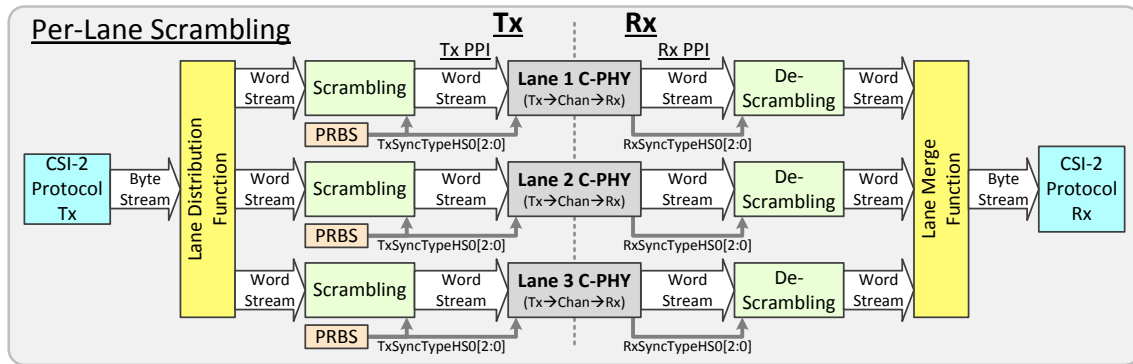


Figure 72 Generating Tx Sync Type Using the C-PHY Physical Layer

10-Nov-2016

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 11* lists all of the fields that are not scrambled.

Table 11 Fields That Are Not Scrambled

PHY	PHY-Generated	CSI-2-Protocol-Generated
D-PHY	<ul style="list-style-type: none"> • HS-Zero • Sync Word (aka Leader Sequence) • HS Trail • SoT • EoT • HS-Idle • All fields of the deskew sequence (aka deskew burst) including: <ul style="list-style-type: none"> • HS-Zero • Deskew sync pattern • '01010101' data • HS-Trail 	<ul style="list-style-type: none"> • LP Mode transactions for SoT, EoT and ULPS
C-PHY	<ul style="list-style-type: none"> • Preamble (including t_{3-PREBEGIN}, t_{3-PROGSEQ} and t_{3-PREEND}) • Sync Word • Post • SoT • EoT 	<ul style="list-style-type: none"> • 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 12* should be used to initialize the D-PHY scrambler LFSR in Lanes 1 through 8.

Table 12 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 12** 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 13 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. Bits 8–N output from $G(x)$ affect bit 0 of the data bytes, and bits 8–N+7 output from $G(x)$ affect bit 7 of the data bytes.

10-Nov-2016

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.

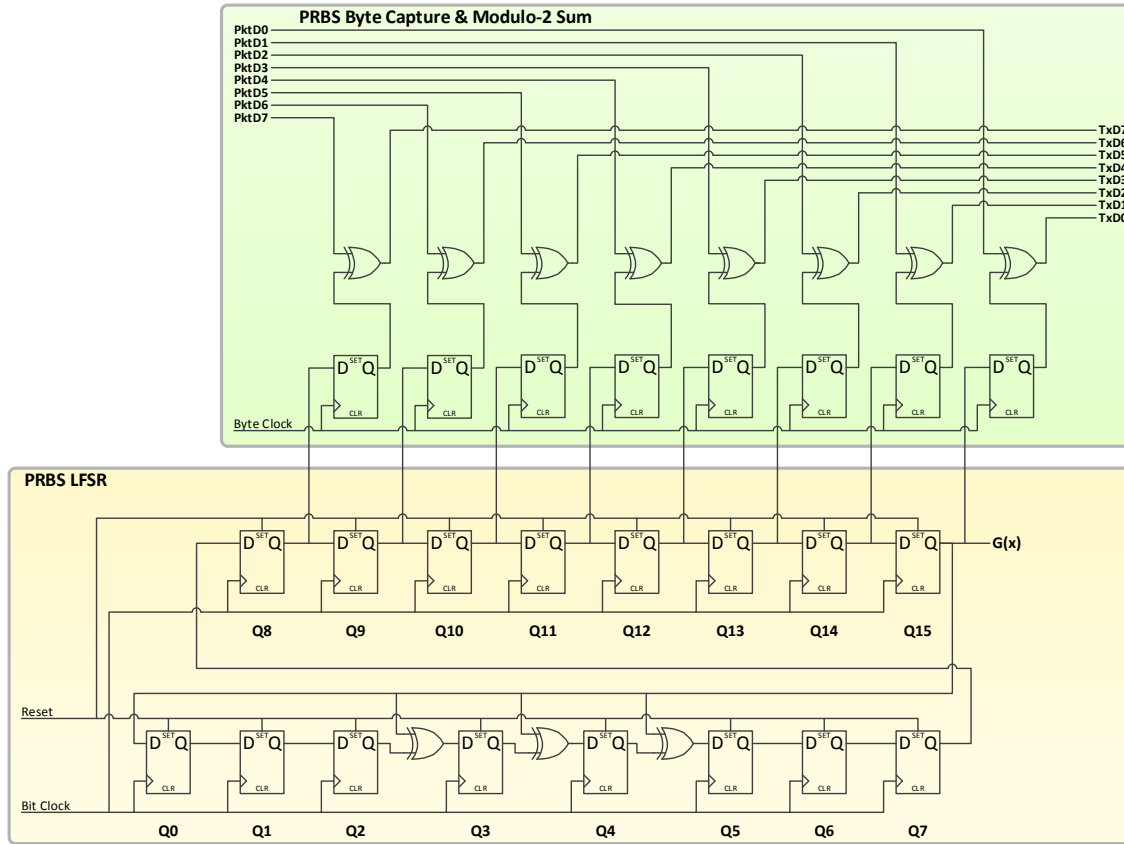


Figure 73 PRBS LFSR Serial Implementation Example

Table 14 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 73**) when the register contains the initial seed value.

Table 14 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 15 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 15 Example PRBS LFSR Byte Sequence for D-PHY Physical Layer

Scrambling Sequence Byte #	PRBS Register	PRBS Byte Output
Initial Seed, Byte 0	0x1188	0x88
Byte 1	0x8ba9	0xd1
Byte 2	0xb403	0x2d
Byte 3	0x1bd4	0xd8
Byte 4	0xd613	0x6b
Byte 5	0x02c6	0x40
Byte 6	0xc672	0x63
Byte 7	0x6056	0x06
Byte 8	0x5f60	0xfa
Byte 9	0x6cb7	0x36

10-Nov-2016

1120 **Table 16** shows the first twelve PRBS Word Outputs produced by the PRBS LFSR in Lane 2 when the
 1121 C-PHY physical layer is being used.

1122 **Table 16 Example PRBS LFSR Byte Sequence for C-PHY Physical Layer**

Scrambling Sequence Word #	PRBS Register	PRBS Word
Initial Seed, Word 0	0x1188	0xd188
Word 1	0xb403	0xd82d
Word 2	0xd613	0x406b
Word 3	0xc672	0x0663
Word 4	0x5f60	0x36fa
Word 5	0xbf4c	0xaafd
Word 6	0x5a0d	0x805a
Word 7	0x6a39	0x8c56
Word 8	0xde89	0x997b
Word 9	0x10e1	0x4708
Word 10	0x8592	0x71a1
Word 11	0x40de	0x0b02

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 74*
 - Digital Interlaced Video Example, *Figure 75*
 - Digital Interlaced Video with accurate synchronization timing information, *Figure 76*

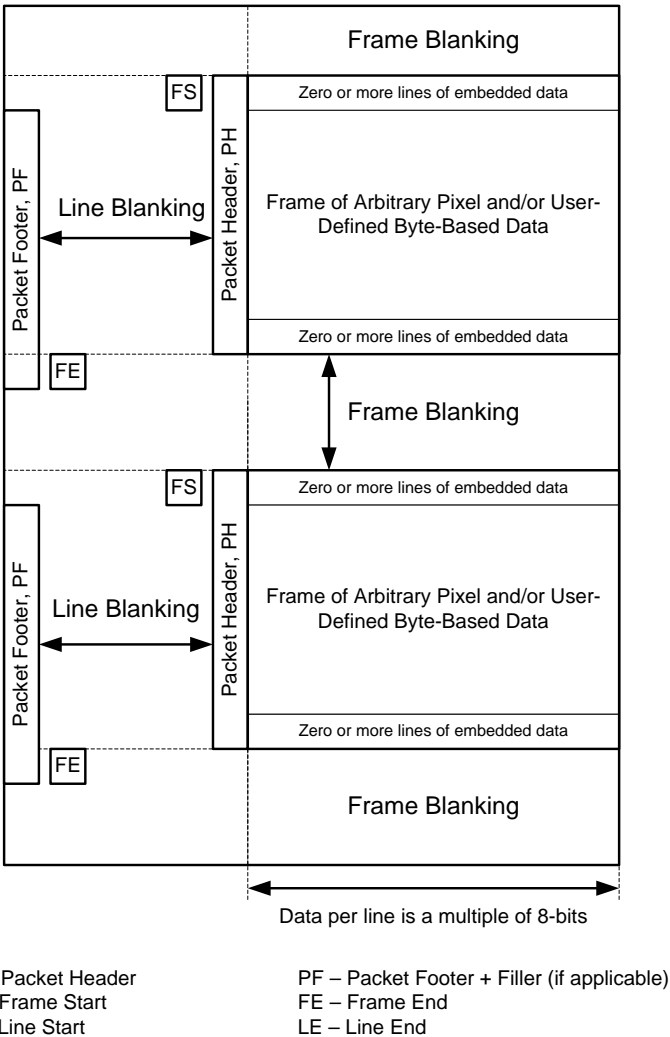


Figure 74 General Frame Format Example

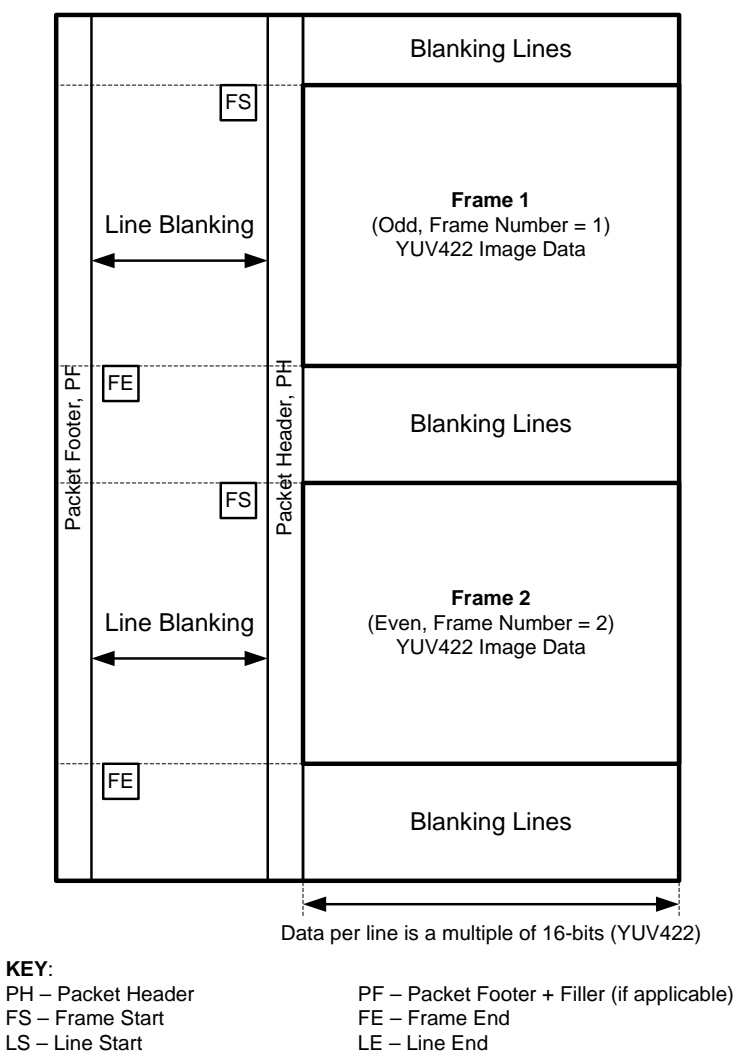
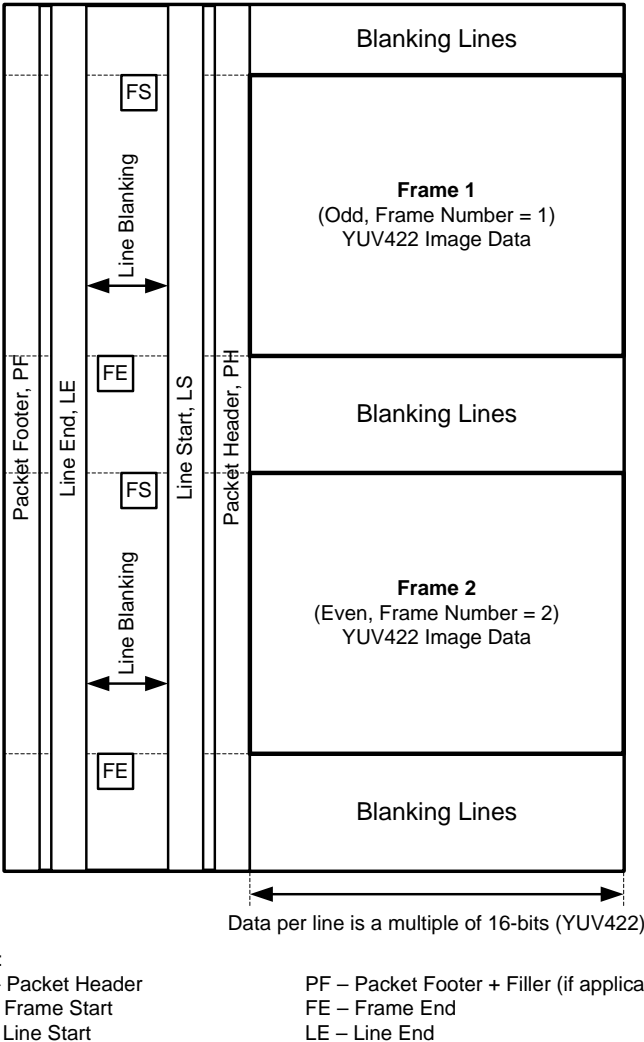


Figure 75 Digital Interlaced Video Example



1145

Figure 76 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 77**. 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**.

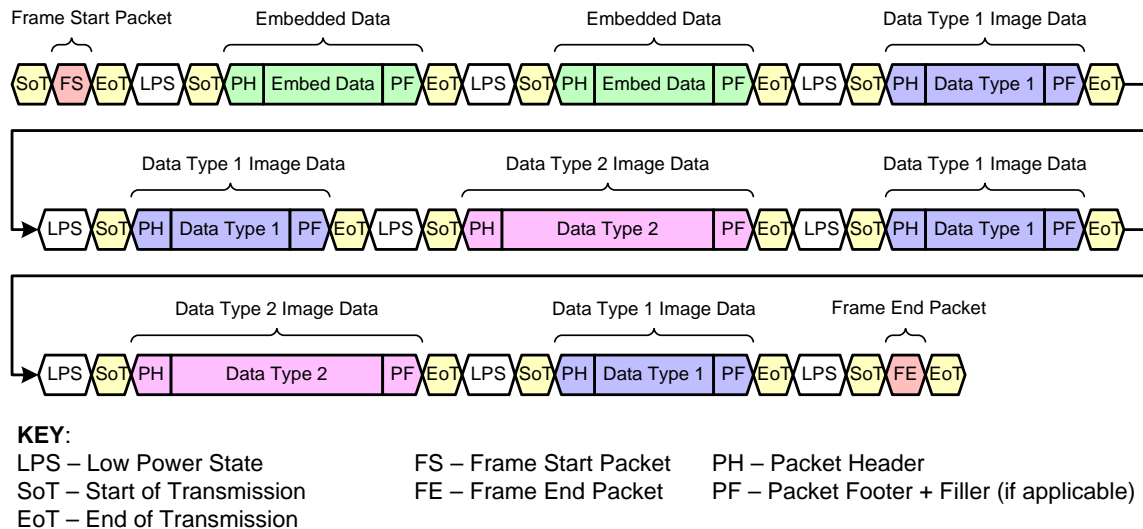


Figure 77 Interleaved Data Transmission using Data Type Value

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,

1174 independent of data type, between a Frame Start and a Frame End packet within the same virtual channel

1175 belong to the same frame.

1176 Packets of different data types may be interleaved at either the packet level as illustrated in *Figure 78* or

1177 the frame level as illustrated in *Figure 79*. Data formats are defined in *Section 11*.

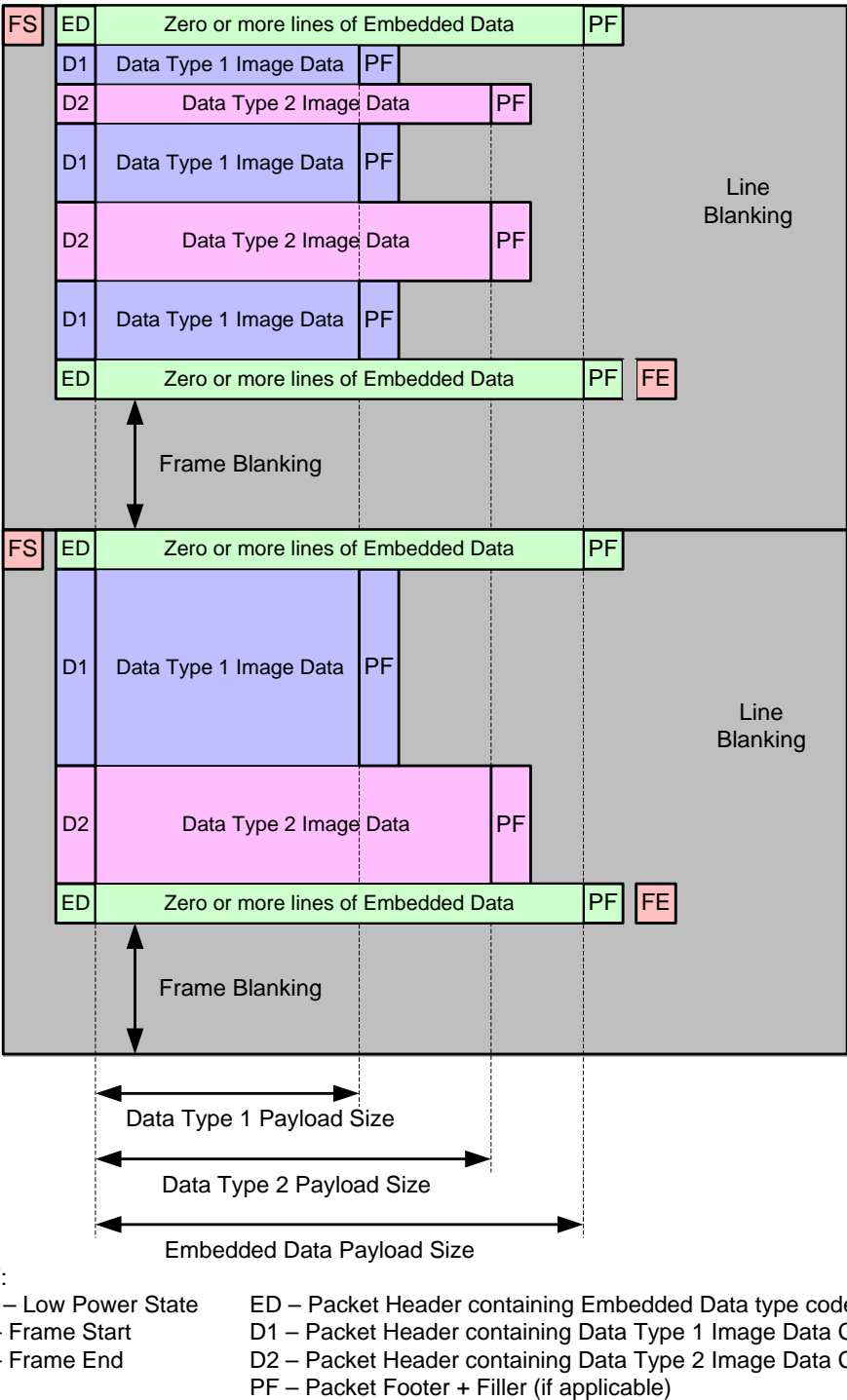
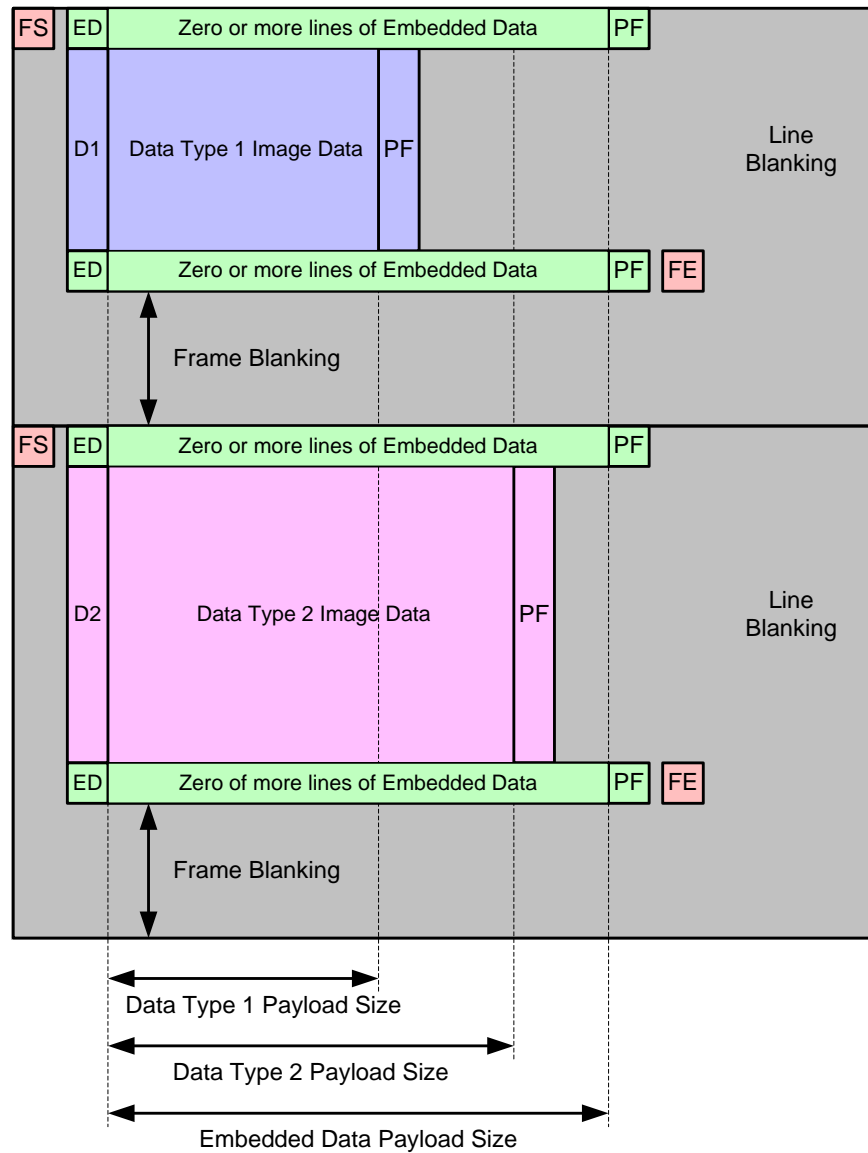


Figure 78 Packet Level Interleaved Data Transmission

**KEY:**

LPS – Low Power State

FS – Frame Start

FE – Frame End

ED – Packet Header containing Embedded Data type code

D1 – Packet Header containing Data Type 1 Image Data Code

D2 – Packet Header containing Data Type 2 Image Data Code

PF – Packet Footer + Filler (if applicable)

Figure 79 Frame Level Interleaved Data Transmission

1179

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 80* 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.

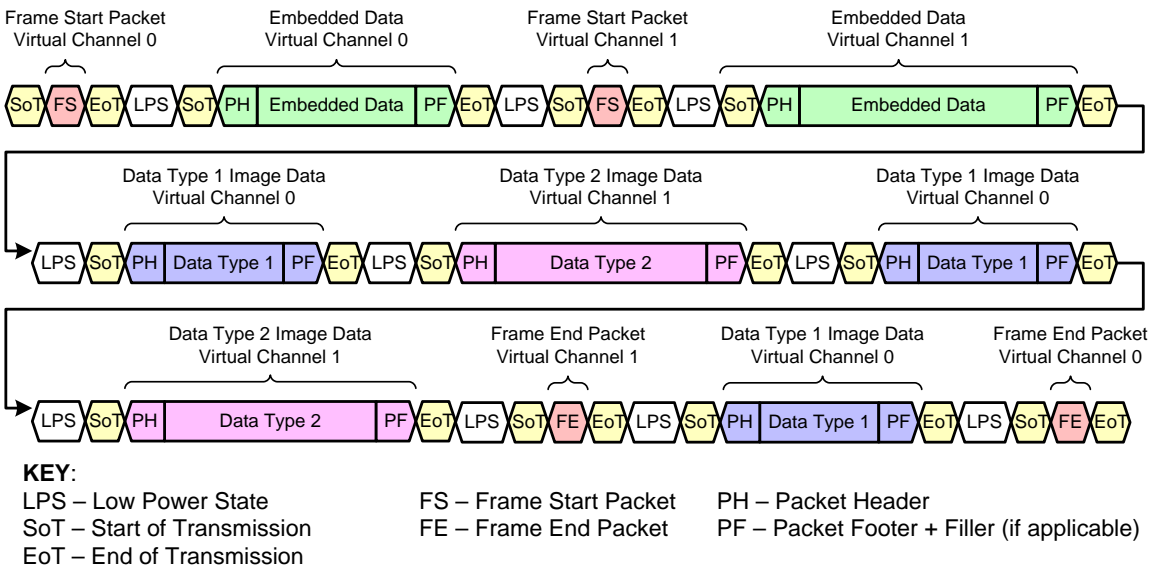


Figure 80 Interleaved Data Transmission using Virtual Channels

This page intentionally left blank

10-Nov-2016

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.

10-Nov-2016

11 Data Formats

The intent of this section is to provide a definitive reference for data formats typically used in CSI-2 applications. **Table 17** 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 17**. 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 17 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 17** 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 81**, 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 81** illustrates these options for a single-Lane system.

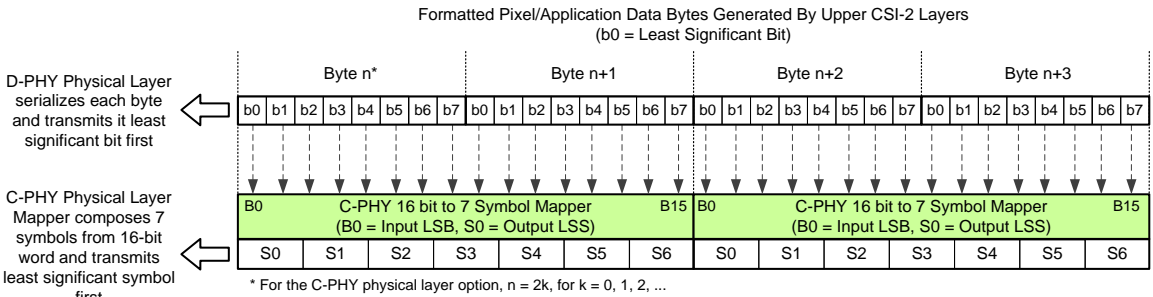


Figure 81 Byte Packing Pixel Data to C-PHY Symbol Illustration

10-Nov-2016

11.1 Generic 8-bit Long Packet Data Types

Table 18 defines the generic 8-bit Long packet data types.

Table 18 Generic 8-bit Long Packet Data Types

Data Type	Description
0x10	Null
0x11	Blanking Data
0x12	Embedded 8-bit non Image Data
0x13	Reserved
0x14	Reserved
0x15	Reserved
0x16	Reserved
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 82**. 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.

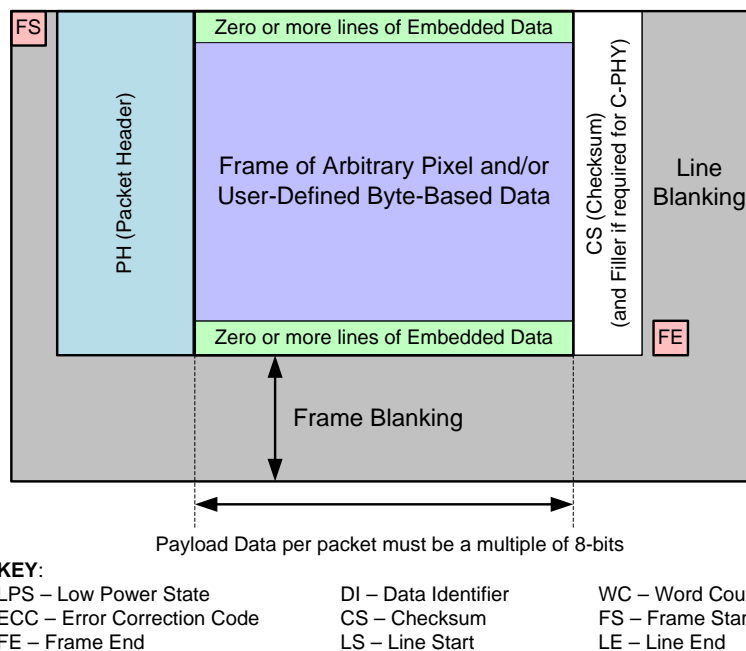


Figure 82 Frame Structure with Embedded Data at the Beginning and End of the Frame

10-Nov-2016

11.2 YUV Image Data

Table 19 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 19 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 83.

Table 20 specifies the packet size constraints for YUV420 8-bit packets. Each packet must be a multiple of the values in the table.

Table 20 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 84.

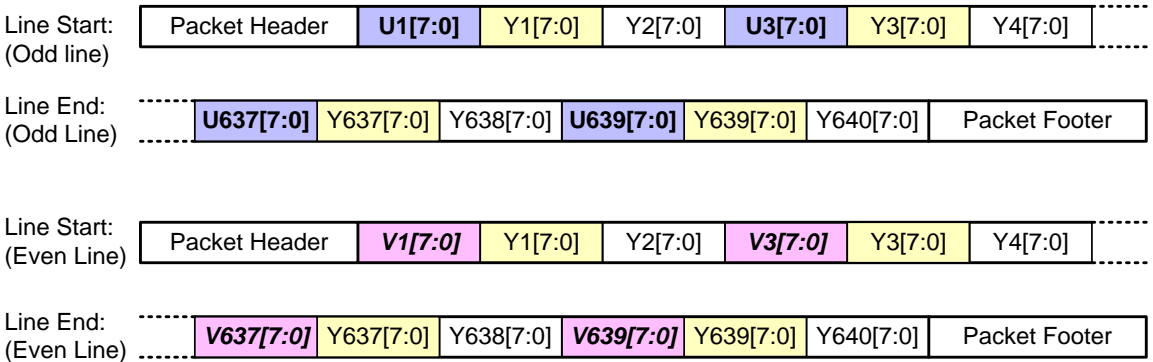


Figure 83 Legacy YUV420 8-bit Transmission

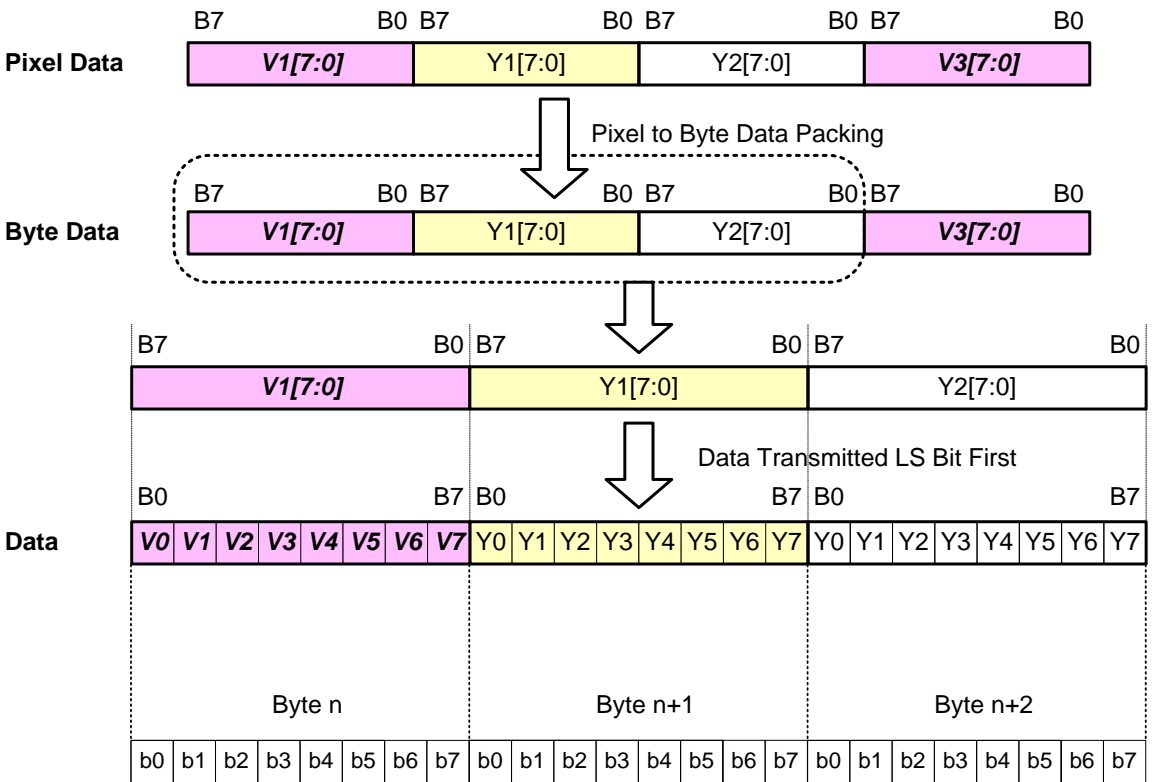


Figure 84 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

- There is one spatial sampling option
- H.261, H.263 and MPEG1 Spatial Sampling (Figure 85).

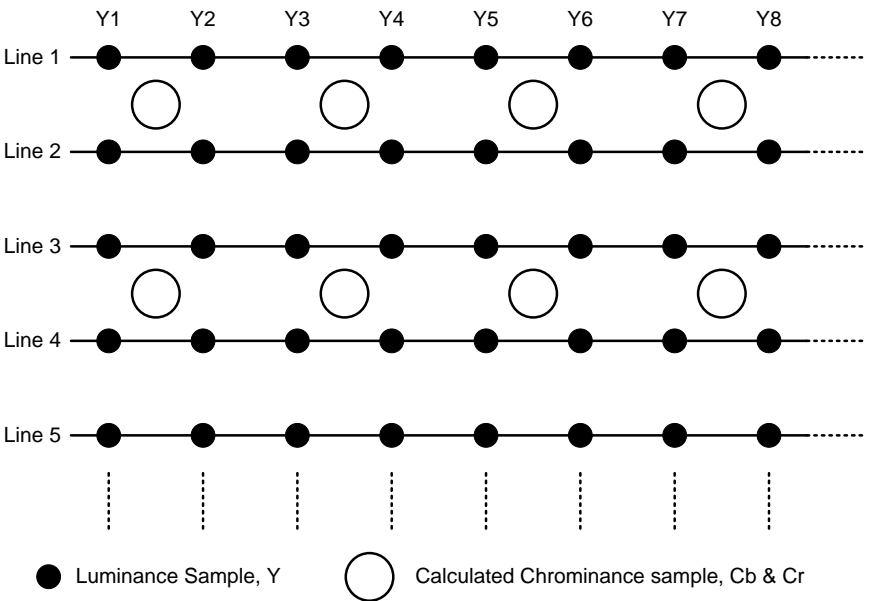


Figure 85 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1

Figure 86 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 87*.

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

Table 21 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 88*.

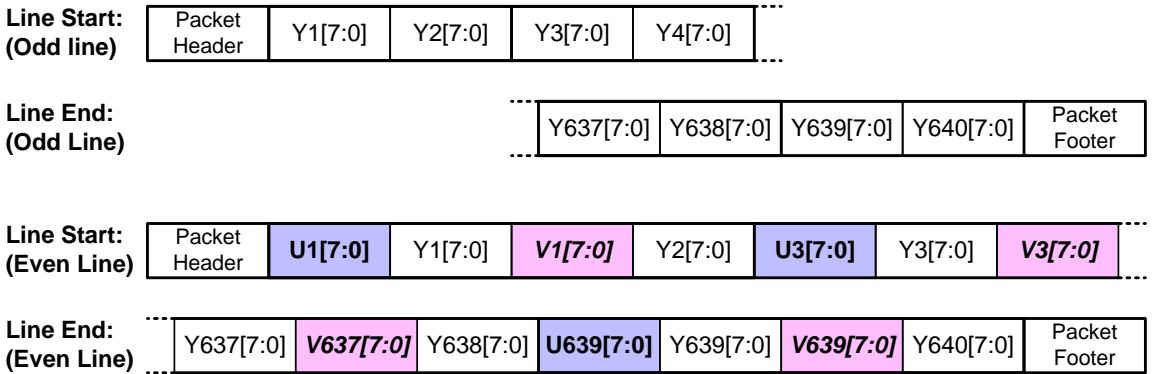
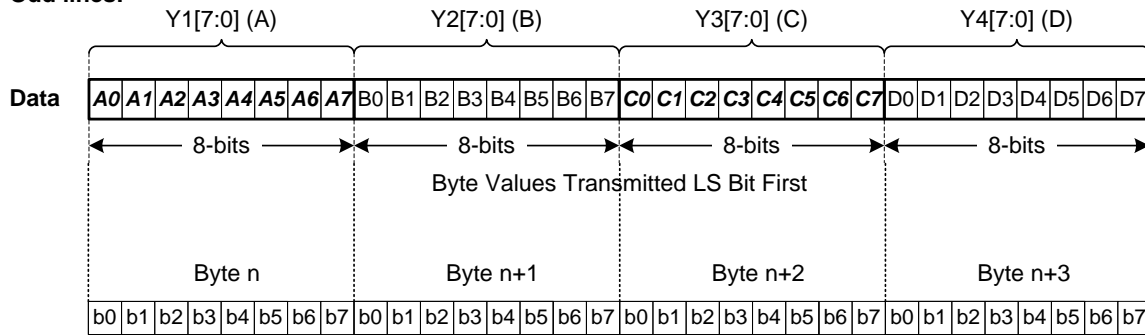


Figure 87 YUV420 8-bit Data Transmission Sequence

Odd lines:



Even lines:

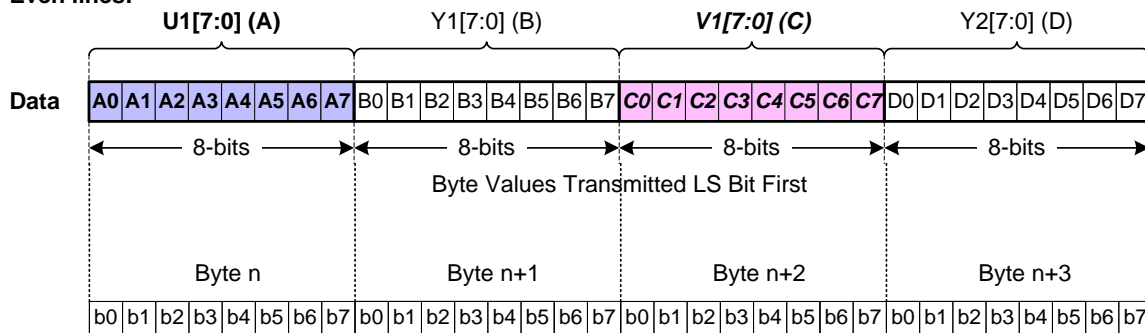


Figure 88 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

There are two spatial sampling options

- H.261, H.263 and MPEG1 Spatial Sampling (*Figure 89*).
- Chroma Shifted Pixel Sampling (CSPS) for MPEG2, MPEG4 (*Figure 90*).

Figure 91 shows the YUV420 frame format.

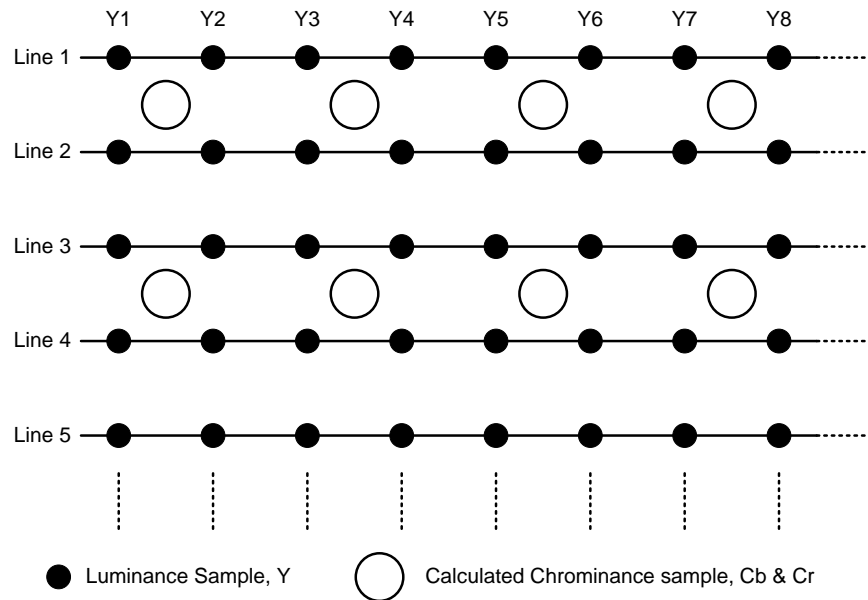


Figure 89 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1

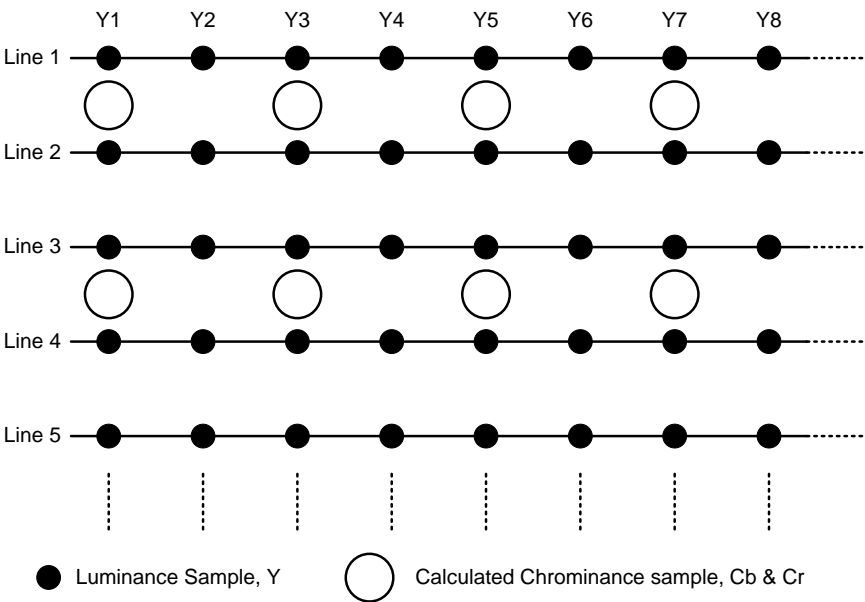


Figure 90 YUV420 Spatial Sampling for MPEG 2 and MPEG 4

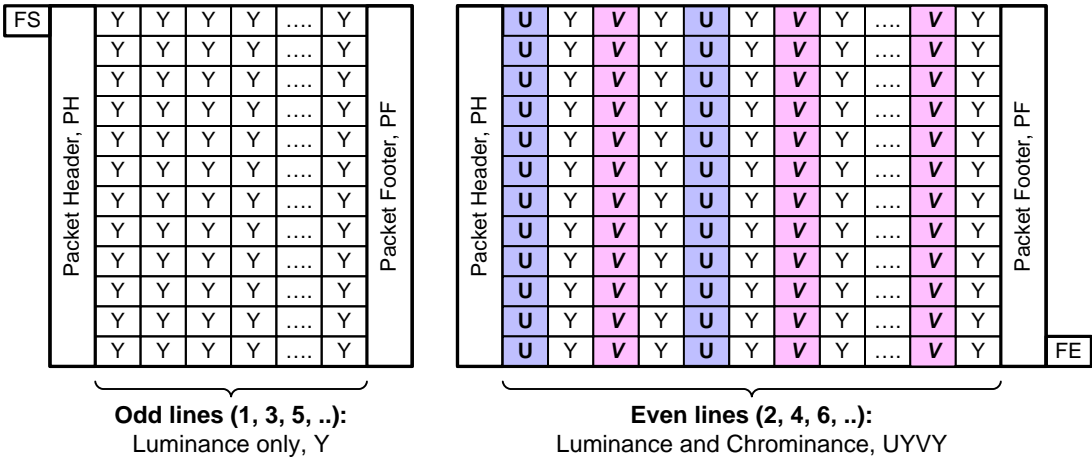


Figure 91 YUV420 8-bit Frame Format

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 92.

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 22 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 22 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 93.

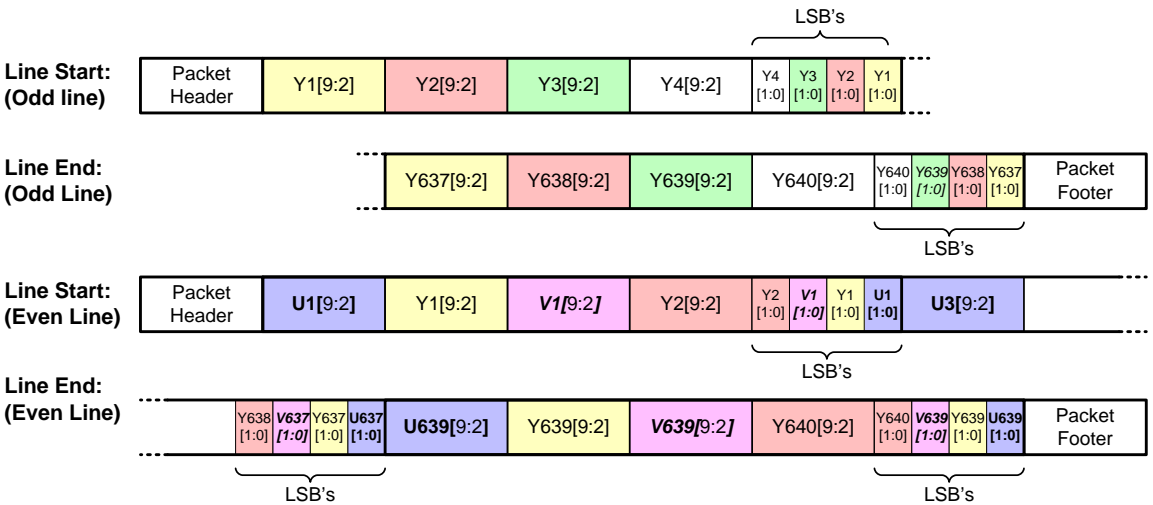


Figure 92 YUV420 10-bit Transmission

10-Nov-2016

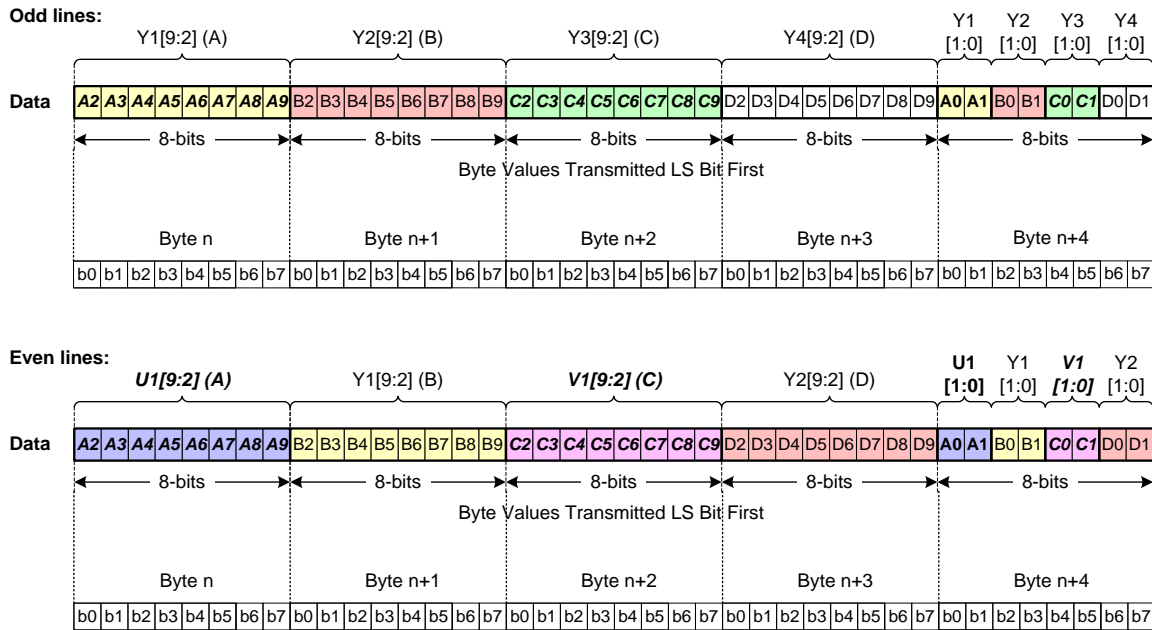


Figure 93 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration

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

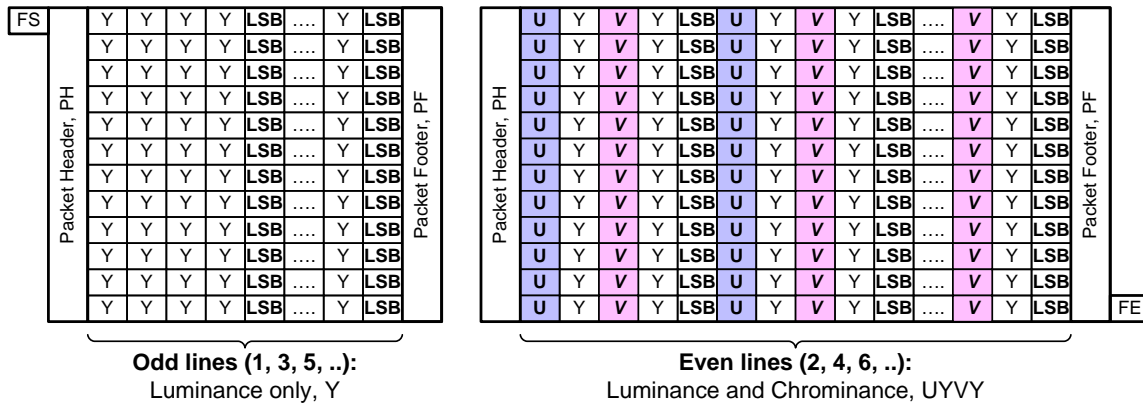


Figure 94 YUV420 10-bit Frame Format

11.2.4 YUV422 8-bit

YUV422 8-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in Figure 95.

Table 23 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 23 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 96.

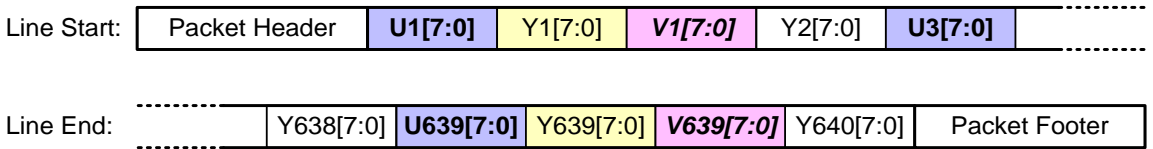


Figure 95 YUV422 8-bit Transmission

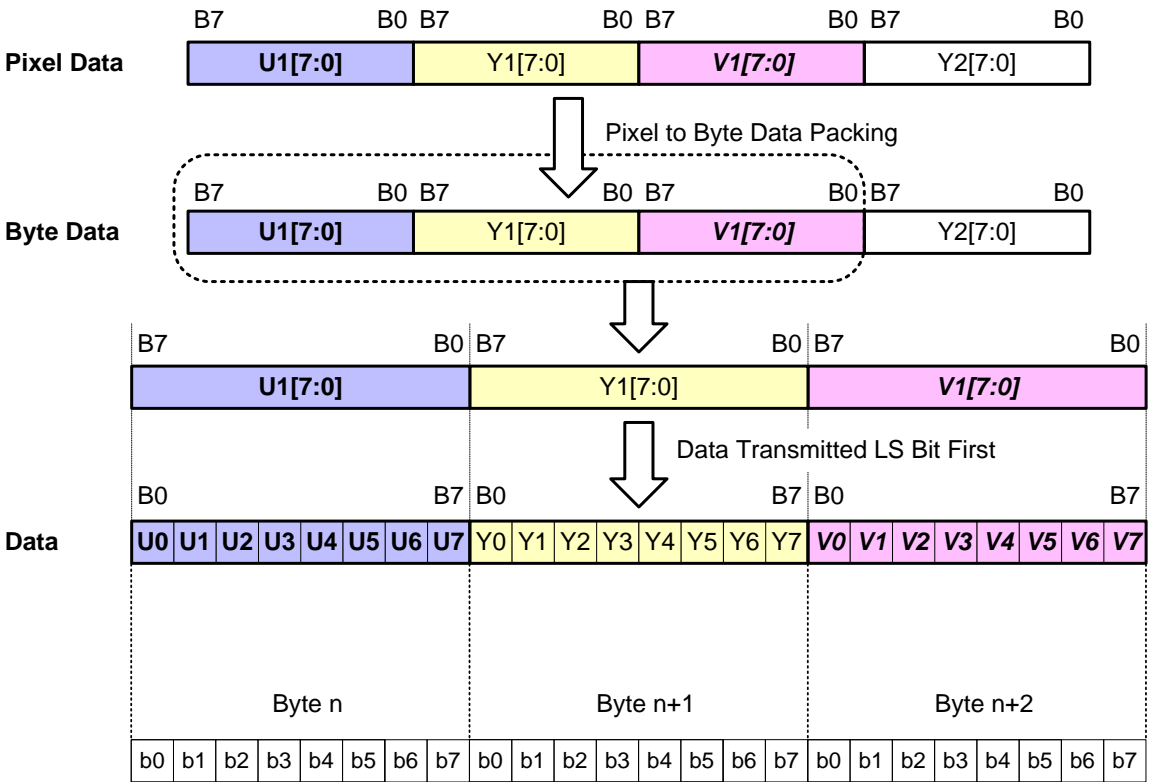


Figure 96 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration

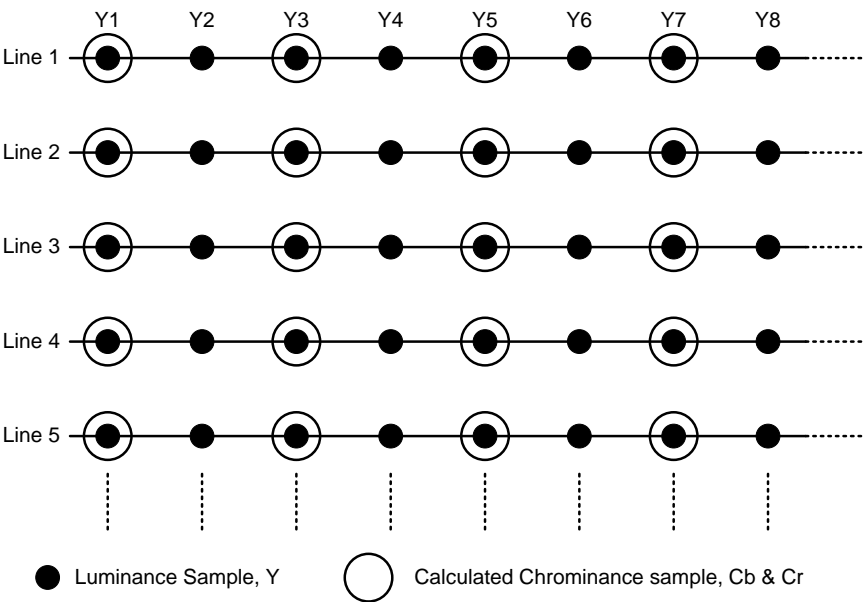


Figure 97 YUV422 Co-sited Spatial Sampling

The pixel spatial alignment is the same as in CCIR-656 standard. The frame format for YUV422 is presented in *Figure 98*.

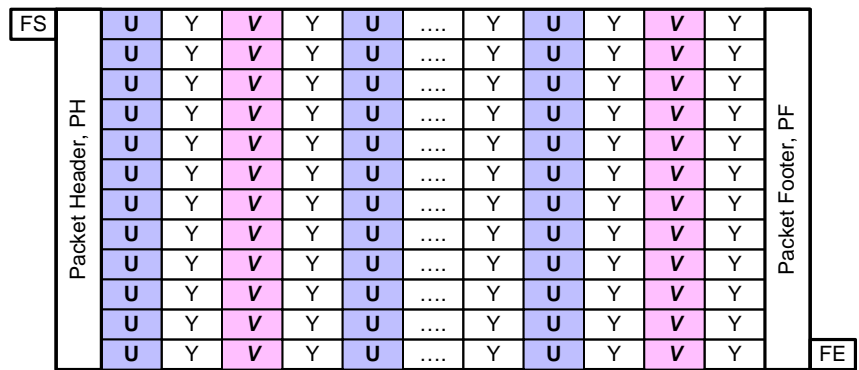


Figure 98 YUV422 8-bit Frame Format

11.2.5 YUV422 10-bit

YUV422 10-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in Figure 99.

Table 24 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 24 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 100.

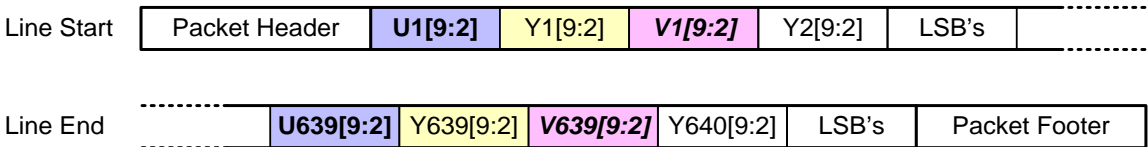


Figure 99 YUV422 10-bit Transmitted Bytes

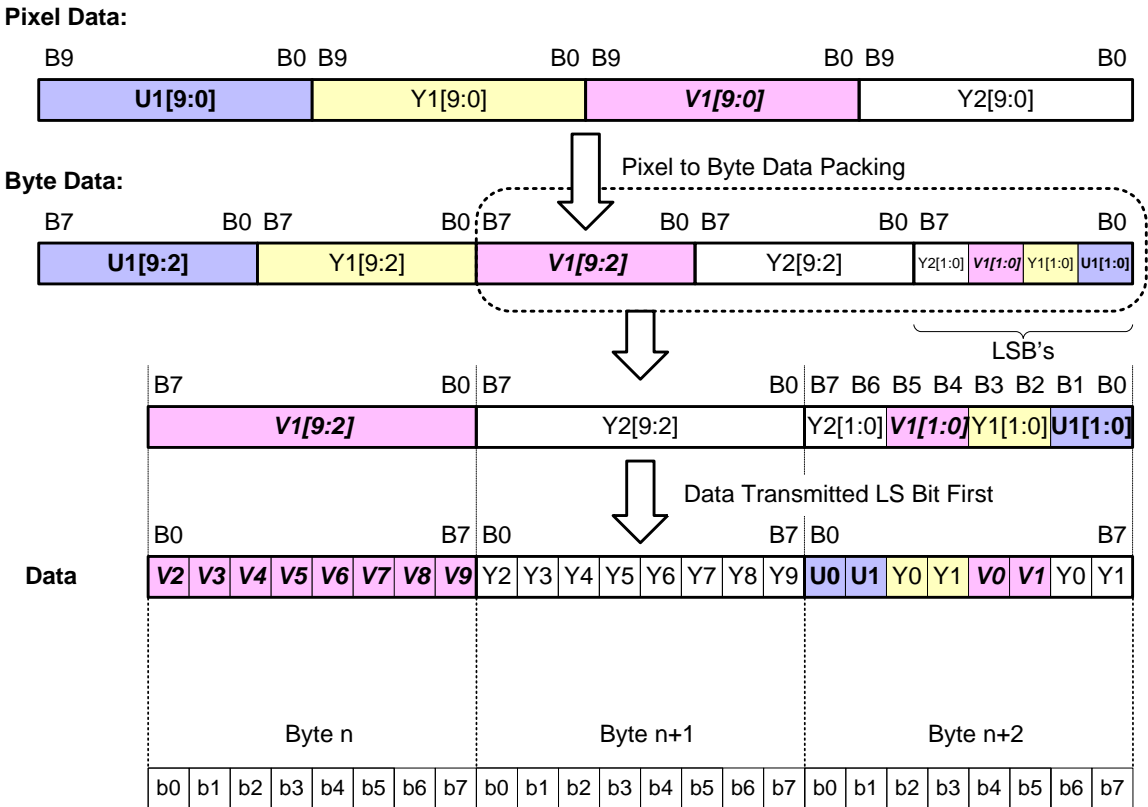


Figure 100 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration

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 101.

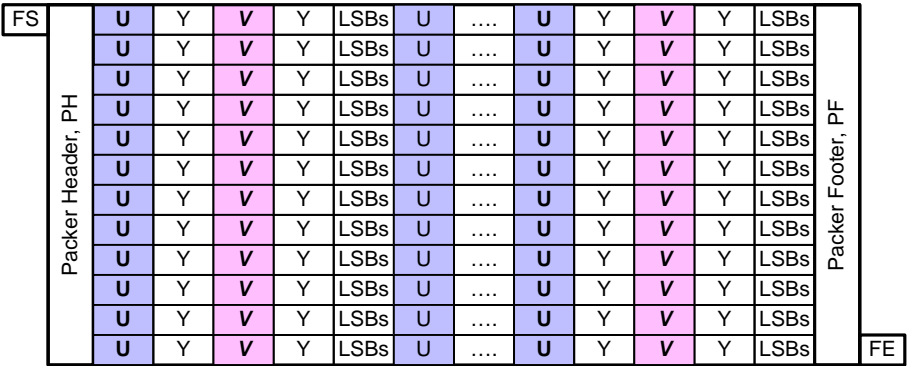


Figure 101 YUV422 10-bit Frame Format

1334

11.3 RGB Image Data

Table 25 defines the data type codes for RGB data formats described in this section.

Table 25 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 102*. The RGB888 frame format is illustrated in *Figure 104*.

Table 26 specifies the packet size constraints for RGB888 packets. The length of each packet must be a multiple of the values in the table.

Table 26 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 103*.

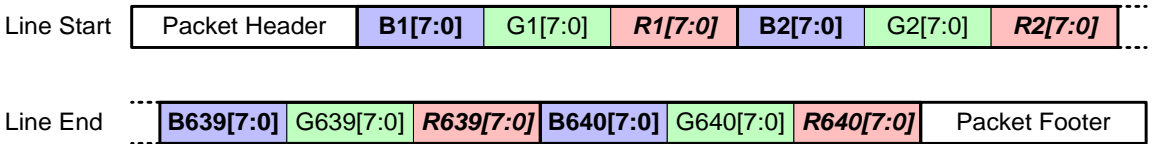


Figure 102 RGB888 Transmission

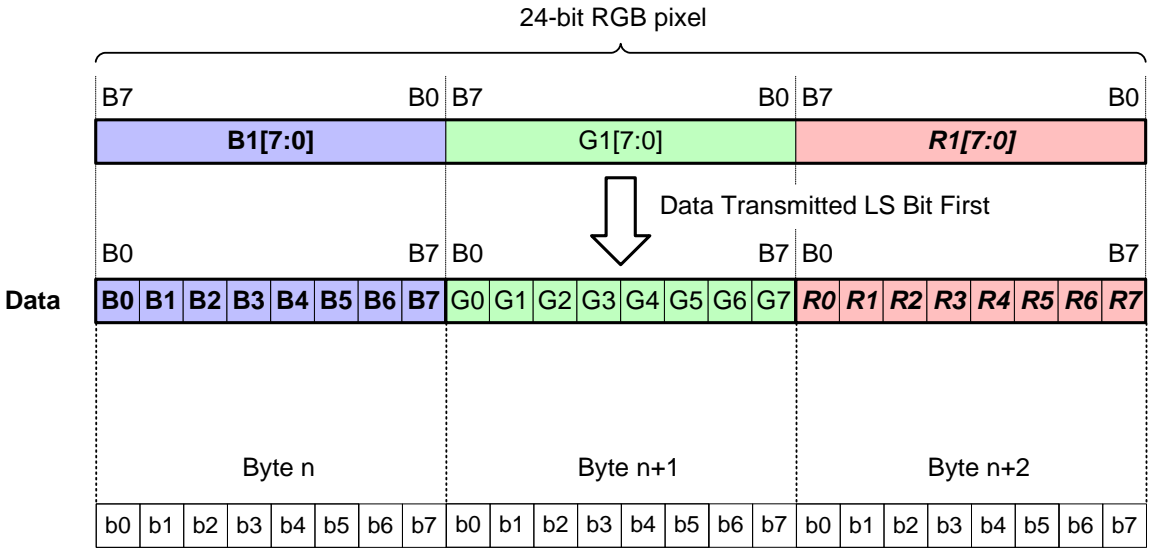


Figure 103 RGB888 Transmission in CSI-2 Bus Bitwise Illustration

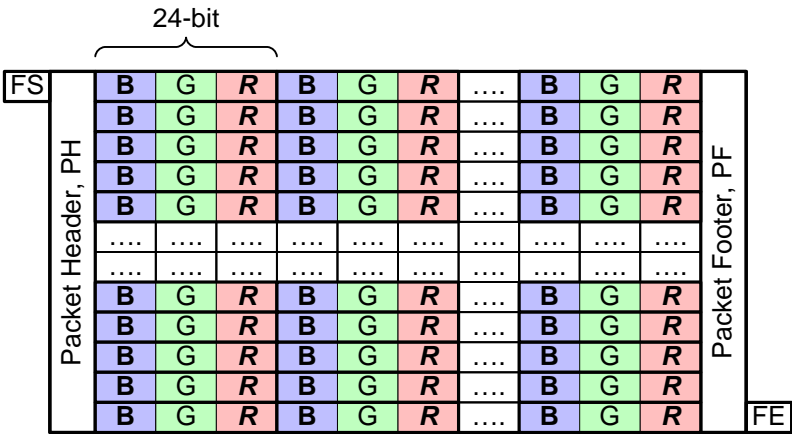


Figure 104 RGB888 Frame Format

1348

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 105*. The frame format for RGB666 is presented in the *Figure 107*. *Table 27* specifies the packet size constraints for RGB666 packets. The length of each packet must be a multiple of the values in the table.

Table 27 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 106*.

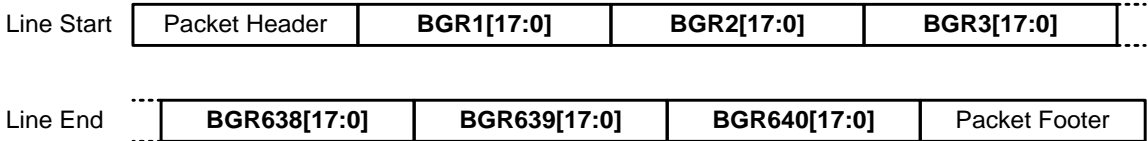


Figure 105 RGB666 Transmission with 18-bit BGR Words

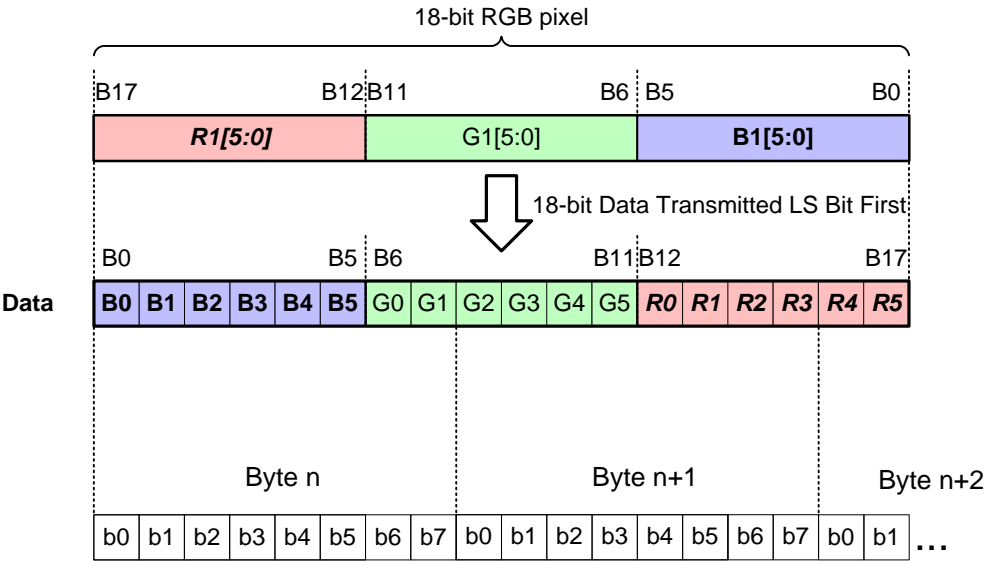


Figure 106 RGB666 Transmission on CSI-2 Bus Bitwise Illustration

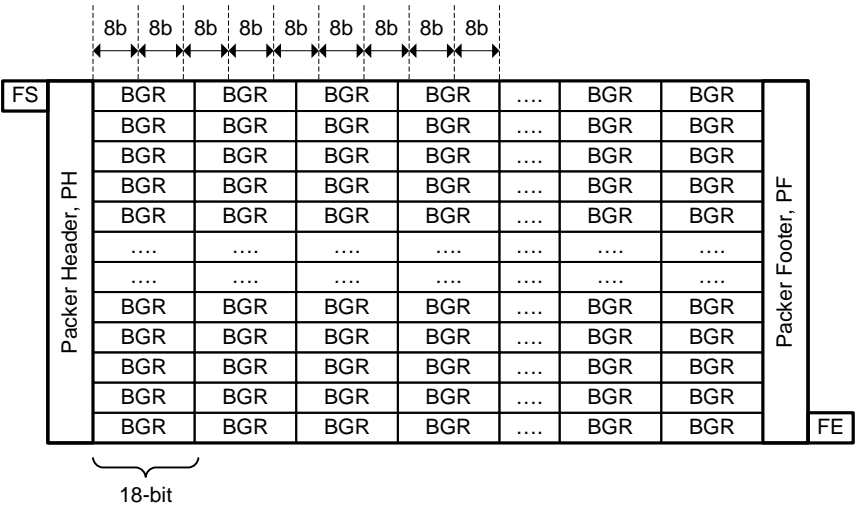


Figure 107 RGB666 Frame Format

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 108*. The frame format for RGB565 is presented in the *Figure 110*. *Table 28* specifies the packet size constraints for RGB565 packets. The length of each packet must be a multiple of the values in the table.

Table 28 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 109*.

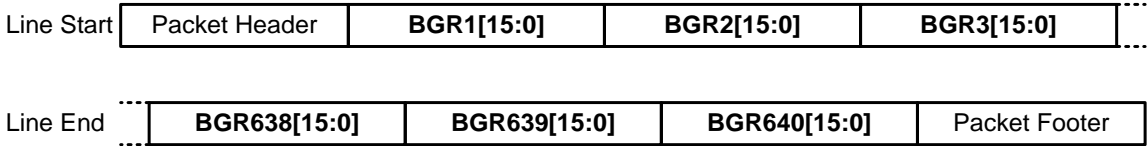


Figure 108 RGB565 Transmission with 16-bit BGR Words

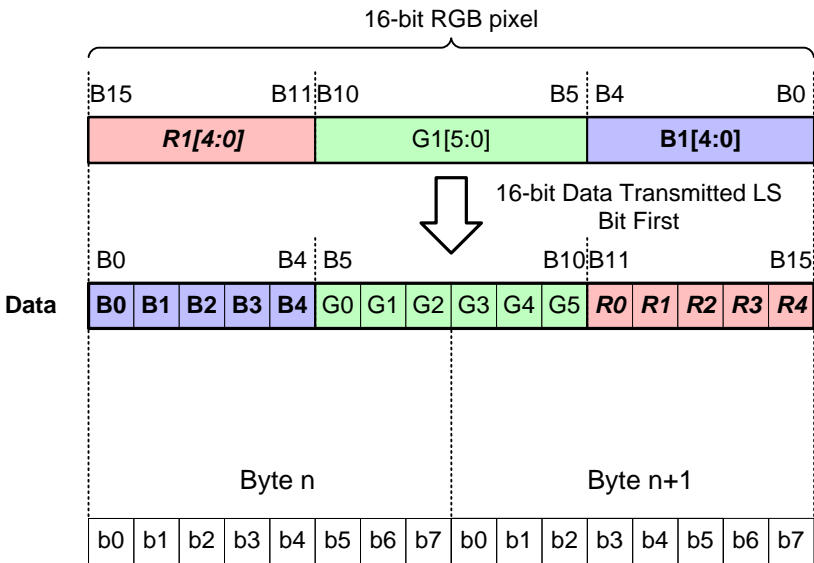


Figure 109 RGB565 Transmission on CSI-2 Bus Bitwise Illustration

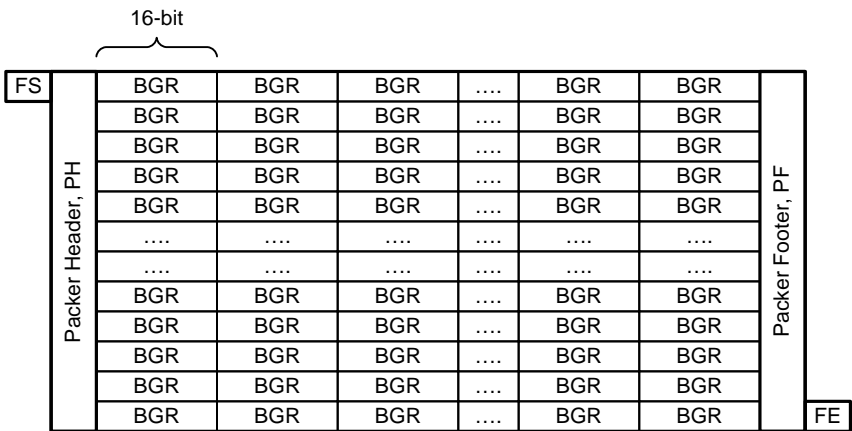


Figure 110 RGB565 Frame Format

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 111*.

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 111*.

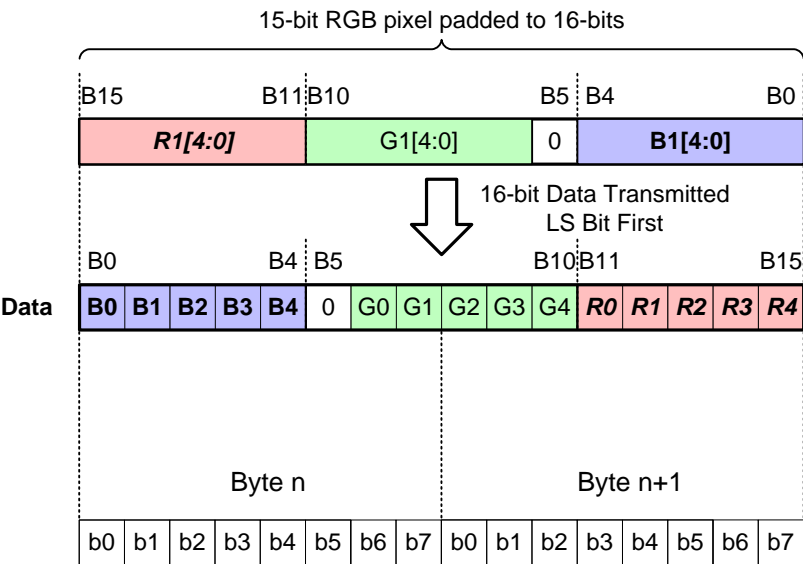


Figure 111 RGB555 Transmission on CSI-2 Bus Bitwise Illustration

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 112*.

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 112*.

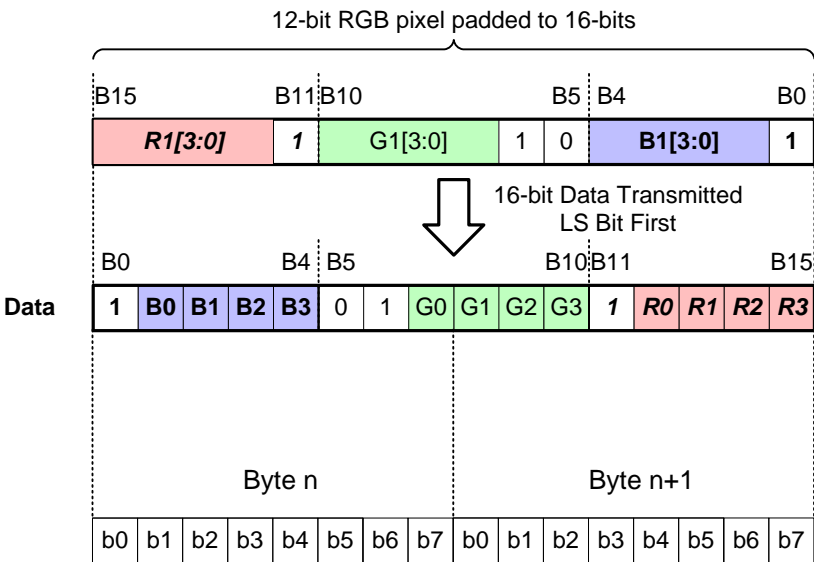


Figure 112 RGB444 Transmission on CSI-2 Bus Bitwise Illustration

10-Nov-2016

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

Table 29 RAW Image Data Types

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

11.4.1 RAW6

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 113** (VGA case). **Table 30** specifies the packet size constraints for RAW6 packets. The length of each packet must be a multiple of the values in the table.

Table 30 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.

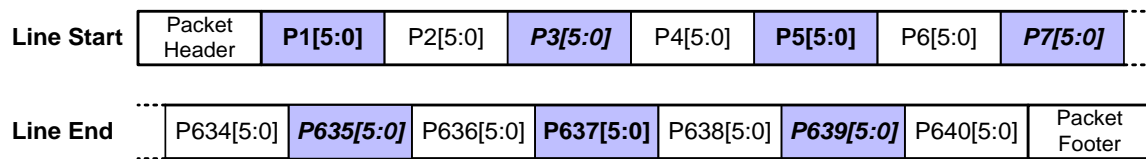


Figure 113 RAW6 Transmission

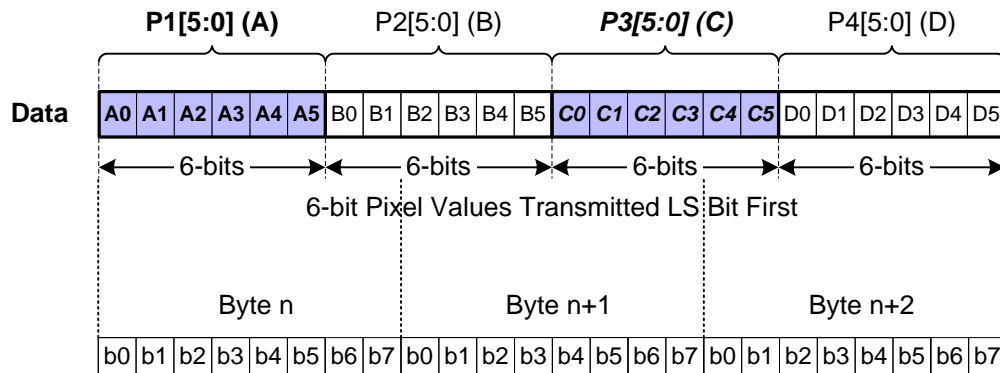


Figure 114 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration

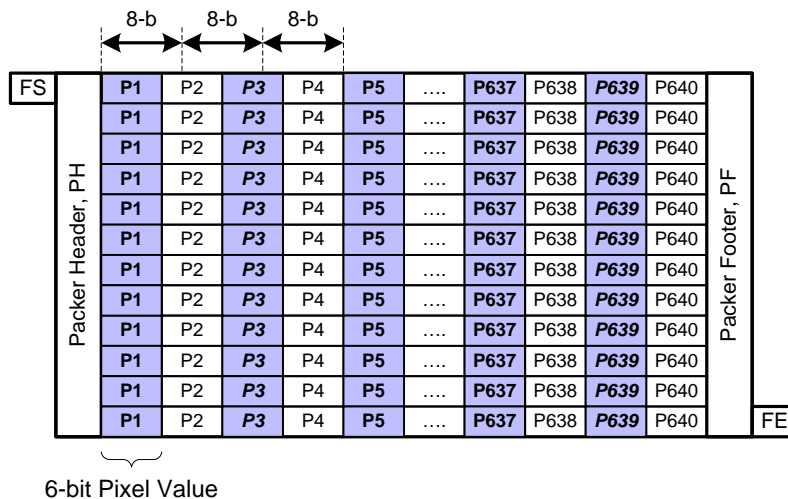


Figure 115 RAW6 Frame Format

10-Nov-2016

11.4.2 RAW7

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 116** (VGA case). Table 31 specifies the packet size constraints for RAW7 packets. The length of each packet must be a multiple of the values in the table.

Table 31 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.

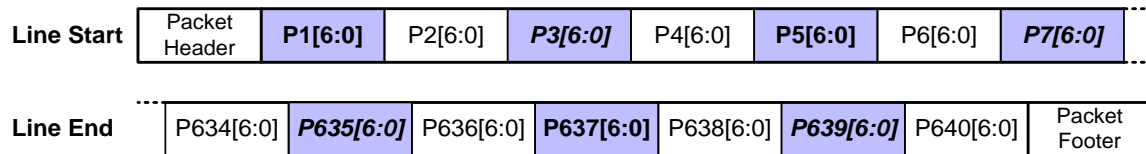


Figure 116 RAW7 Transmission

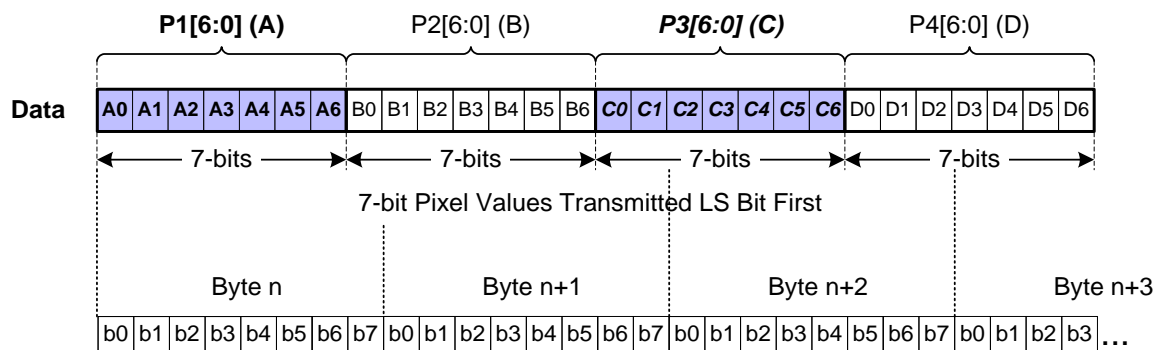


Figure 117 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration

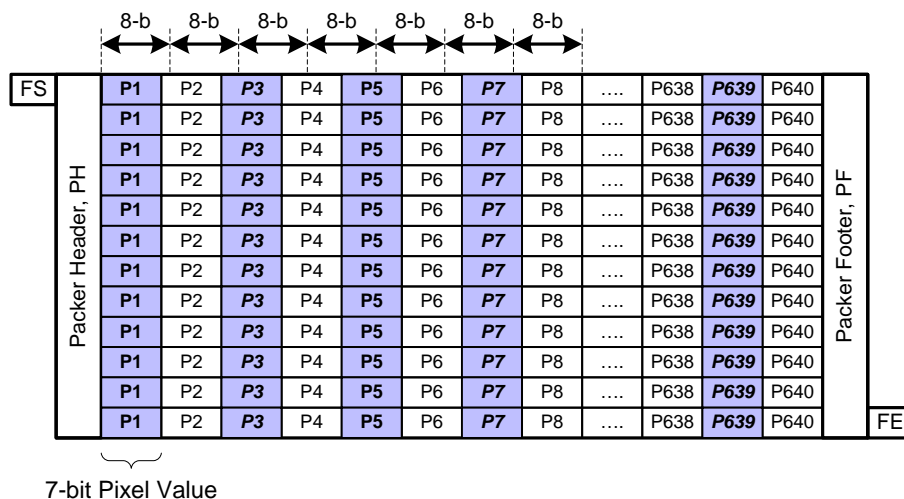


Figure 118 RAW7 Frame Format

11.4.3 RAW8

The 8-bit Raw data transmission is done by transmitting the pixel data over a CSI-2 bus. *Table 32* specifies the packet size constraints for RAW8 packets. The length of each packet must be a multiple of the values in the table.

Table 32 RAW8 Packet Data Size Constraints

Pixels	Bytes	Bits
1	1	8

This sequence is illustrated in *Figure 119* (VGA case).
Bit order in transmission follows the general CSI-2 rule, LSB first.

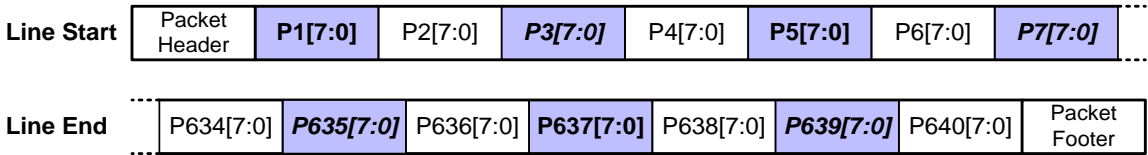


Figure 119 RAW8 Transmission

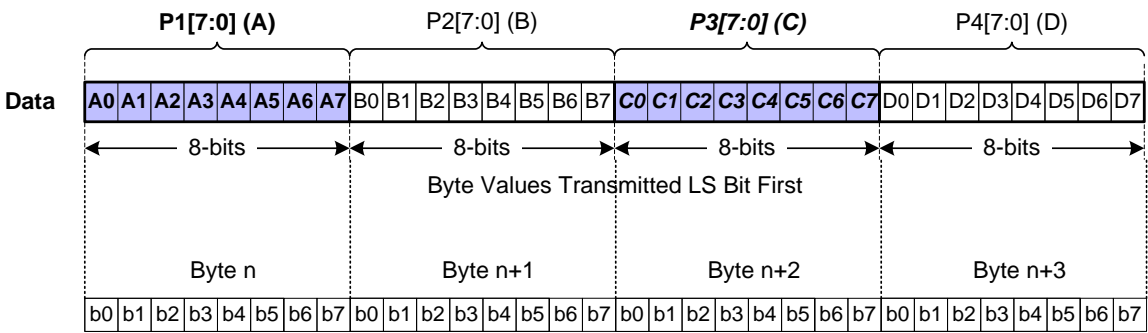


Figure 120 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration

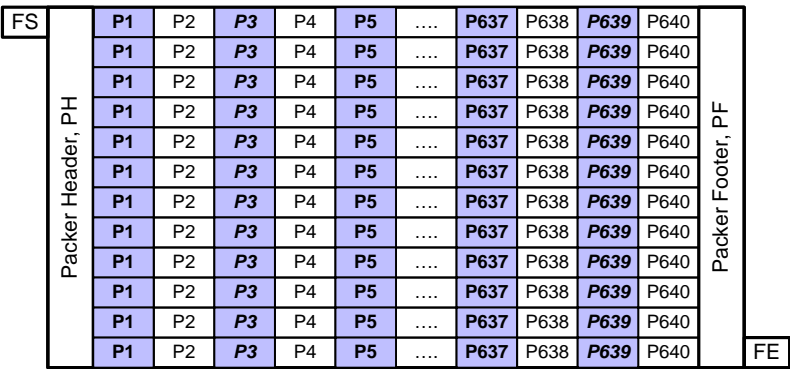


Figure 121 RAW8 Frame Format

10-Nov-2016

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 33** specifies the packet size constraints for RAW10 packets. The length of each packet must be a multiple of the values in the table.

Table 33 RAW10 Packet Data Size Constraints

Pixels	Bytes	Bits
4	5	40

This sequence is illustrated in **Figure 122** (VGA case).

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

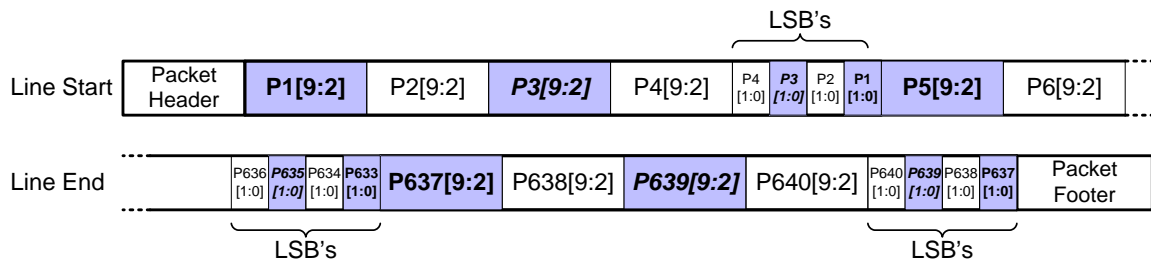


Figure 122 RAW10 Transmission

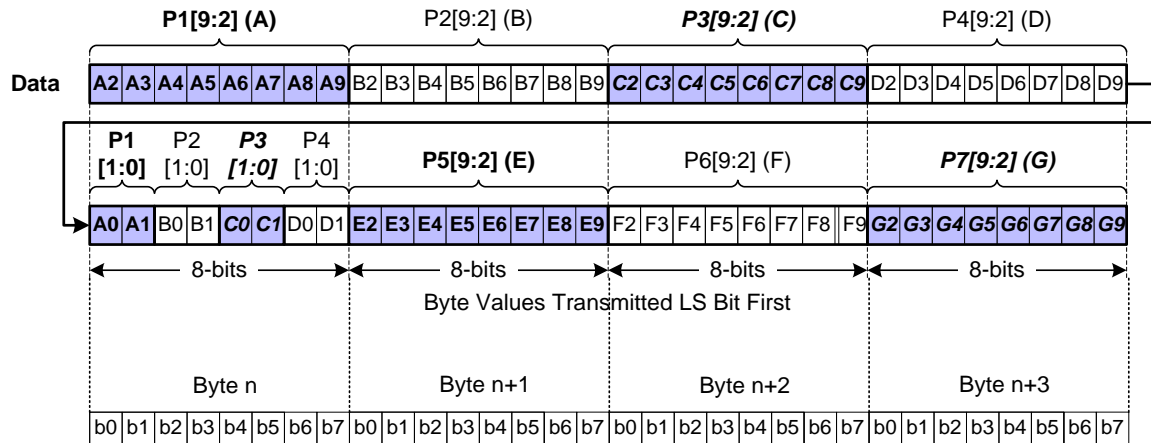


Figure 123 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration

10-Nov-2016

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 34** specifies the packet size constraints for RAW12 packets. The length of each packet must be a multiple of the values in the table.

Table 34 RAW12 Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

This sequence is illustrated in **Figure 125** (VGA case).

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

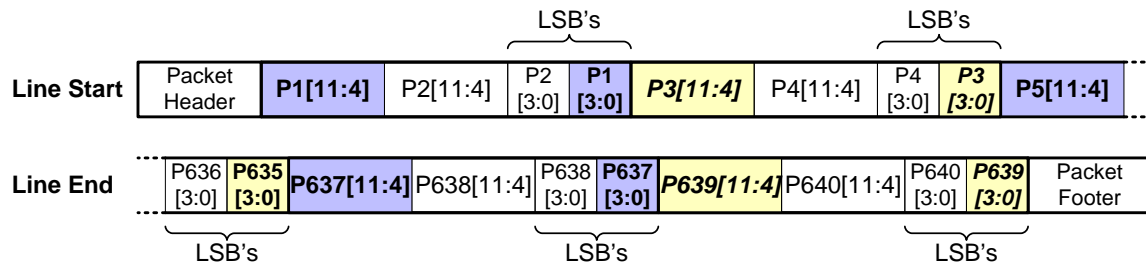


Figure 125 RAW12 Transmission

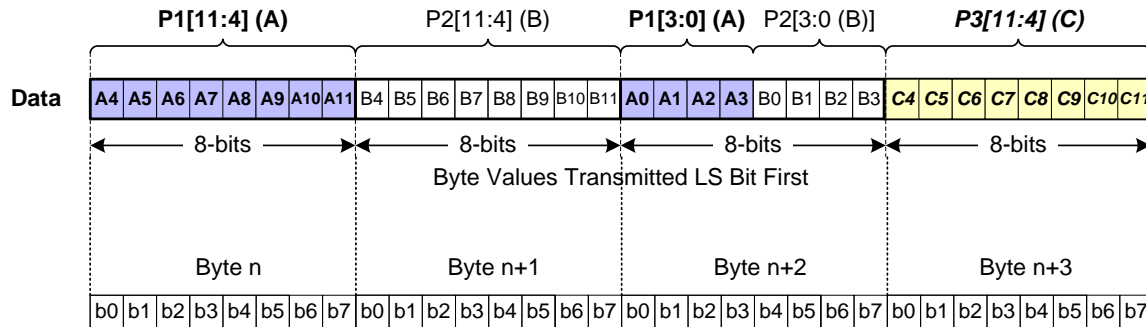


Figure 126 RAW12 Transmission on CSI-2 Bus Bitwise Illustration

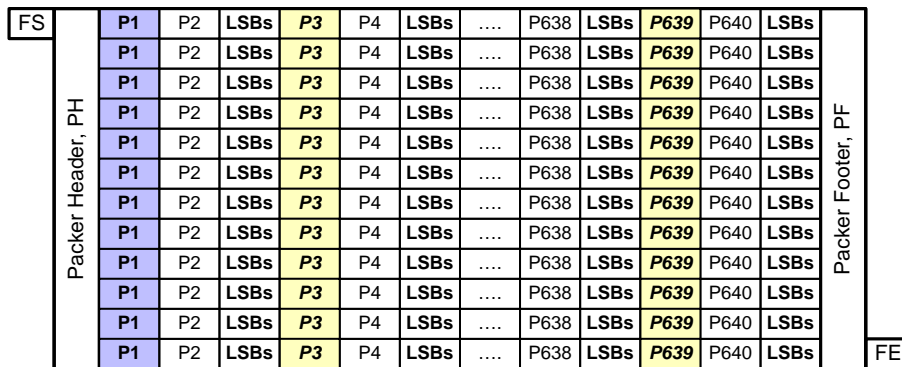


Figure 127 RAW12 Frame Format

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 35** specifies the packet size constraints for RAW14 packets. The length of each packet must be a multiple of the values in the table.

Table 35 RAW14 Packet Data Size Constraints

Pixels	Bytes	Bits
4	7	56

The sequence is illustrated in **Figure 128** (VGA case).
The LS bits for P1, P2, P3 and P4 are distributed in three bytes as shown in **Figure 129**. The same is true for the LS bits for P637, P638, P639 and P640. The bit order during transmission follows the general CSI-2 rule, i.e. LSB first.

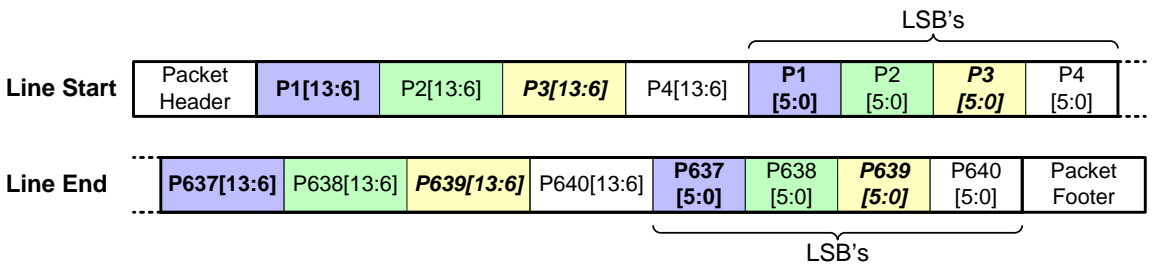


Figure 128 RAW14 Transmission

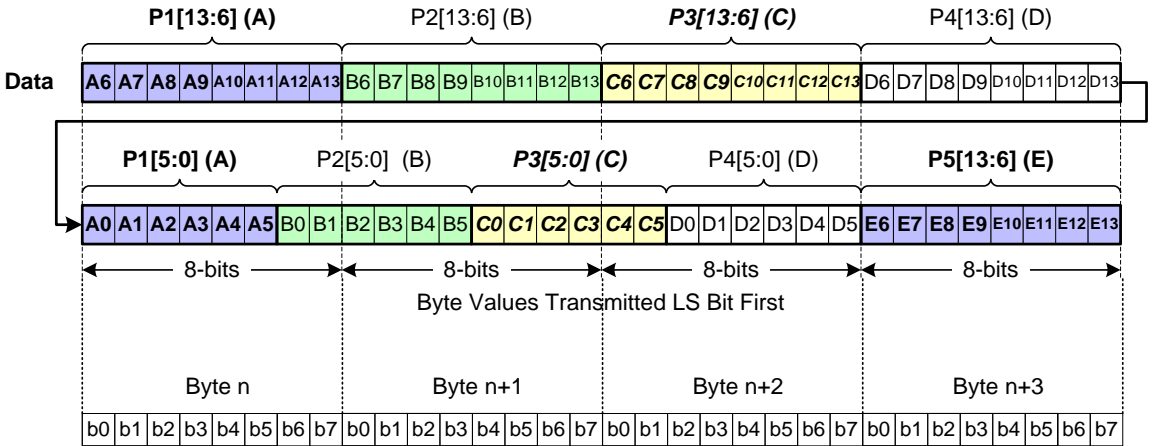


Figure 129 RAW14 Transmission on CSI-2 Bus Bitwise Illustration

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 36** specifies the packet size constraints for RAW16 packets. The length of each packet must be a multiple of the values in the table.

Table 36 RAW16 Packet Data Size Constraints

Pixels	Bytes	Bits
1	2	16

This sequence is illustrated in **Figure 131** (VGA case).
Bit order in transmission follows the general CSI-2 rule: LSB first.

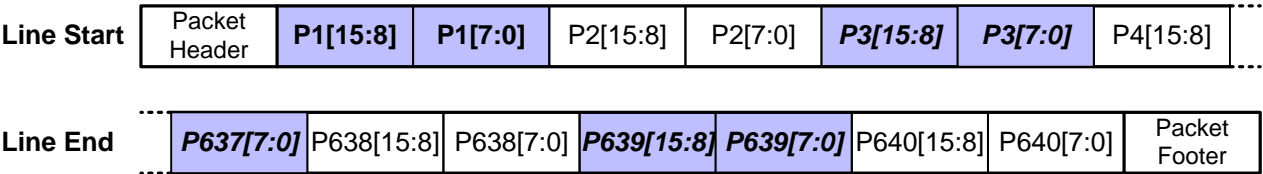


Figure 131 RAW16 Transmission

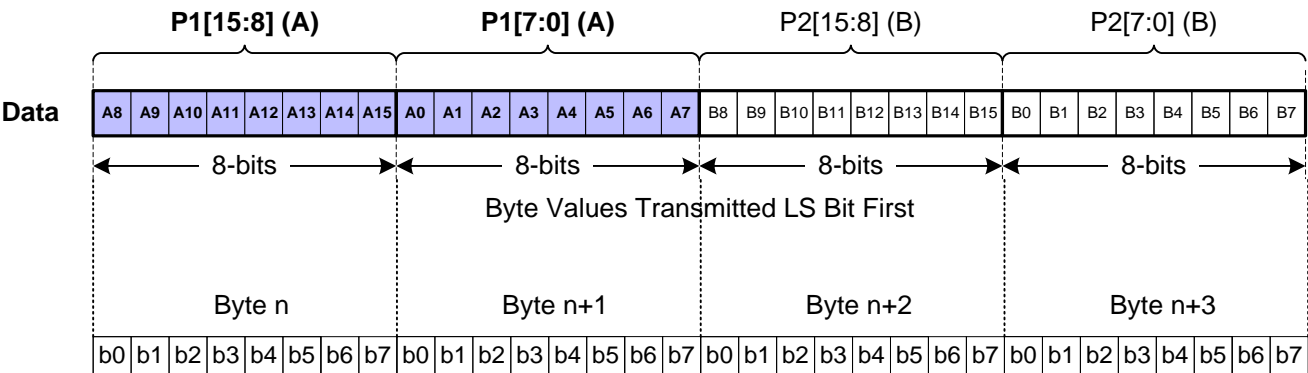


Figure 132 RAW16 Transmission on CSI-2 Bus Bitwise Illustration

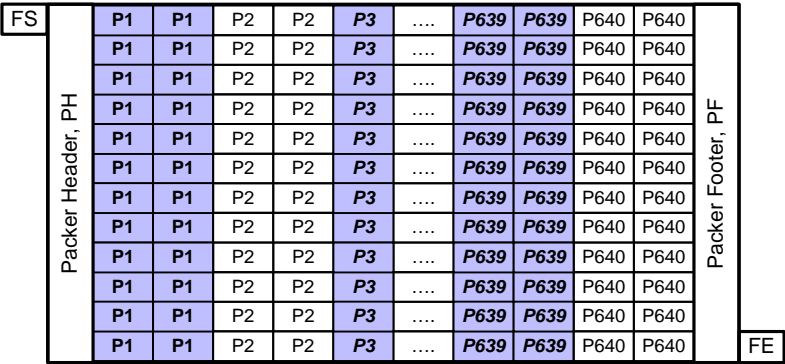


Figure 133 RAW16 Frame Format

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 37 specifies the packet size constraints for RAW20 packets. The length of each packet must be a multiple of the values in the table.

Table 37 RAW20 Packet Data Size Constraints

Pixels	Bytes	Bits
2	5	40

This sequence is illustrated in Figure 134 (VGA case).
Bit order in transmission follows the general CSI-2 rule: LSB first.

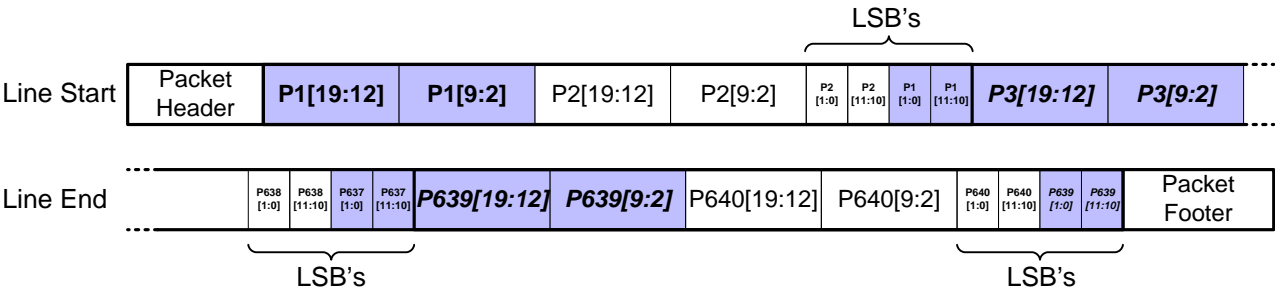


Figure 134 RAW20 Transmission

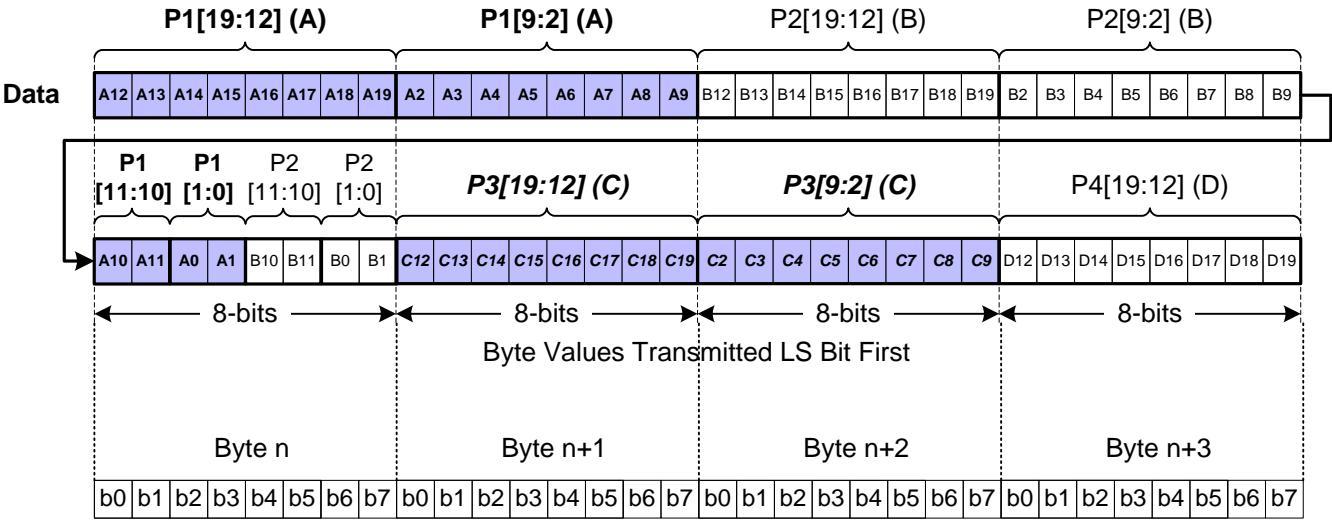


Figure 135 RAW20 Transmission on CSI-2 Bus Bitwise Illustration

1485

Figure 136 RAW20 Frame Format

10-Nov-2016

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.

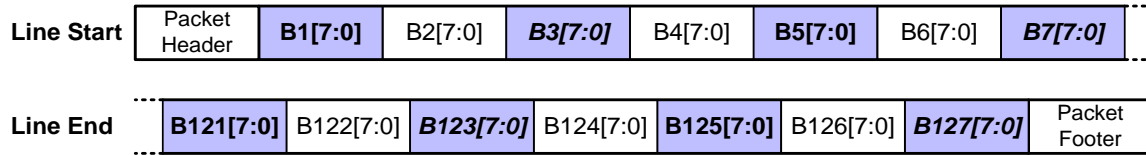


Figure 137 User Defined 8-bit Data (128 Byte Packet)

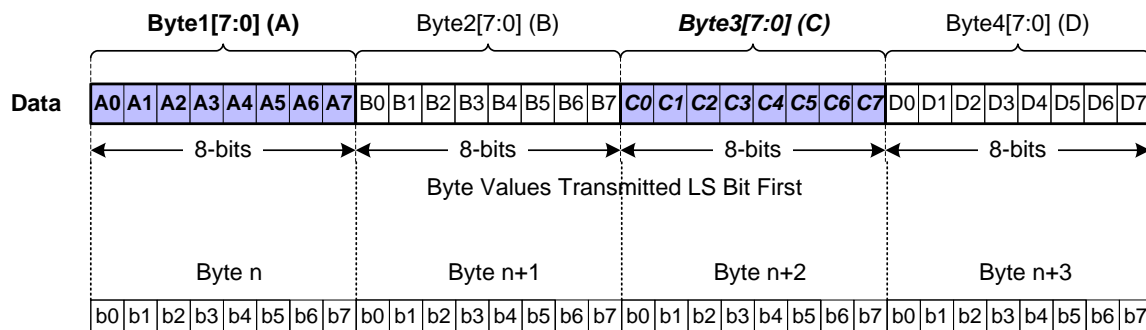


Figure 138 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration

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.

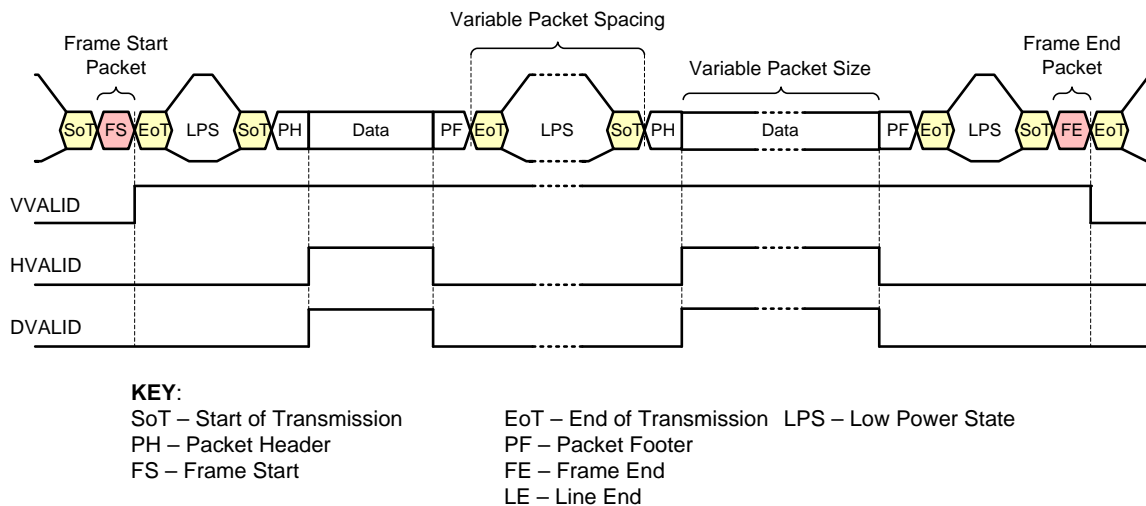


Figure 139 Transmission of User Defined 8-bit Data

1499 Eight different User Defined data type codes are available as shown in *Table 38*.

1500

Table 38 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

10-Nov-2016

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

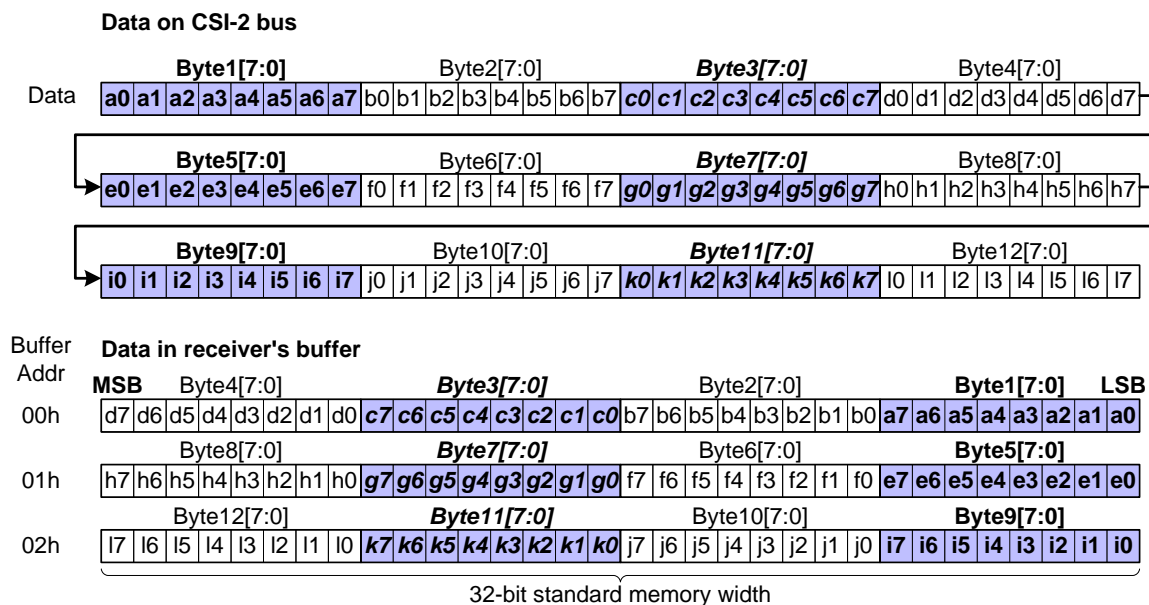


Figure 140 General/Arbitrary Data Reception

12.2 RGB888 Data Reception

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

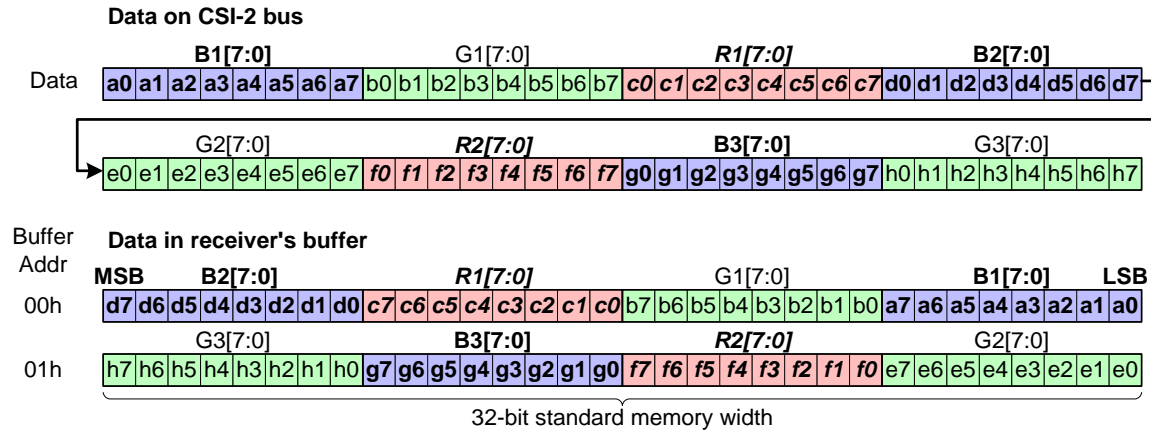


Figure 141 RGB888 Data Format Reception

12.3 RGB666 Data Reception

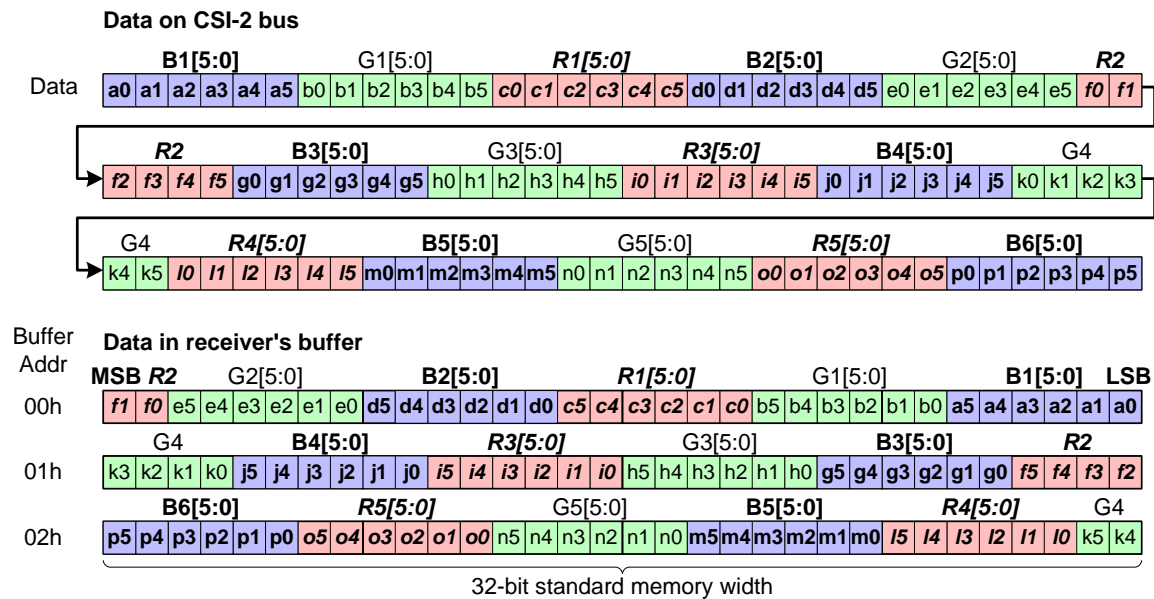


Figure 142 RGB666 Data Format Reception

10-Nov-2016

12.4 RGB565 Data Reception

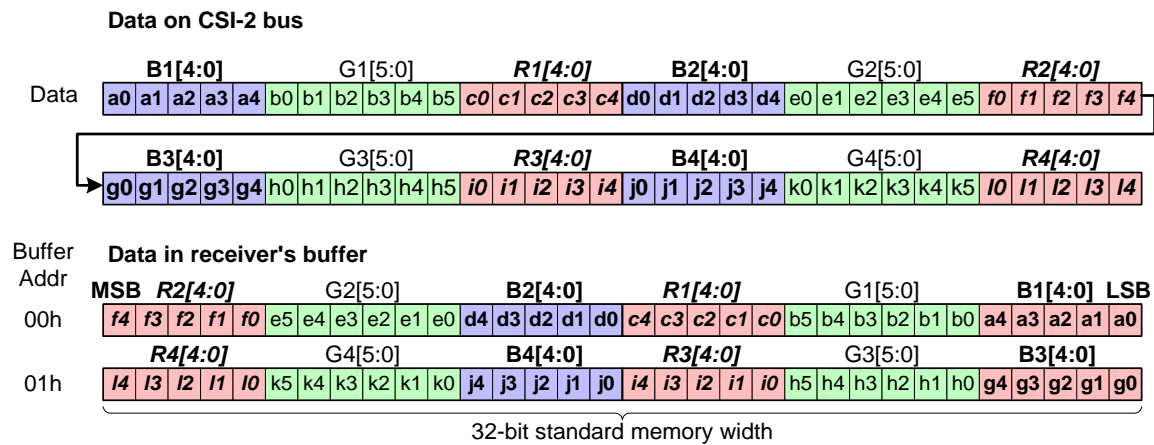


Figure 143 RGB565 Data Format Reception

12.5 RGB555 Data Reception

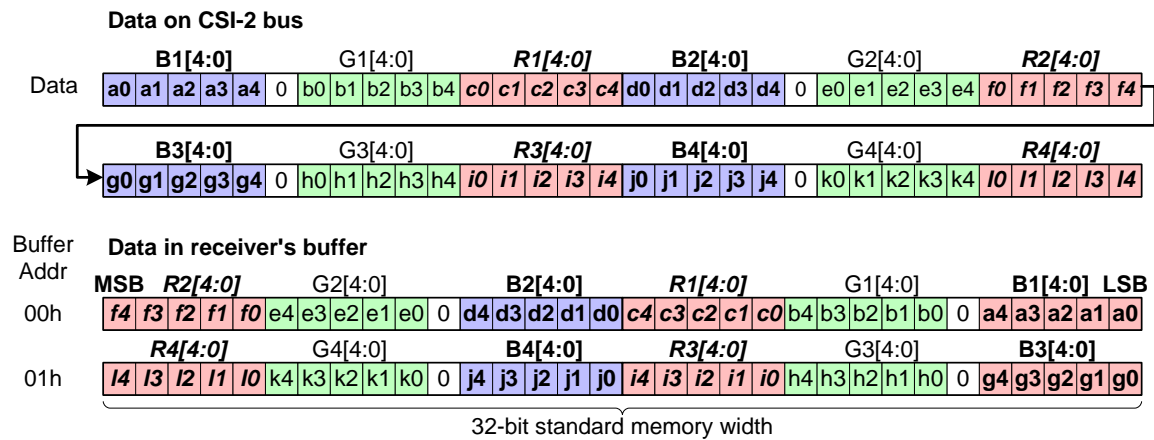


Figure 144 RGB555 Data Format Reception

12.6 RGB444 Data Reception

The RGB444 data format byte to 32-bit memory word mapping has a special transform as shown in **Figure 145**.

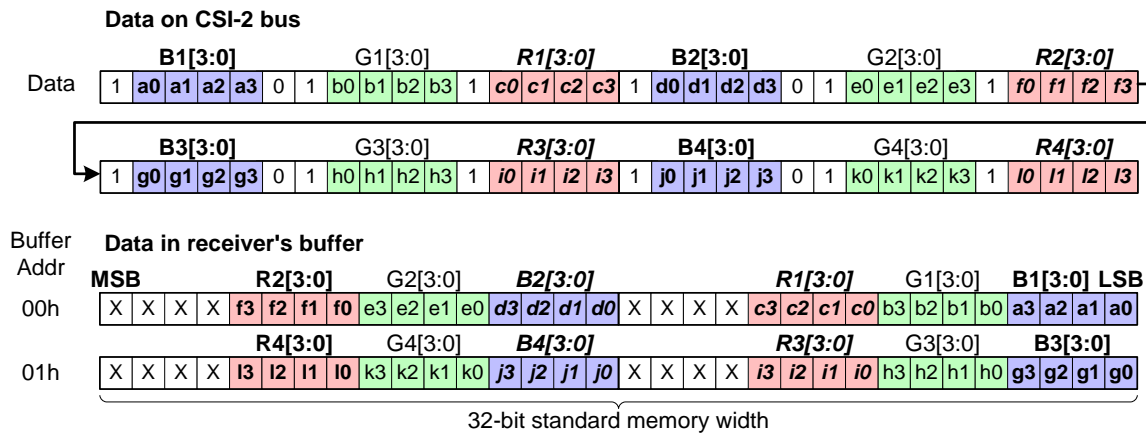


Figure 145 RGB444 Data Format Reception

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.

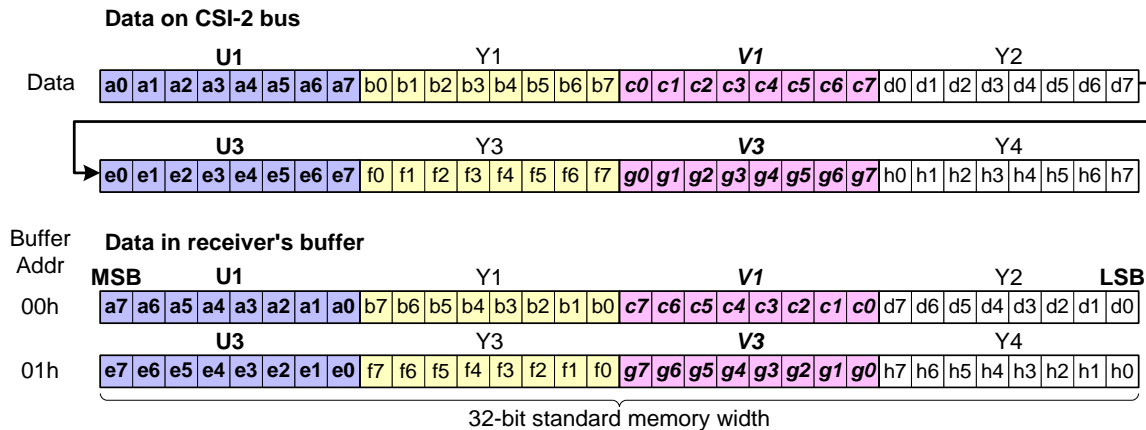


Figure 146 YUV422 8-bit Data Format Reception

10-Nov-2016

12.8 YUV422 10-bit Data Reception

The YUV422 10-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

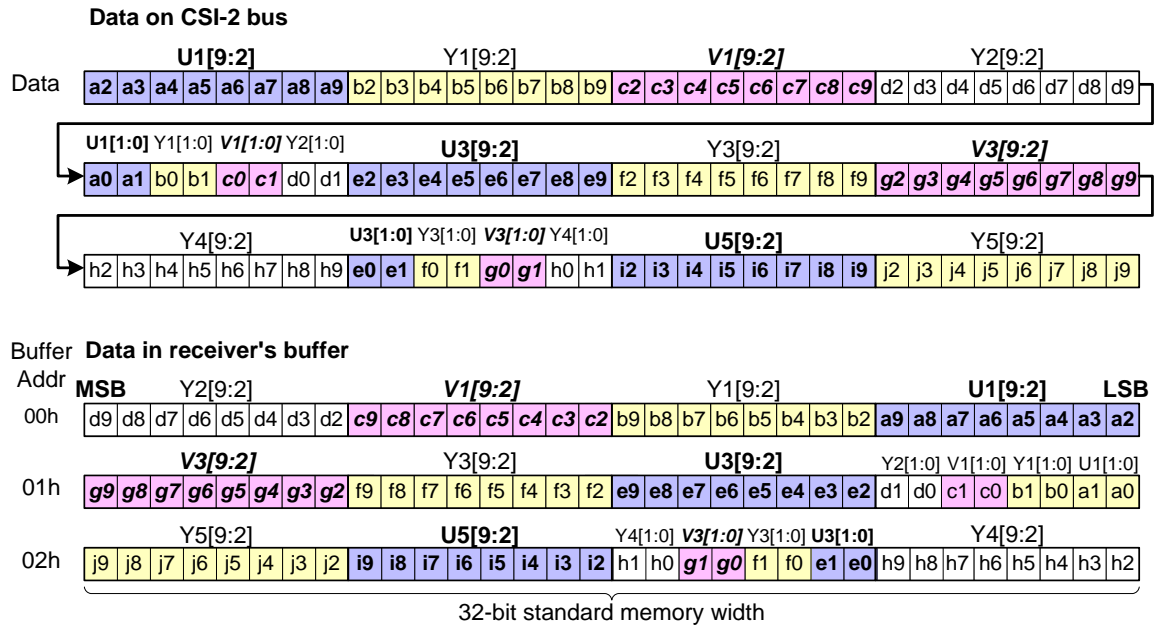


Figure 147 YUV422 10-bit Data Format Reception

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.

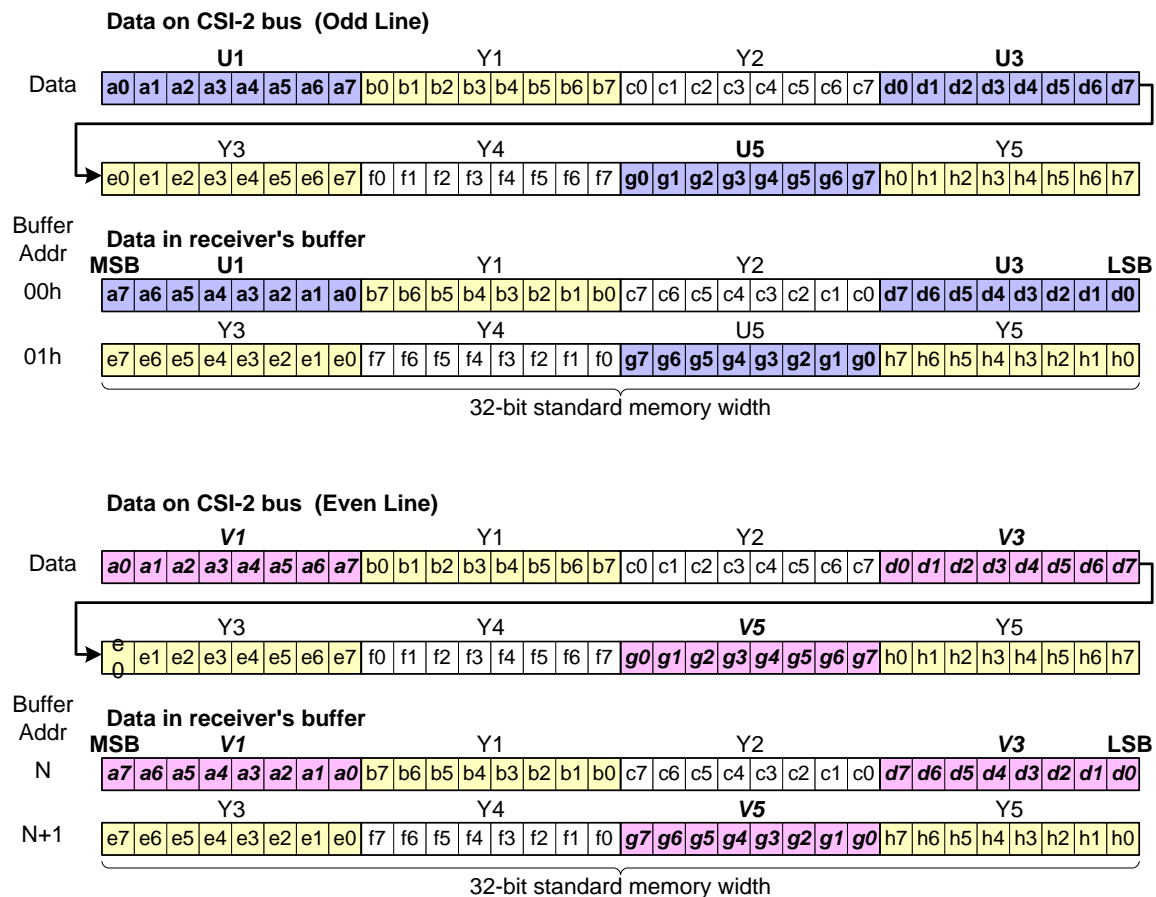


Figure 148 YUV420 8-bit Legacy Data Format Reception

10-Nov-2016

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.

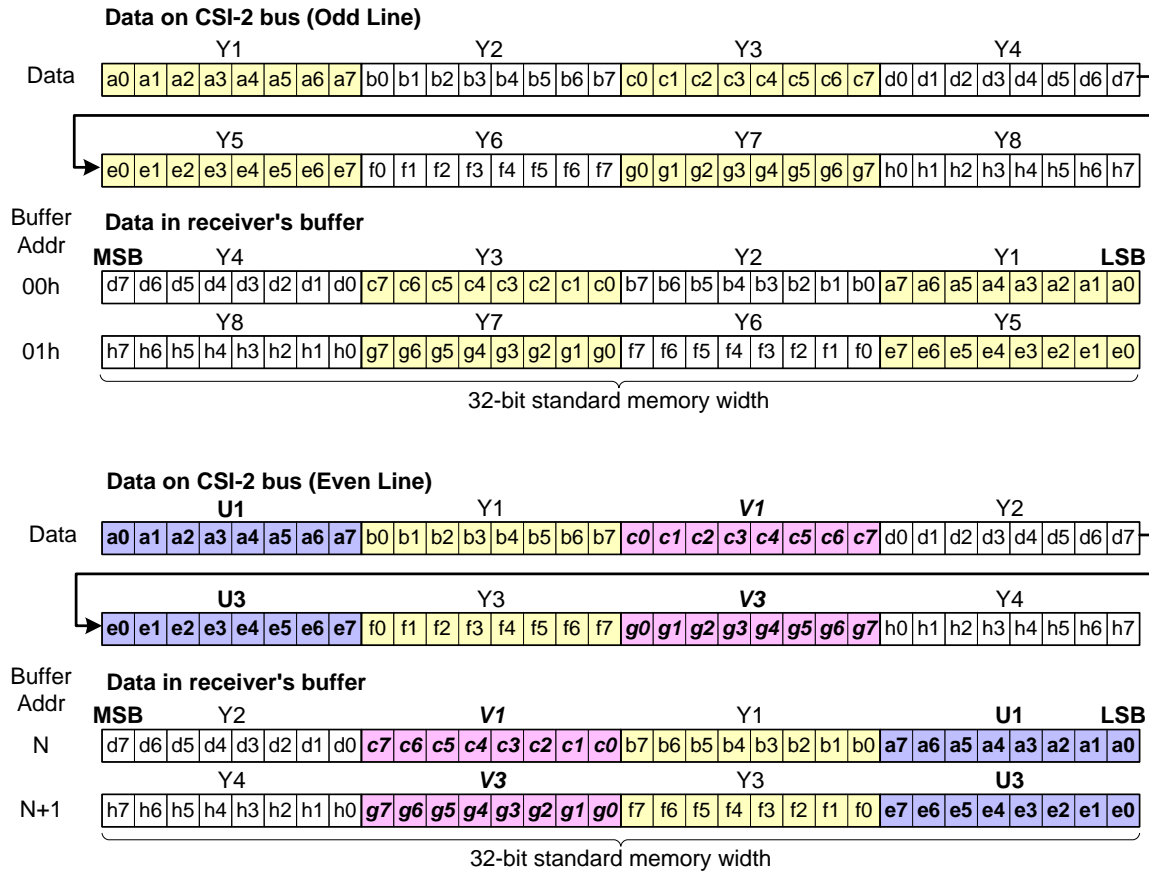


Figure 149 YUV420 8-bit Data Format Reception

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.

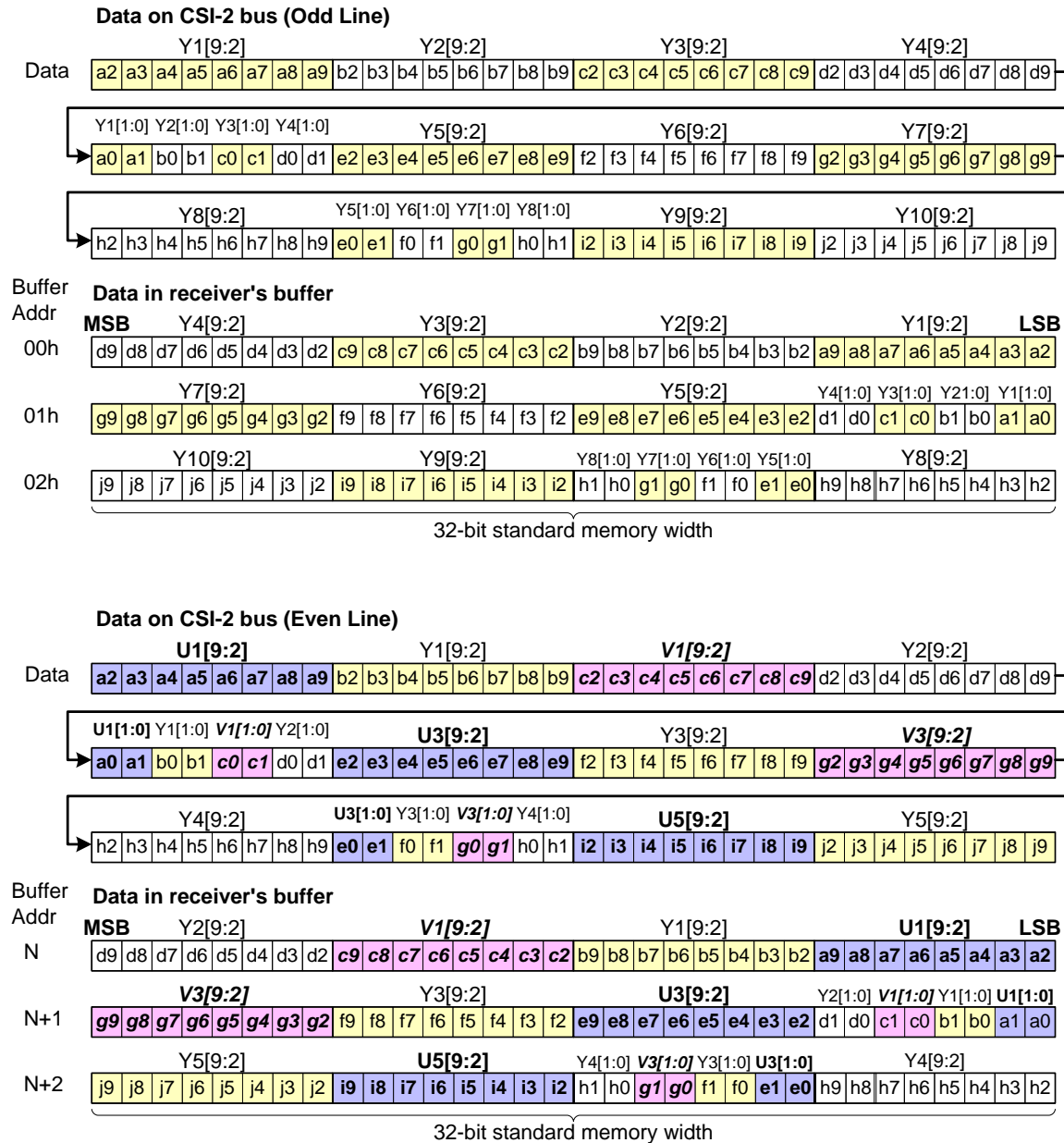


Figure 150 YUV420 10-bit Data Format Reception

10-Nov-2016

12.12 RAW6 Data Reception

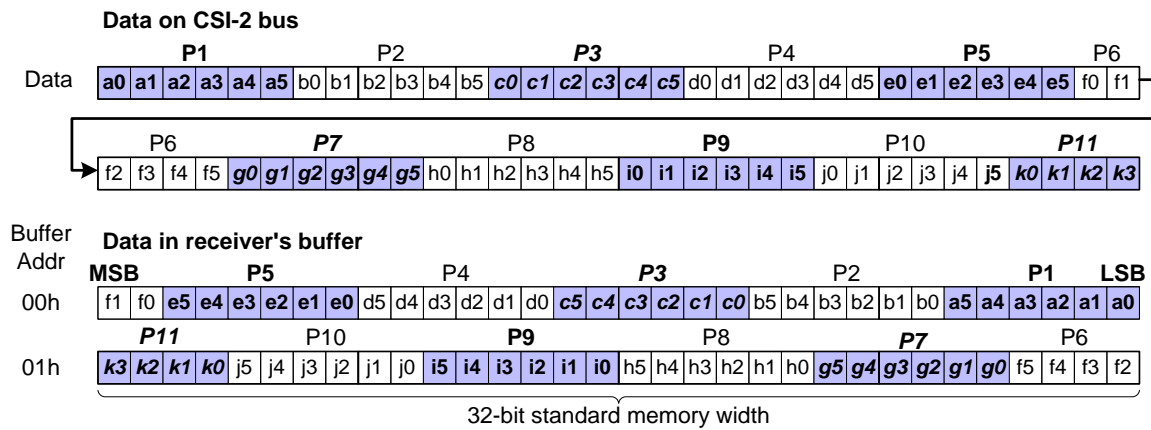


Figure 151 RAW6 Data Format Reception

12.13 RAW7 Data Reception

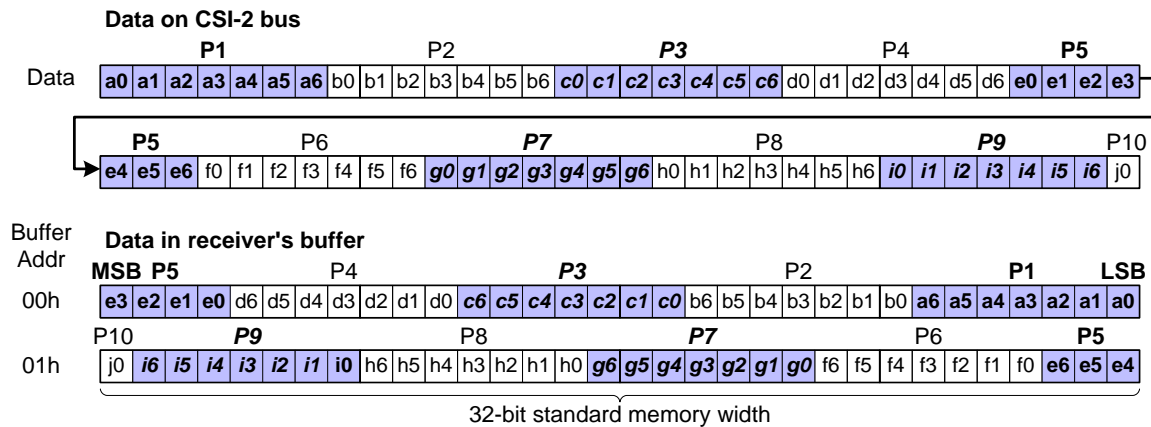


Figure 152 RAW7 Data Format Reception

12.14 RAW8 Data Reception

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

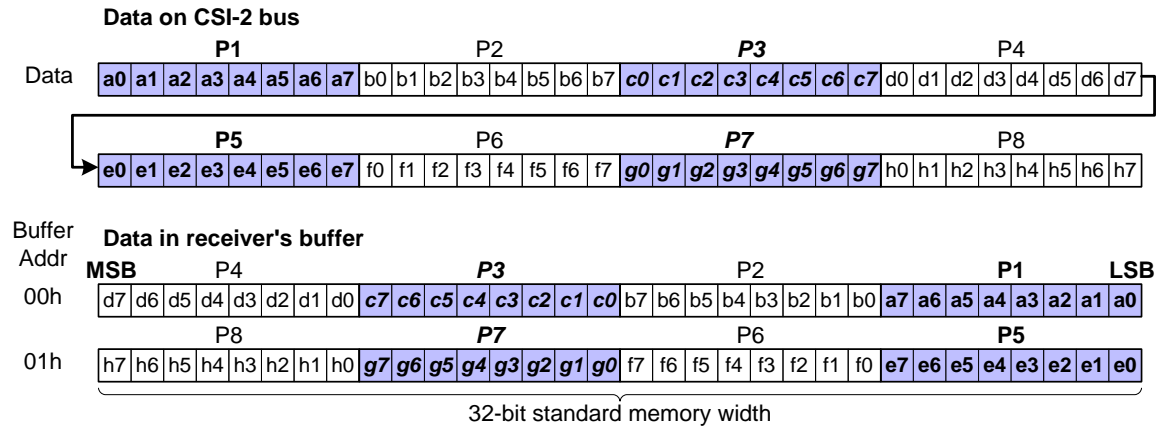


Figure 153 RAW8 Data Format Reception

12.15 RAW10 Data Reception

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

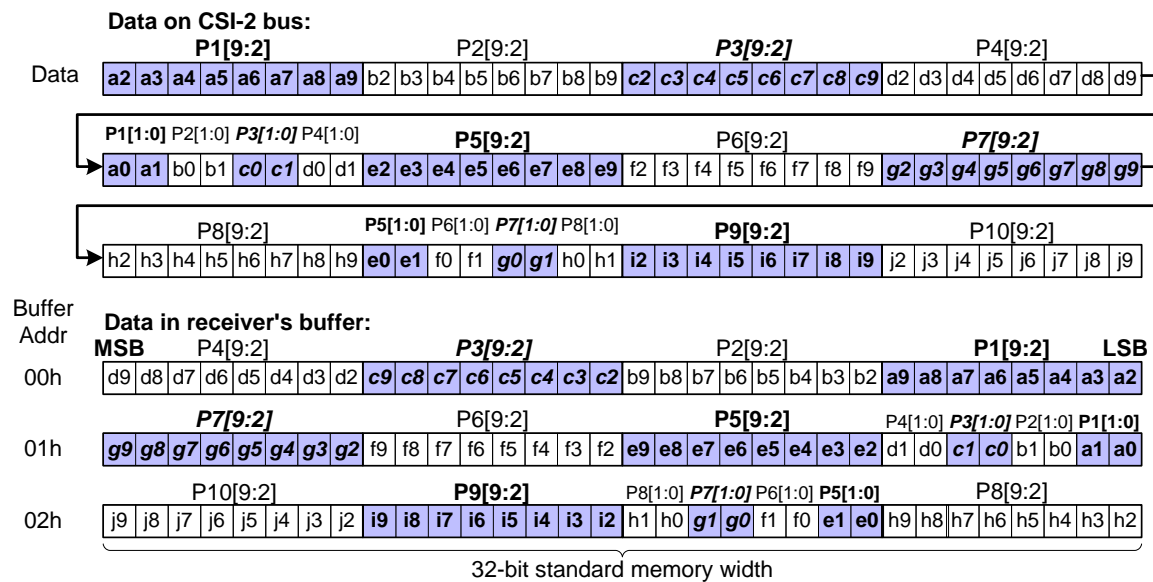


Figure 154 RAW10 Data Format Reception

10-Nov-2016

12.16 RAW12 Data Reception

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

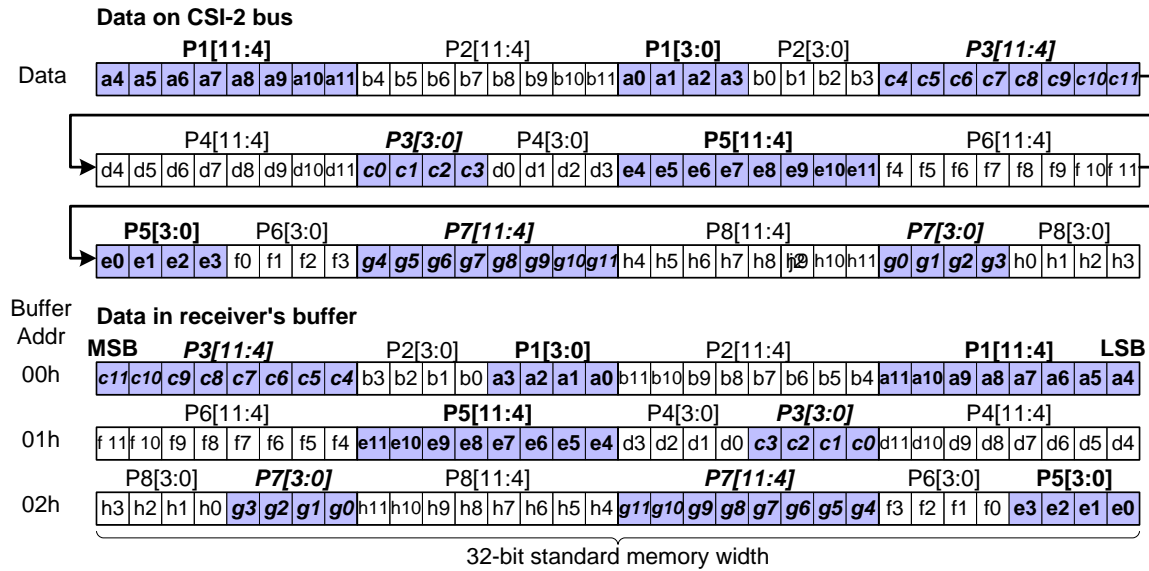


Figure 155 RAW12 Data Format Reception

12.17 RAW14 Data Reception

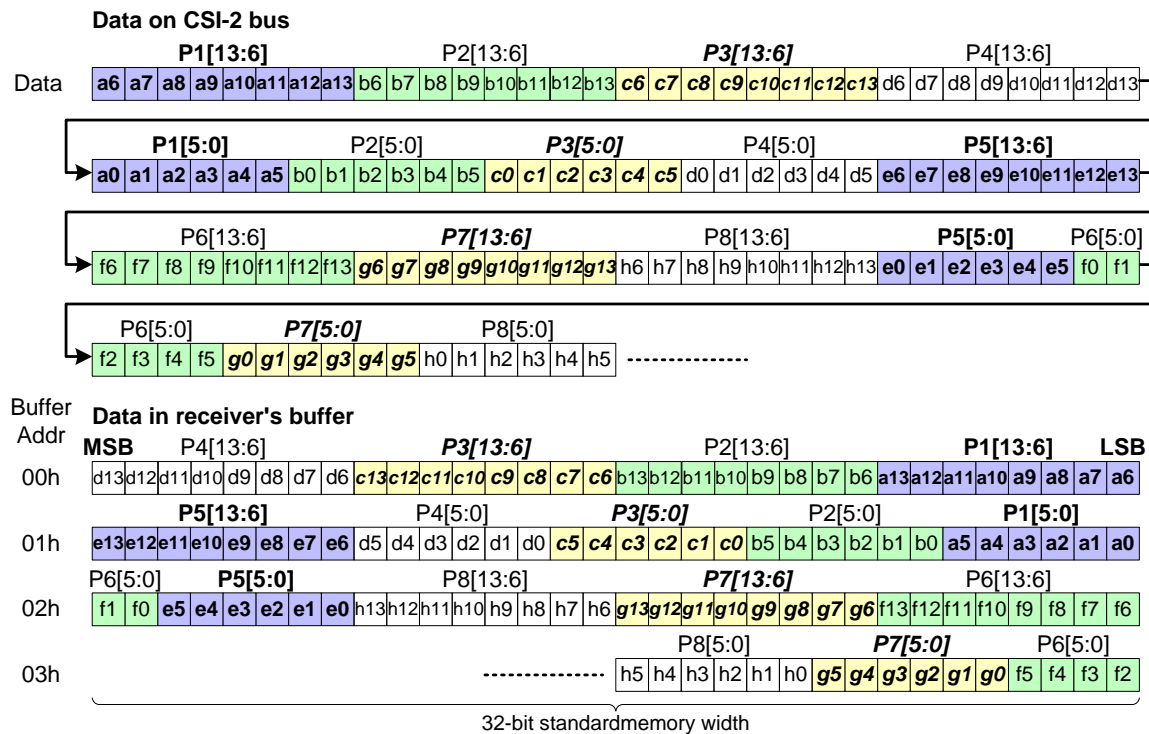


Figure 156 RAW 14 Data Format Reception

12.18 RAW16 Data Reception

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

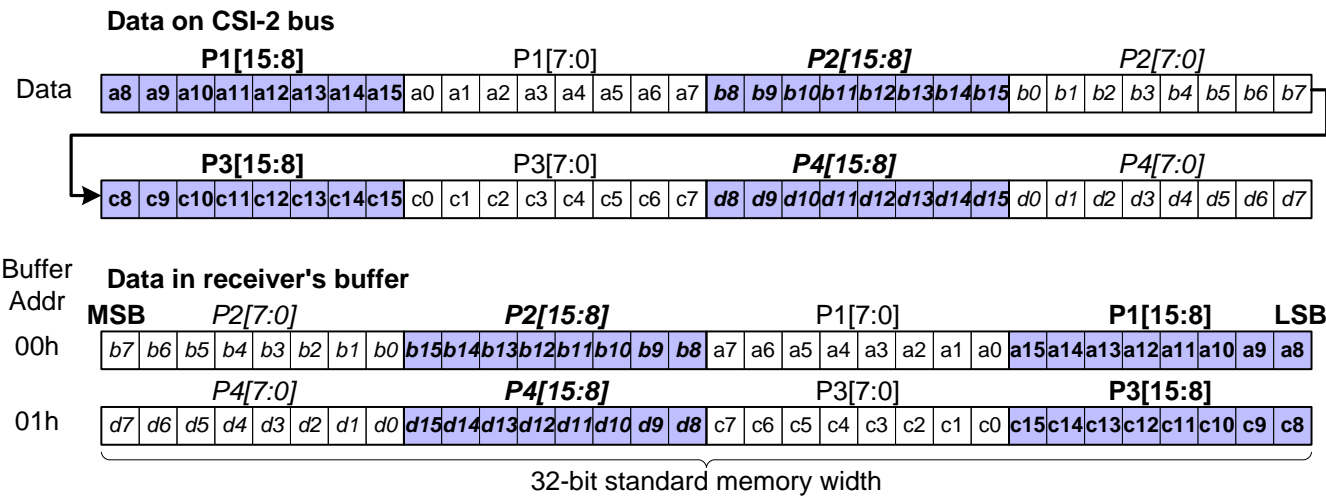


Figure 157 RAW16 Data Format Reception

12.19 RAW20 Data Reception

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

10-Nov-2016

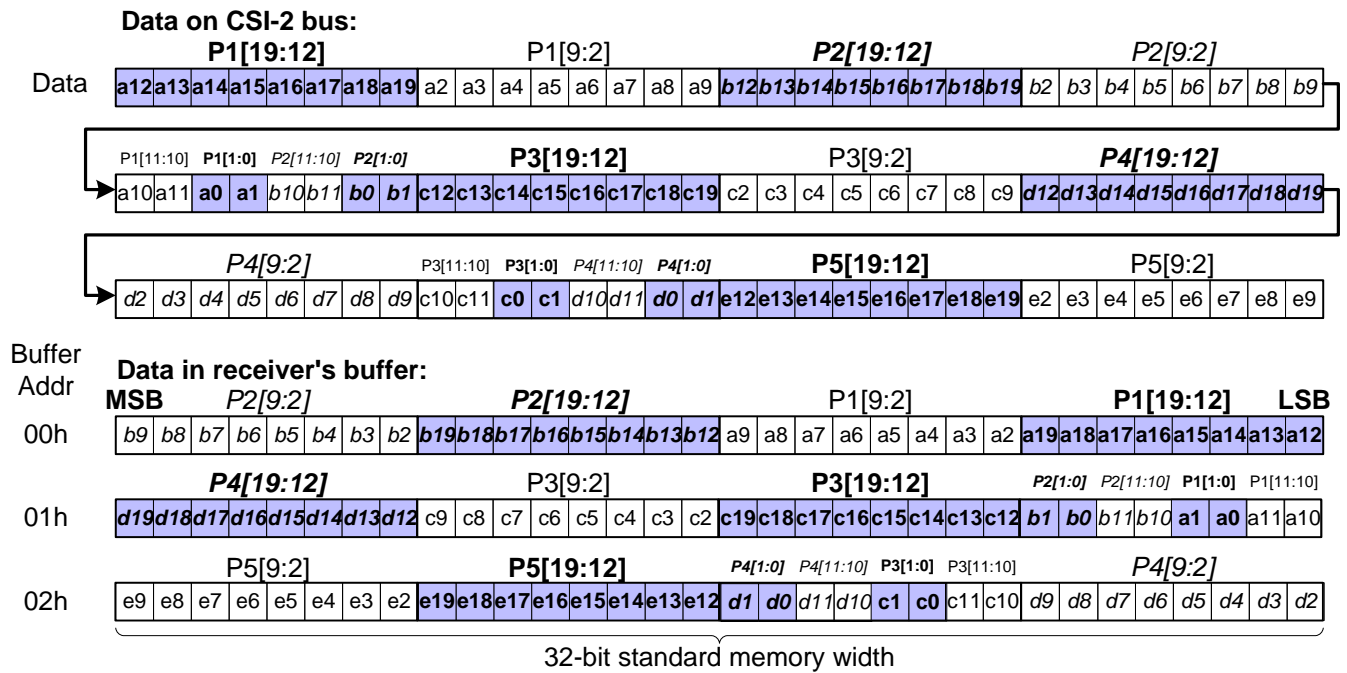


Figure 158 RAW20 Data Format Reception

This page intentionally left blank

10-Nov-2016

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 159* and *Figure 160*.

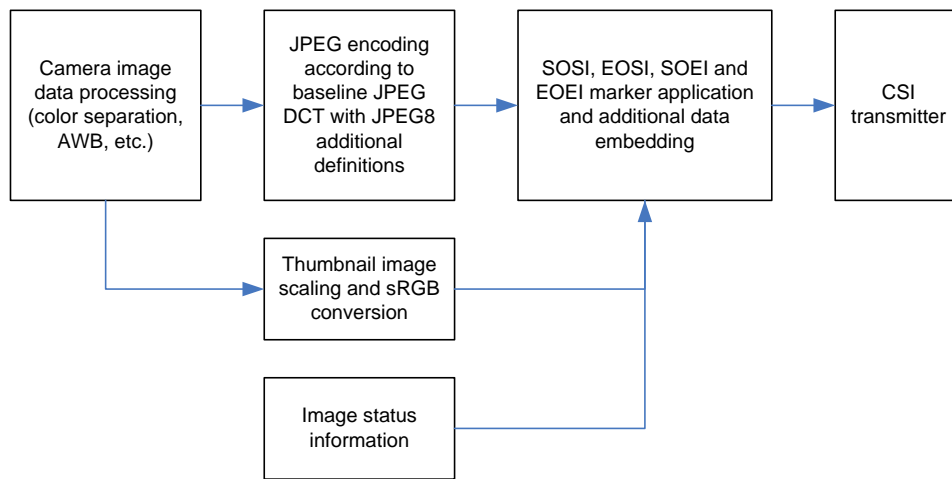


Figure 159 JPEG8 Data Flow in the Encoder

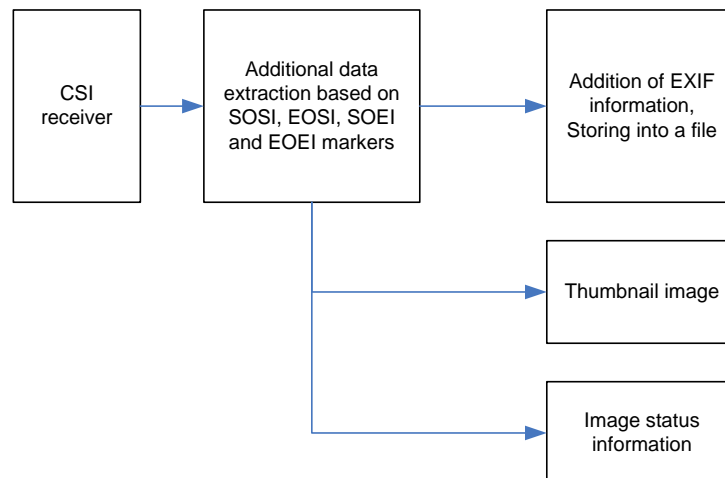


Figure 160 JPEG8 Data Flow in the Decoder

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 161*.
- 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*.

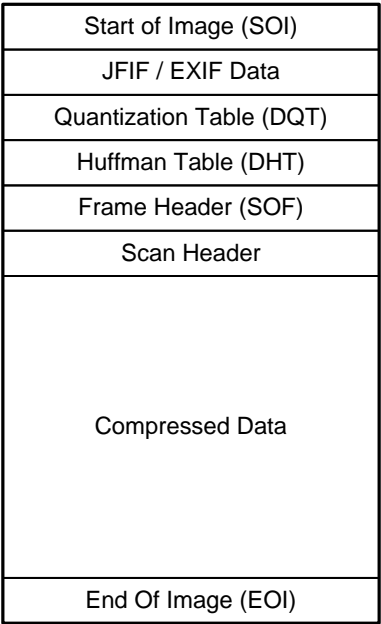


Figure 161 EXIF Compatible Baseline JPEG DCT Format

10-Nov-2016

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 162**:

- 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 39**. 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 39 Status Data Padding

Data Word	After Padding
D7D6D5D4 D3D2D1D0	1010D7D6D5D4 1010D3D2D1D0

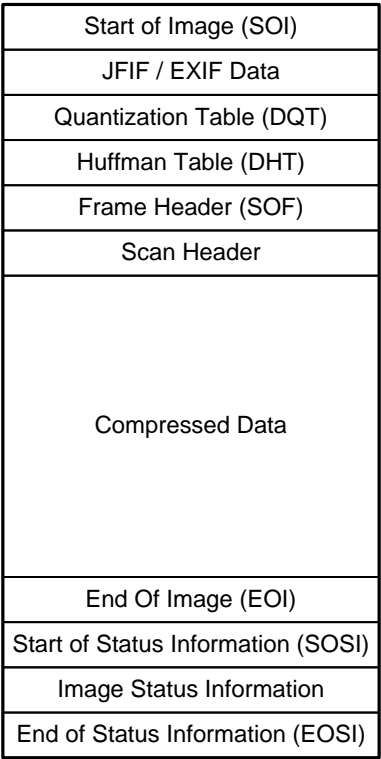


Figure 162 Status Information Field in the End of Baseline JPEG Frame

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 163*.

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.

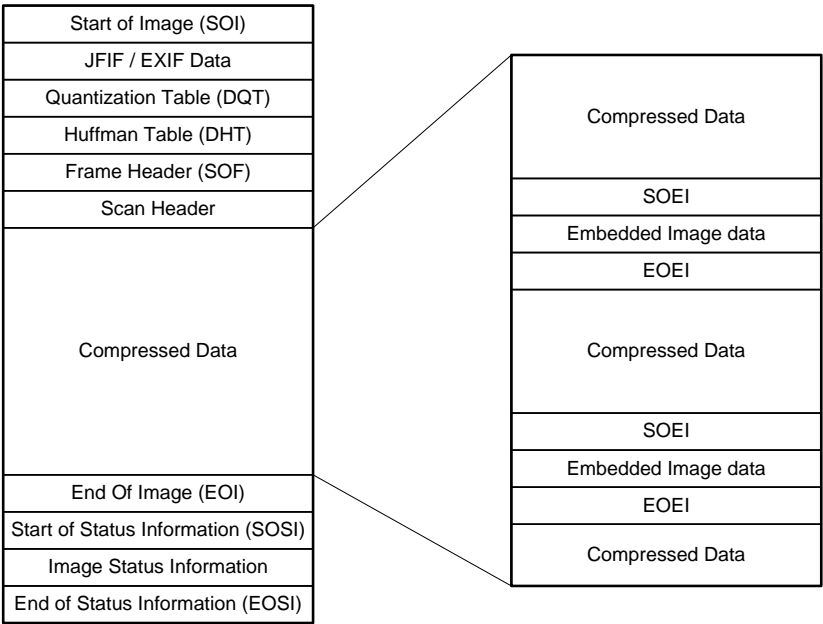


Figure 163 Example of TN Image Embedding Inside the Compressed JPEG Data Block

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 40 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.

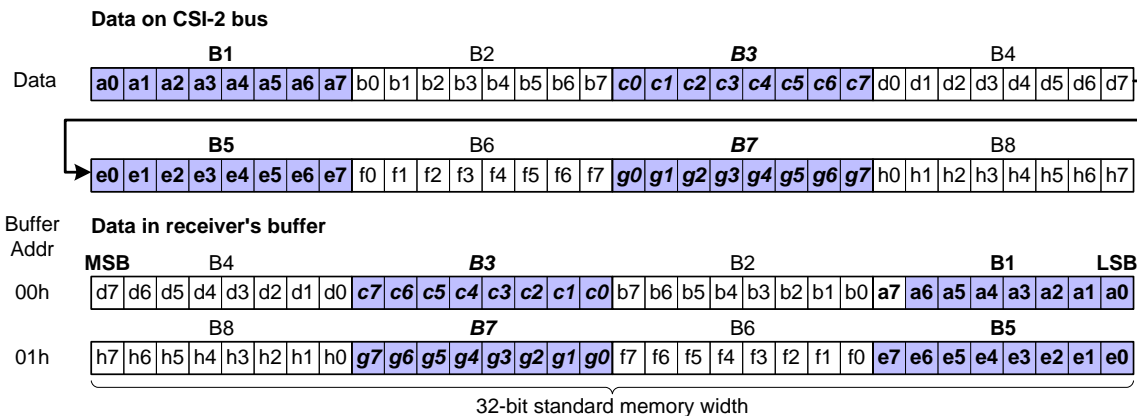


Figure 164 JPEG8 Data Format Reception

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 165*. This implementation example varies from the informative PPI example in [MIP101].

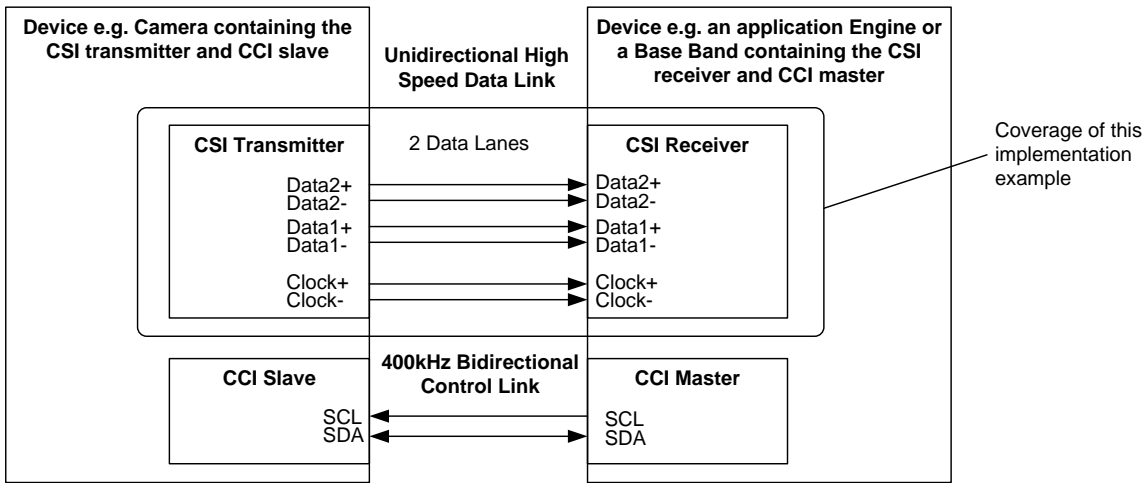


Figure 165 Implementation Example Block Diagram and Coverage

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 166**.

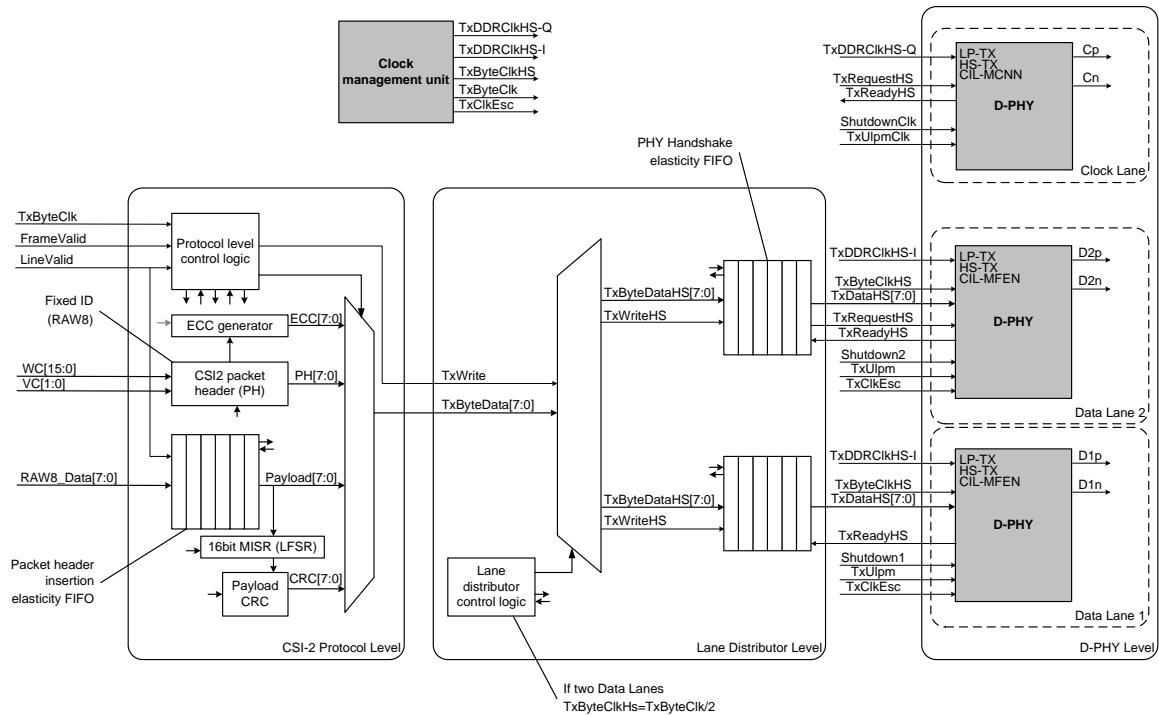


Figure 166 CSI-2 Transmitter Block Diagram

10-Nov-2016

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 167**.

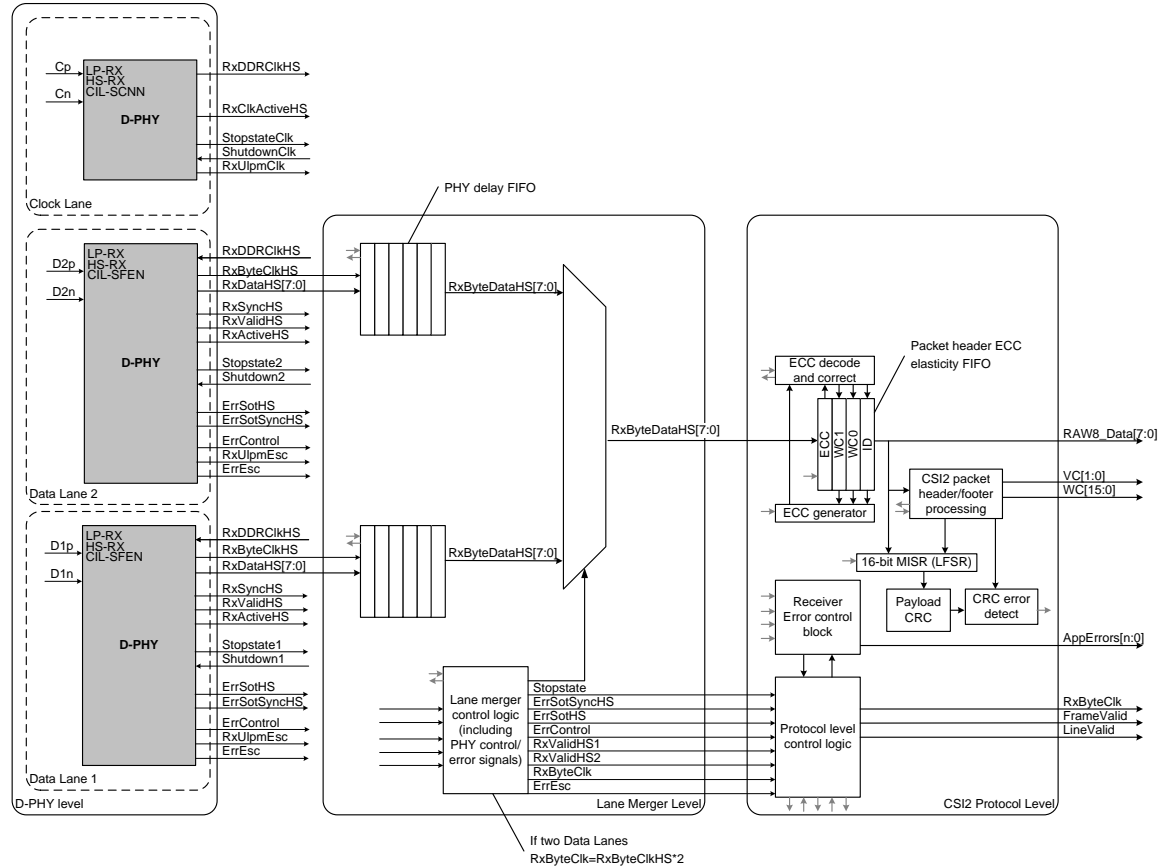


Figure 167 CSI-2 Receiver Block Diagram

B.4 Details on the D-PHY Implementation

The PHY level of implementation has the top level structure as seen in *Figure 168*.

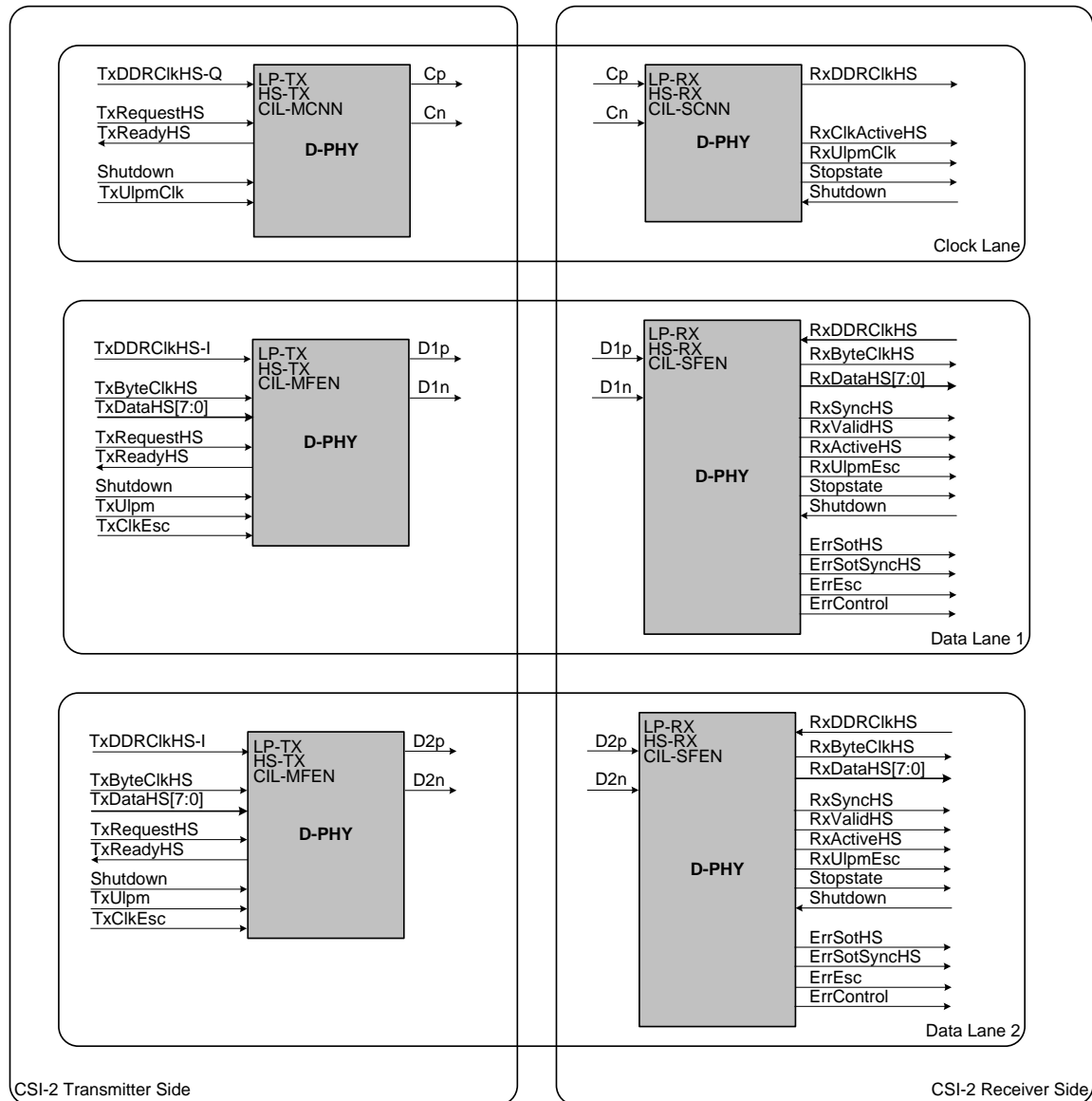


Figure 168 D-PHY Level Block Diagram

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)

10-Nov-2016

- Data2 lane (Receiver)

B.4.1 CSI-2 Clock Lane Transmitter

The suggested implementation can be seen in *Figure 169*.

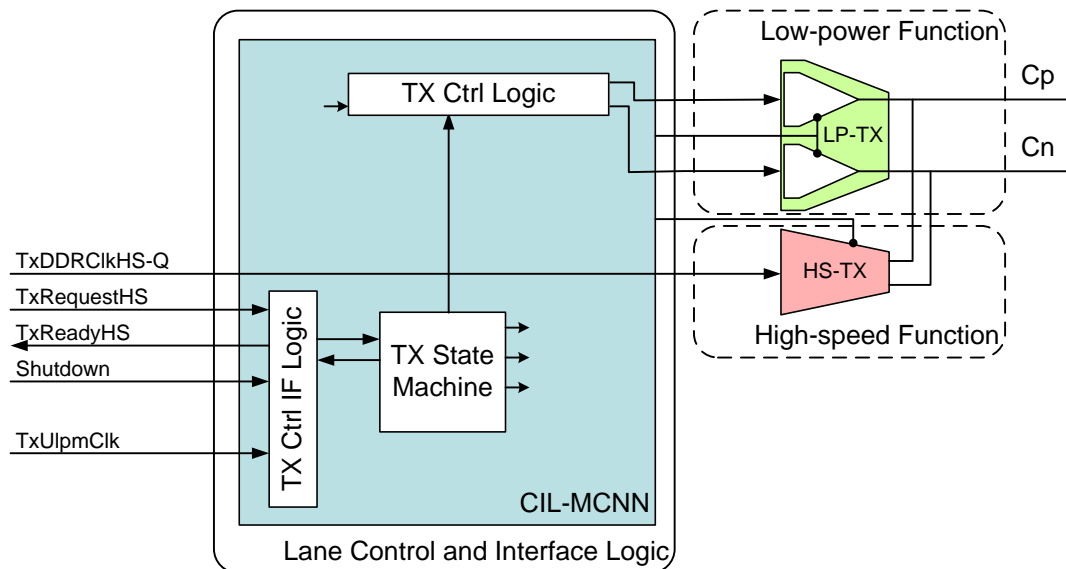


Figure 169 CSI-2 Clock Lane Transmitter

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:

- **TxDDRCIkHS-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 170*.

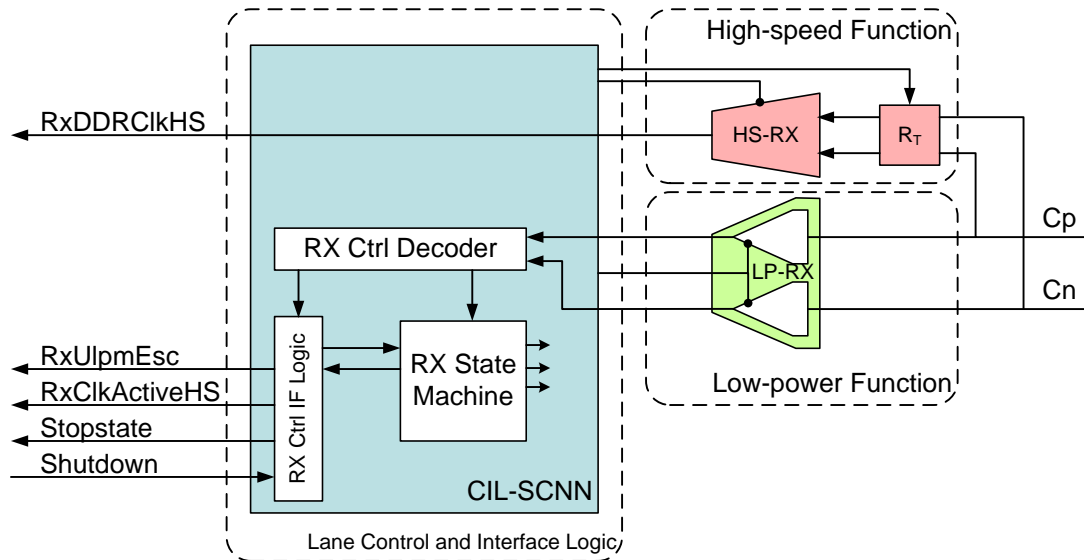


Figure 170 CSI-2 Clock Lane Receiver

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.

10-Nov-2016

B.4.3 CSI-2 Data Lane Transmitter

The suggested implementation can be seen in *Figure 171*.

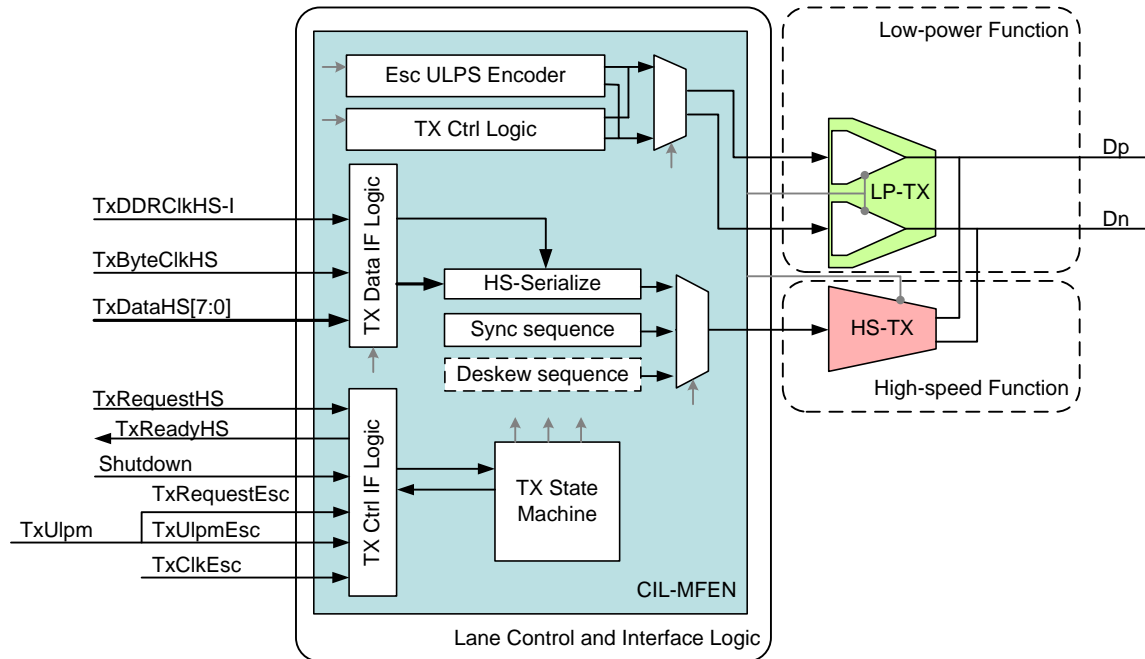


Figure 171 CSI-2 Data Lane Transmitter

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:

- **TxDDRCIkHS-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.

- 1750 • **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that
1751 TxDataHS is accepted by the lane module to be serially transmitted. TxReadyHS is valid on rising
1752 edges of TxByteClkHS. Valid data has to be provided for the whole duration of active
1753 TxReadyHS.
- 1754 • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
1755 “shutdown”, disabling all activity. All line drivers, including terminators, are turned off when
1756 Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs
1757 are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on
1758 any clock.
- 1759 • **TxUlpmEsc** (Input): Escape mode Transmit Ultra Low Power. This active high signal is asserted
1760 with TxRequestEsc to cause the lane module to enter the Ultra Low-Power mode. The lane
1761 module remains in this mode until TxRequestEsc is de-asserted.
- 1762 • **TxRequestEsc** (Input): This active high signal, asserted together with TxUlpmEsc is used to
1763 request entry into escape mode. Once in escape mode, the lane stays in escape mode until
1764 TxRequestEsc is de-asserted. TxRequestEsc is only asserted by the protocol while TxRequestHS
1765 is low.
- 1766 • **TxCikEsc** (Input): Escape mode Transmit Clock. This clock is directly used to generate escape
1767 sequences. The period of this clock determines the symbol time for low power signals. It is
1768 therefore constrained by the normative part of the *[MIP101]*.

10-Nov-2016

B.4.4 CSI-2 Data Lane Receiver

The suggested implementation can be seen in *Figure 172*.

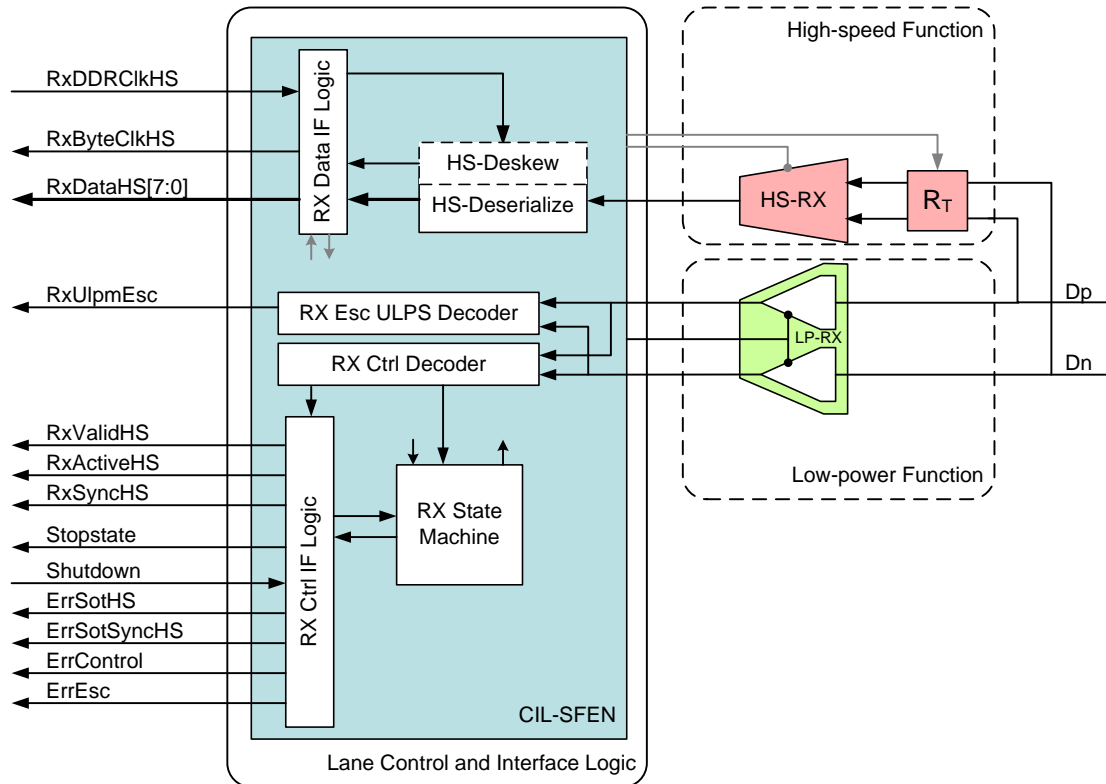


Figure 172 CSI-2 Data Lane Receiver

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 RxDDRCIkHS Clock.

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

- **RxDDRCIkHS** (Input): High-Speed Receive DDR Clock used to sample the data 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 RxDDRCIkHS.
- **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.

- 1791 • **RxActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the
1792 lane module is actively receiving a high-speed transmission from the lane interconnect.
- 1793 • **RxSyncHS** (Output): Receiver Synchronization Observed. This active high signal indicates that
1794 the lane module has seen an appropriate synchronization event. In a typical high-speed
1795 transmission, RxSyncHS is high for one cycle of RxByteClkHS at the beginning of a high-speed
1796 transmission when RxActiveHS is first asserted. This signal missing is signaled using
1797 ErrSotSyncHS.
- 1798 • **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is
1799 asserted to indicate that the lane module has entered the Ultra Low-Power mode. The lane module
1800 remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane
1801 interconnect.
- 1802 • **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is
1803 currently in Stop state. This signal is asynchronous.
- 1804 • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
1805 “shutdown”, disabling all activity. All line drivers, including terminators, are turned off when
1806 Shutdown is asserted. When Shutdown is high, all PPI outputs are driven to the default inactive
1807 state. Shutdown is a level sensitive signal and does not depend on any clock.
- 1808 • **ErrSotHS** (Output): Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is
1809 corrupted, but in such a way that proper synchronization can still be achieved, this error signal is
1810 asserted for one cycle of RxByteClkHS. This is considered to be a “soft error” in the leader
1811 sequence and confidence in the payload data is reduced.
- 1812 • **ErrSotSyncHS** (Output): Start-of-Transmission Synchronization Error. If the high-speed SoT
1813 leader sequence is corrupted in a way that proper synchronization cannot be expected, this error is
1814 asserted for one cycle of RxByteClkHS.
- 1815 • **ErrControl** (Output): Control Error. This signal is asserted when an incorrect line state sequence
1816 is detected.
- 1817 • **ErrEsc** (Output): Escape Entry Error. If an unrecognized escape entry command is received, this
1818 signal is asserted and remains high until the next change in line state. The only escape entry
1819 command supported by the receiver is the ULPS.

10-Nov-2016

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 [MIP101]. 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 [MIP101], PPI Annex, signal errors at the PHY-Protocol Interface (PPI) level consists 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.

10-Nov-2016

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.

10-Nov-2016

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 173**, 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.

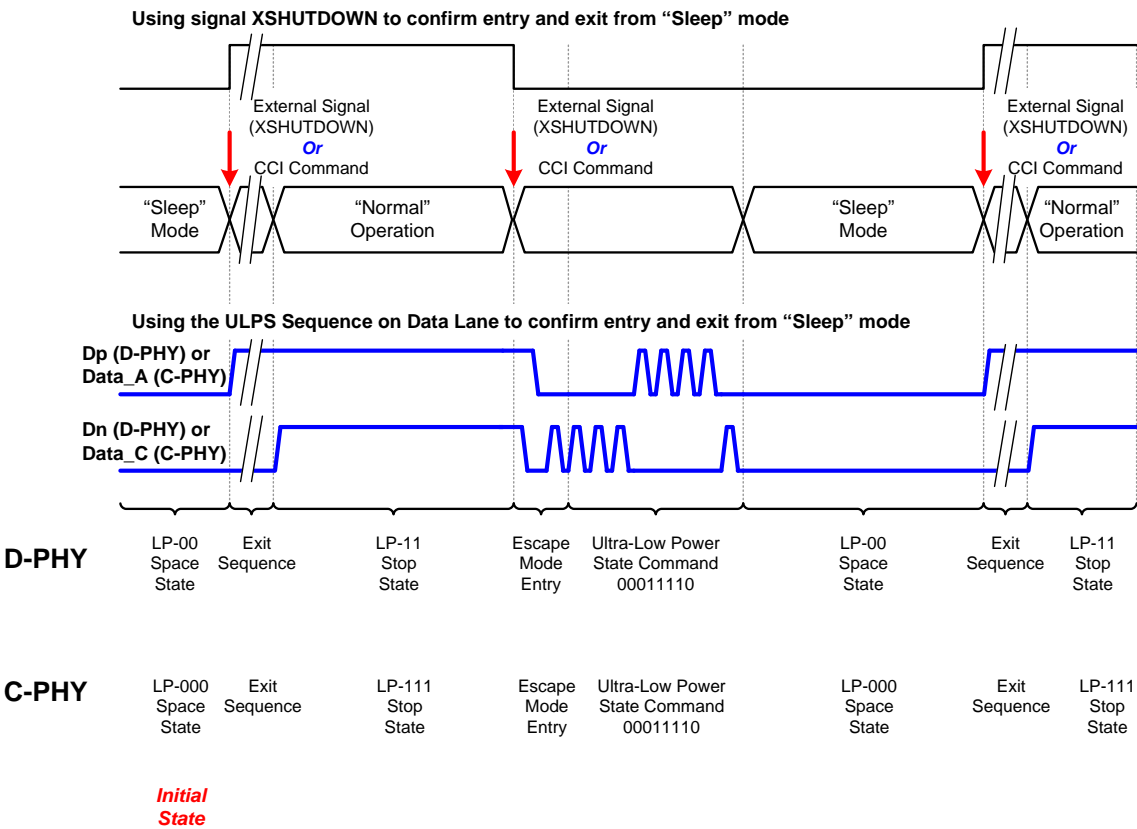


Figure 173 SLM Synchronization

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.

10-Nov-2016

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 38**. 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 less 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 174**.

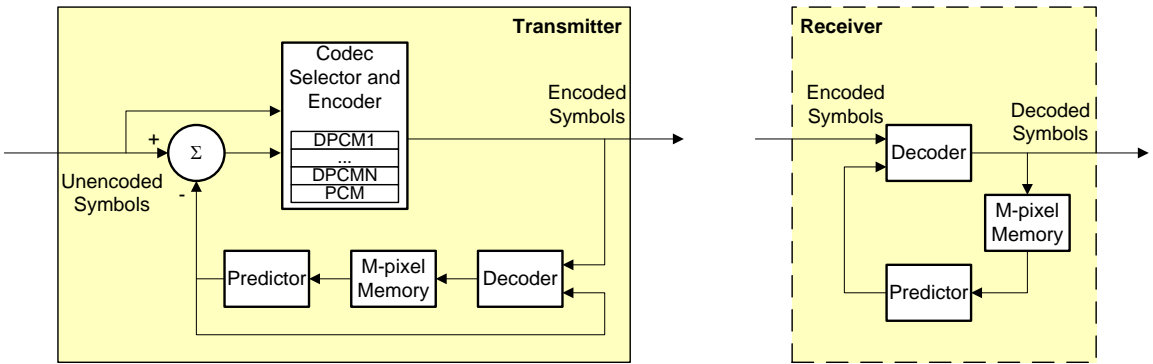


Figure 174 Data Compression System Block Diagram

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**) ($\text{abs}(\mathbf{Xorig} - \mathbf{Xpred}) < \text{difference limit}$), 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**).

10-Nov-2016

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 175*.

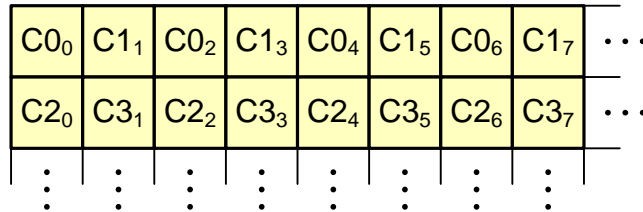


Figure 175 Pixel Order of the Original Image

Figure 176 shows an example of the pixel order with RGB data.

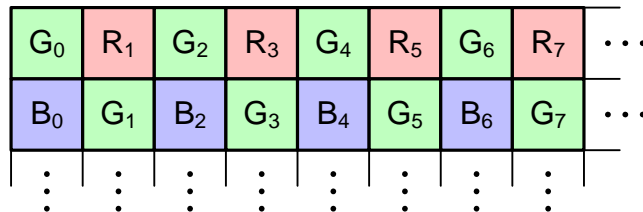


Figure 176 Example Pixel Order of the Original Image

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 (C0₀, C1₁ / C2₀, C3₁ or as in example G₀, R₁ / B₀, G₁) 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:

$$X_{pred}(n) = X_{deco}(n-2)$$

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
```

10-Nov-2016

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

2111 **Xenco(n)** has the following format:
 2112 **Xenco(n)** = "010 s xxxx"
 2113 where,
 2114 "010" is the code word
 2115 "s" is the **sign** bit
 2116 "xxxx" is the four bit **value** field
 2117 The coder equation is described as follows:
 2118 if (**Xdiff(n)** < 0) then
 2119 **sign** = 1
 2120 else
 2121 **sign** = 0
 2122 endif
 2123 **value** = (abs(**Xdiff(n)**) - 32) / 2

E.2.1.3 DPCM3 for 10–8–10 Coder

2124 **Xenco(n)** has the following format:
 2125 **Xenco(n)** = "011 s xxxx"
 2126 where,
 2127 "011" is the code word
 2128 "s" is the **sign** bit
 2129 "xxxx" is the four bit **value** field
 2130 The coder equation is described as follows:
 2131 if (**Xdiff(n)** < 0) then
 2132 **sign** = 1
 2133 else
 2134 **sign** = 0
 2135 endif
 2136 **value** = (abs(**Xdiff(n)**) - 64) / 4

E.2.1.4 PCM for 10–8–10 Coder

2137 **Xenco(n)** has the following format:
 2138 **Xenco(n)** = "1 xxxxxxxx"
 2139 where,
 2140 "1" is the code word
 2141 the **sign** bit is not used
 2142 "xxxxxxx" is the seven bit **value** field
 2143 The coder equation is described as follows:
 2144 **value** = **Xorig(n)** / 8

10-Nov-2016

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

2177 **Xenco(n)** has the following format:

2178 **Xenco(n)** = "0010 s xx"

2179 where,

2180 "0010" is the code word

2181 "s" is the **sign** bit

2182 "xx" is the two bit **value** field

2183 The coder equation is described as follows:

2184 if (**Xdiff(n)** < 0) then

2185 **sign** = 1

2186 else

2187 **sign** = 0

2188 endif

2189 **value** = (abs(**Xdiff(n)**) - 8) / 2

E.2.2.3 DPCM3 for 10–7–10 Coder

2190 **Xenco(n)** has the following format:

2191 **Xenco(n)** = "0011 s xx"

2192 where,

2193 "0011" is the code word

2194 "s" is the **sign** bit

2195 "xx" is the two bit **value** field

2196 The coder equation is described as follows:

2197 if (**Xdiff(n)** < 0) then

2198 **sign** = 1

2199 else

2200 **sign** = 0

2201 endif

2202 **value** = (abs(**Xdiff(n)**) - 16) / 4

E.2.2.4 DPCM4 for 10–7–10 Coder

2203 **Xenco(n)** has the following format:

2204 **Xenco(n)** = "01 s xxxx"

2205 where,

2206 "01" is the code word

2207 "s" is the **sign** bit

2208 "xxxx" is the four bit **value** field

2209 The coder equation is described as follows:

2210 if (**Xdiff(n)** < 0) then

2211 **sign** = 1

2212 else

2213 **sign** = 0

2214 endif

2215 **value** = (abs(**Xdiff(n)**) - 32) / 8

10-Nov-2016

E.2.2.5 PCM for 10–7–10 Coder

2216 **Xenco(n)** has the following format:

2217 **Xenco(n)** = "1 xxxxxx"

2218 where,

2219 "1" is the code word

2220 the **sign** bit is not used

2221 "xxxxxx" is the six bit **value** field

2222 The coder equation is described as follows:

2223 **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
```

10-Nov-2016

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

2305 **Xenco(n)** has the following format:

2306 **Xenco(n) = "1 xxxxx"**

2307 where,

2308 "1" is the code word

2309 the **sign** bit is not used

2310 "xxxxx" is the five bit **value** field

2311 The coder equation is described as follows:

2312 **value = Xorig(n) / 32**

10-Nov-2016

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 xxxxxxxxxxx"

where,

"1" is the code word

the **sign** bit is not used

"xxxxxxxxxx" is the nine bit **value** field

The coder equation is described as follows:

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

10-Nov-2016

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

2415 **Xenco(n)** has the following format:
 2416 **Xenco(n)** = "011 s xxxx"
 2417 where,
 2418 "011" is the code word
 2419 "s" is the **sign** bit
 2420 "xxxx" is the four bit **value** field
 2421 The coder equation is described as follows:
 2422 if (**Xdiff(n)** < 0) then
 2423 **sign** = 1
 2424 else
 2425 **sign** = 0
 2426 endif
 2427 **value** = (abs(**Xdiff(n)**) - 8) / 2

E.2.5.3 DPCM3 for 12–8–12 Coder

2428 **Xenco(n)** has the following format:
 2429 **Xenco(n)** = "010 s xxxx"
 2430 where,
 2431 "010" is the code word
 2432 "s" is the **sign** bit
 2433 "xxxx" is the four bit **value** field
 2434 The coder equation is described as follows:
 2435 if (**Xdiff(n)** < 0) then
 2436 **sign** = 1
 2437 else
 2438 **sign** = 0
 2439 endif
 2440 **value** = (abs(**Xdiff(n)**) - 40) / 4

E.2.5.4 DPCM4 for 12–8–12 Coder

2441 **Xenco(n)** has the following format:
 2442 **Xenco(n)** = "001 s xxxx"
 2443 where,
 2444 "001" is the code word
 2445 "s" is the **sign** bit
 2446 "xxxx" is the four bit **value** field
 2447 The coder equation is described as follows:
 2448 if (**Xdiff(n)** < 0) then
 2449 **sign** = 1
 2450 else
 2451 **sign** = 0
 2452 endif
 2453 **value** = (abs(**Xdiff(n)**) - 104) / 8

10-Nov-2016

E.2.5.5 DPCM5 for 12–8–12 Coder

2454 **Xenco(n)** has the following format:

2455 **Xenco(n)** = "0001 s xxx"

2456 where,

2457 "0001" is the code word

2458 "s" is the **sign** bit

2459 "xxx" is the three bit **value** field

2460 The coder equation is described as follows:

2461 if (**Xdiff(n)** < 0) then

2462 **sign** = 1

2463 else

2464 **sign** = 0

2465 endif

2466 **value** = (abs(**Xdiff(n)**) - 232) / 16

E.2.5.6 PCM for 12–8–12 Coder

2467 **Xenco(n)** has the following format:

2468 **Xenco(n)** = "1 xxxxxxx"

2469 where,

2470 "1" is the code word

2471 the **sign** bit is not used

2472 "xxxxxxx" is the seven bit **value** field

2473 The coder equation is described as follows:

2474 **value** = **Xorig(n)** / 32

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

2475 The 12–7–12 coder offers 42% bit rate reduction with high image quality.

2476 Pixels without prediction are encoded using the following formula:

2477 **Xenco**(**n**) = **Xorig**(**n**) / 32

2478 To avoid a full-zero encoded value, the following check is performed:

2479 if (**Xenco**(**n**) == 0) then

2480 **Xenco**(**n**) = 1

2481 endif

2482 Pixels with prediction are encoded using the following formula:

2483 if (abs(**Xdiff**(**n**)) < 4) then

2484 use **DPCM1**

2485 else if (abs(**Xdiff**(**n**)) < 12) then

2486 use **DPCM2**

2487 else if (abs(**Xdiff**(**n**)) < 28) then

2488 use **DPCM3**

2489 else if (abs(**Xdiff**(**n**)) < 92) then

2490 use **DPCM4**

2491 else if (abs(**Xdiff**(**n**)) < 220) then

2492 use **DPCM5**

2493 else if (abs(**Xdiff**(**n**)) < 348) then

2494 use **DPCM6**

2495 else

2496 use **PCM**

2497 endif

E.2.6.1 DPCM1 for 12–7–12 Coder

2498 **Xenco**(**n**) has the following format:

2499 **Xenco**(**n**) = "0000 s xx"

2500 where,

2501 "0000" is the code word

2502 "s" is the **sign** bit

2503 "xx" is the two bit **value** field

2504 The coder equation is described as follows:

2505 if (**Xdiff**(**n**) <= 0) then

2506 **sign** = 1

2507 else

2508 **sign** = 0

2509 endif

2510 **value** = abs(**Xdiff**(**n**))

2511 *Note: Zero code has been avoided (0 is sent as -0).*

10-Nov-2016

E.2.6.2 DPCM2 for 12–7–12 Coder

2512 **Xenco(n)** has the following format:

2513 **Xenco(n)** = "0001 s xx"

2514 where,

2515 "0001" is the code word

2516 "s" is the **sign** bit

2517 "xx" is the two bit **value** field

2518 The coder equation is described as follows:

2519 if (**Xdiff(n)** < 0) then

2520 **sign** = 1

2521 else

2522 **sign** = 0

2523 endif

2524 **value** = (abs(**Xdiff(n)**) - 4) / 2

E.2.6.3 DPCM3 for 12–7–12 Coder

2525 **Xenco(n)** has the following format:

2526 **Xenco(n)** = "0010 s xx"

2527 where,

2528 "0010" is the code word

2529 "s" is the **sign** bit

2530 "xx" is the two bit **value** field

2531 The coder equation is described as follows:

2532 if (**Xdiff(n)** < 0) then

2533 **sign** = 1

2534 else

2535 **sign** = 0

2536 endif

2537 **value** = (abs(**Xdiff(n)**) - 12) / 4

E.2.6.4 DPCM4 for 12–7–12 Coder

2538 **Xenco(n)** has the following format:

2539 **Xenco(n)** = "010 s xxx"

2540 where,

2541 "010" is the code word

2542 "s" is the **sign** bit

2543 "xxx" is the three bit **value** field

2544 The coder equation is described as follows:

2545 if (**Xdiff(n)** < 0) then

2546 **sign** = 1

2547 else

2548 **sign** = 0

2549 endif

2550 **value** = (abs(**Xdiff(n)**) - 28) / 8

E.2.6.5 DPCM5 for 12–7–12 Coder

2551 **Xenco(n)** has the following format:

2552 **Xenco(n)** = "011 s xxx"

2553 where,

2554 "011" is the code word

2555 "s" is the **sign** bit

2556 "xxx" is the three bit **value** field

2557 The coder equation is described as follows:

2558 if (**Xdiff(n)** < 0) then

2559 **sign** = 1

2560 else

2561 **sign** = 0

2562 endif

2563 **value** = (abs(**Xdiff(n)**) - 92) / 16

E.2.6.6 DPCM6 for 12–7–12 Coder

2564 **Xenco(n)** has the following format:

2565 **Xenco(n)** = "0011 s xx"

2566 where,

2567 "0011" is the code word

2568 "s" is the **sign** bit

2569 "xx" is the two bit **value** field

2570 The coder equation is described as follows:

2571 if (**Xdiff(n)** < 0) then

2572 **sign** = 1

2573 else

2574 **sign** = 0

2575 endif

2576 **value** = (abs(**Xdiff(n)**) - 220) / 32

E.2.6.7 PCM for 12–7–12 Coder

2577 **Xenco(n)** has the following format:

2578 **Xenco(n)** = "1 xxxxxx"

2579 where,

2580 "1" is the code word

2581 the **sign** bit is not used

2582 "xxxxxx" is the six bit **value** field

2583 The coder equation is described as follows:

2584 **value** = **Xorig(n)** / 64

10-Nov-2016

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

2623 **Xenco(n)** has the following format:

2624 **Xenco(n)** = "0001 s x"

2625 where,

2626 "0001" is the code word

2627 "s" is the **sign** bit

2628 "x" is the one bit **value** field

2629 The coder equation is described as follows:

2630 if (**Xdiff(n)** < 0) then

2631 **sign** = 1

2632 else

2633 **sign** = 0

2634 endif

2635 **value** = (abs(**Xdiff(n)**) - 2) / 4

E.2.7.3 DPCM4 for 12–6–12 Coder

2636 **Xenco(n)** has the following format:

2637 **Xenco(n)** = "010 s xx"

2638 where,

2639 "010" is the code word

2640 "s" is the **sign** bit

2641 "xx" is the two bit **value** field

2642 The coder equation is described as follows:

2643 if (**Xdiff(n)** < 0) then

2644 **sign** = 1

2645 else

2646 **sign** = 0

2647 endif

2648 **value** = (abs(**Xdiff(n)**) - 10) / 8

E.2.7.4 DPCM5 for 12–6–12 Coder

2649 **Xenco(n)** has the following format:

2650 **Xenco(n)** = "0010 s x"

2651 where,

2652 "0010" is the code word

2653 "s" is the **sign** bit

2654 "x" is the one bit **value** field

2655 The coder equation is described as follows:

2656 if (**Xdiff(n)** < 0) then

2657 **sign** = 1

2658 else

2659 **sign** = 0

2660 endif

2661 **value** = (abs(**Xdiff(n)**) - 42) / 16

10-Nov-2016

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

2696 There are six different decoders available, one for each data compression scheme.

2697 For all decoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
2698 for predicted pixels.

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

2699 Pixels without prediction are decoded using the following formula:

2700 $\text{Xdco}(n) = 4 * \text{Xenco}(n) + 2$

2701 Pixels with prediction are decoded using the following formula:

```
2702 if (Xenco( n ) & 0xc0 == 0x00) then
2703     use DPCM1
2704 else if (Xenco( n ) & 0xe0 == 0x40) then
2705     use DPCM2
2706 else if (Xenco( n ) & 0xe0 == 0x60) then
2707     use DPCM3
2708 else
2709     use PCM
2710 endif
```

E.3.1.1 DPCM1 for 10–8–10 Decoder

2711 $\text{Xenco}(n)$ has the following format:

2712 $\text{Xenco}(n) = "00 \text{ s } \text{xxxxx}"$

2713 where,

2714 "00" is the code word

2715 "s" is the **sign** bit

2716 "xxxxx" is the five bit **value** field

2717 The decoder equation is described as follows:

```
2718 sign = Xenco( n ) & 0x20
2719 value = Xenco( n ) & 0x1f
2720 if (sign > 0) then
2721     Xdco( n ) = Xpred( n ) - value
2722 else
2723     Xdco( n ) = Xpred( n ) + value
2724 endif
```

10-Nov-2016

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) + 32if (**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 + 1if (**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

2759 **Xenco(n)** has the following format:

2760 **Xenco(n)** = "1 xxxxxxx"

2761 where,

2762 "1" is the code word

2763 the **sign** bit is not used

2764 "xxxxxxx" is the seven bit **value** field

2765 The codec equation is described as follows:

2766 **value** = 8 * (**Xenco(n)** & 0x7f)

2767 if (**value** > **Xpred(n)**) then

2768 **Xdeco(n)** = **value** + 3

2769 endif

2770 else

2771 **Xdeco(n)** = **value** + 4

2772 endif

10-Nov-2016

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

```

10-Nov-2016

E.3.2.5 PCM for 10–7–10 Decoder

2855 **Xenco(n)** has the following format:

2856 **Xenco(n)** = "1 xxxxxx"

2857 where,

2858 "1" is the code word

2859 the **sign** bit is not used

2860 "xxxxxx" is the six bit **value** field

2861 The codec equation is described as follows:

2862 **value** = 16 * (**Xenco(n)** & 0x3f)

2863 if (**value** > **Xpred(n)**) then

2864 **Xdeco(n)** = **value** + 7

2865 else

2866 **Xdeco(n)** = **value** + 8

2867 endif

E.3.3 Decoder for 10–6–10 Data Compression

2868 Pixels without prediction are decoded using the following formula:

2869 $\text{Xdeco}(n) = 16 * \text{Xenco}(n) + 8$

2870 Pixels with prediction are decoded using the following formula:

```

2871   if (Xenco( n ) & 0x3e == 0x00) then
2872       use DPCM1
2873   else if (Xenco( n ) & 0x3e == 0x02) then
2874       use DPCM2
2875   else if (Xenco( n ) & 0x3c == 0x04) then
2876       use DPCM3
2877   else if (Xenco( n ) & 0x38 == 0x08) then
2878       use DPCM4
2879   else if (Xenco( n ) & 0x30 == 0x10) then
2880       use DPCM5
2881   else
2882       use PCM
2883   endif

```

E.3.3.1 DPCM1 for 10–6–10 Decoder

2884 **Xenco(n)** has the following format:

2885 $\text{Xenco}(n) = \text{"00000 s"}$

2886 where,

2887 "00000" is the code word
 2888 "s" is the **sign** bit
 2889 the **value** field is not used

2890 The codec equation is described as follows:

2891 $\text{Xdeco}(n) = \text{Xpred}(n)$

E.3.3.2 DPCM2 for 10–6–10 Decoder

2892 **Xenco(n)** has the following format:

2893 $\text{Xenco}(n) = \text{"00001 s"}$

2894 where,

2895 "00001" is the code word
 2896 "s" is the **sign** bit
 2897 the **value** field is not used

2898 The codec equation is described as follows:

```

2899   sign = Xenco( n ) & 0x1
2900   value = 1
2901   if (sign > 0) then
2902       Xdeco( n ) = Xpred( n ) - value
2903   else
2904       Xdeco( n ) = Xpred( n ) + value
2905   endif

```


10-Nov-2016

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 + 1if (**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 + 3if (**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 xxxxxx"
```

where,

"1" is the code word

the **sign** bit is not used

"xxxxxx" 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
```

10-Nov-2016

E.3.4 Decoder for 12–10–12 Data Compression

2979 Pixels without prediction are decoded using the following formula:

2980 $\text{Xdeco}(n) = 4 * \text{Xenco}(n) + 2$

2981 Pixels with prediction are decoded using the following formula:

```

2982   if (Xenco( n ) & 0x300 == 0x000) then
2983       use DPCM1
2984   else if (Xenco( n ) & 0x380 == 0x100) then
2985       use DPCM2
2986   else if (Xenco( n ) & 0x380 == 0x180) then
2987       use DPCM3
2988   else
2989       use PCM
2990   endif

```

E.3.4.1 DPCM1 for 12–10–12 Decoder

2991 $\text{Xenco}(n)$ has the following format:

2992 $\text{Xenco}(n) = "00 \text{ s } \text{xxxxxxx}"$

2993 where,

2994 "00" is the code word

2995 "s" is the **sign** bit

2996 "xxxxxxx" is the seven bit **value** field

2997 The decoder equation is described as follows:

```

2998   sign = Xenco( n ) & 0x80
2999   value = Xenco( n ) & 0x7f
3000   if (sign > 0) then
3001       Xdeco( n ) = Xpred( n ) - value
3002   else
3003       Xdeco( n ) = Xpred( n ) + value
3004   endif

```

E.3.4.2 DPCM2 for 12–10–12 Decoder

3005 **Xenco(n)** has the following format:

3006 **Xenco(n)** = "010 s xxxxxx"

3007 where,

3008 "010" is the code word

3009 "s" is the **sign** bit

3010 "xxxxxx" is the six bit **value** field

3011 The decoder equation is described as follows:

3012 **sign** = **Xenco(n)** & 0x40

3013 **value** = 2 * (**Xenco(n)** & 0x3f) + 128

3014 if (**sign** > 0) then

3015 **Xdeco(n)** = **Xpred(n)** - **value**

3016 else

3017 **Xdeco(n)** = **Xpred(n)** + **value**

3018 endif

E.3.4.3 DPCM3 for 12–10–12 Decoder

3019 **Xenco(n)** has the following format:

3020 **Xenco(n)** = "011 s xxxxxx"

3021 where,

3022 "011" is the code word

3023 "s" is the **sign** bit

3024 "xxxxxx" is the six bit **value** field

3025 The decoder equation is described as follows:

3026 **sign** = **Xenco(n)** & 0x40

3027 **value** = 4 * (**Xenco(n)** & 0x3f) + 256 + 1

3028 if (**sign** > 0) then

3029 **Xdeco(n)** = **Xpred(n)** - **value**

3030 if (**Xdeco(n)** < 0) then

3031 **Xdeco(n)** = 0

3032 endif

3033 else

3034 **Xdeco(n)** = **Xpred(n)** + **value**

3035 if (**Xdeco(n)** > 4095) then

3036 **Xdeco(n)** = 4095

3037 endif

3038 endif

10-Nov-2016

E.3.4.4 PCM for 12–10–12 Decoder

3039 **Xenco(n)** has the following format:

3040 **Xenco(n)** = "1 xxxxxxxxx"

3041 where,

3042 "1" is the code word

3043 the **sign** bit is not used

3044 "xxxxxxxxx" is the nine bit **value** field

3045 The codec equation is described as follows:

3046 **value** = 8 * (**Xenco(n)** & 0x1ff)

3047 if (**value** > **Xpred(n)**) then

3048 **Xdeco(n)** = **value** + 3

3049 endif

3050 else

3051 **Xdeco(n)** = **value** + 4

3052 endif

E.3.5 Decoder for 12–8–12 Data Compression

3053 Pixels without prediction are decoded using the following formula:

3054 $\text{Xdeco}(n) = 16 * \text{Xenco}(n) + 8$

3055 Pixels with prediction are decoded using the following formula:

```

3056   if (Xenco( n ) & 0xf0 == 0x00) then
3057       use DPCM1
3058   else if (Xenco( n ) & 0xe0 == 0x60) then
3059       use DPCM2
3060   else if (Xenco( n ) & 0xe0 == 0x40) then
3061       use DPCM3
3062   else if (Xenco( n ) & 0xe0 == 0x20) then
3063       use DPCM4
3064   else if (Xenco( n ) & 0xf0 == 0x10) then
3065       use DPCM5
3066   else
3067       use PCM
3068   endif

```

E.3.5.1 DPCM1 for 12–8–12 Decoder

3069 **Xenco(n)** has the following format:

3070 $\text{Xenco}(n) = \text{"0000 s xxx"}$

3071 where,

3072 "0000" is the code word

3073 "s" is the **sign** bit

3074 "xxx" is the three bit **value** field

3075 The codec equation is described as follows:

```

3076   sign = Xenco( n ) & 0x8
3077   value = Xenco( n ) & 0x7
3078   if (sign > 0) then
3079       Xdeco( n ) = Xpred( n ) - value
3080   else
3081       Xdeco( n ) = Xpred( n ) + value
3082   endif

```

E.3.5.2 DPCM2 for 12–8–12 Decoder

3083 **Xenco(n)** has the following format:

3084 $\text{Xenco}(n) = \text{"011 s xxxx"}$

3085 where,

3086 "011" is the code word

3087 "s" is the **sign** bit

3088 "xxxx" is the four bit **value** field

3089 The codec equation is described as follows:

```

3090   sign = Xenco( n ) & 0x10
3091   value = 2 * (Xenco( n ) & 0xf) + 8
3092   if (sign > 0) then
3093       Xdeco( n ) = Xpred( n ) - value
3094   else
3095       Xdeco( n ) = Xpred( n ) + value
3096   endif

```

10-Nov-2016

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

3137 **Xenco(n)** has the following format:

3138 **Xenco(n)** = "0001 s xxx"

3139 where,

3140 "0001" is the code word

3141 "s" is the **sign** bit

3142 "xxx" is the three bit **value** field

3143 The codec equation is described as follows:

```

3144   sign = Xenco( n ) & 0x8
3145   value = 16 * (Xenco( n ) & 0x7) + 232 + 7
3146   if (sign > 0) then
3147       Xdeco( n ) = Xpred( n ) - value
3148       if (Xdeco( n ) < 0) then
3149           Xdeco( n ) = 0
3150       endif
3151   else
3152       Xdeco( n ) = Xpred( n ) + value
3153       if (Xdeco( n ) > 4095) then
3154           Xdeco( n ) = 4095
3155       endif
3156   endif

```

E.3.5.6 PCM for 12–8–12 Decoder

3157 **Xenco(n)** has the following format:

3158 **Xenco(n)** = "1 xxxxxxx"

3159 where,

3160 "1" is the code word

3161 the **sign** bit is not used

3162 "xxxxxxx" is the seven bit **value** field

3163 The codec equation is described as follows:

```

3164   value = 32 * (Xenco( n ) & 0x7f)
3165   if (value > Xpred( n )) then
3166       Xdeco( n ) = value + 15
3167   else
3168       Xdeco( n ) = value + 16
3169   endif

```


10-Nov-2016

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

3202 **Xenco(n)** has the following format:

3203 **Xenco(n)** = "0001 s xx"

3204 where,

3205 "0001" is the code word

3206 "s" is the **sign** bit

3207 "xx" is the two bit **value** field

3208 The codec equation is described as follows:

3209 **sign** = **Xenco(n)** & 0x4

3210 **value** = 2 * (**Xenco(n)** & 0x3) + 4

3211 if (**sign** > 0) then

3212 **Xdeco(n)** = **Xpred(n)** - **value**

3213 else

3214 **Xdeco(n)** = **Xpred(n)** + **value**

3215 endif

E.3.6.3 DPCM3 for 12–7–12 Decoder

3216 **Xenco(n)** has the following format:

3217 **Xenco(n)** = "0010 s xx"

3218 where,

3219 "0010" is the code word

3220 "s" is the **sign** bit

3221 "xx" is the two bit **value** field

3222 The codec equation is described as follows:

3223 **sign** = **Xenco(n)** & 0x4

3224 **value** = 4 * (**Xenco(n)** & 0x3) + 12 + 1

3225 if (**sign** > 0) then

3226 **Xdeco(n)** = **Xpred(n)** - **value**

3227 if (**Xdeco(n)** < 0) then

3228 **Xdeco(n)** = 0

3229 endif

3230 else

3231 **Xdeco(n)** = **Xpred(n)** + **value**

3232 if (**Xdeco(n)** > 4095) then

3233 **Xdeco(n)** = 4095

3234 endif

3235 endif

10-Nov-2016

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 + 3if (**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 + 7if (**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

3276 **Xenco(n)** has the following format:

3277 **Xenco(n)** = "0011 s xx"

3278 where,

3279 "0011" is the code word

3280 "s" is the **sign** bit

3281 "xx" is the two bit **value** field

3282 The codec equation is described as follows:

3283 **sign** = **Xenco(n)** & 0x4

3284 **value** = 32 * (**Xenco(n)** & 0x3) + 220 + 15

3285 if (**sign** > 0) then

3286 **Xdeco(n)** = **Xpred(n)** - **value**

3287 if (**Xdeco(n)** < 0) then

3288 **Xdeco(n)** = 0

3289 endif

3290 else

3291 **Xdeco(n)** = **Xpred(n)** + **value**

3292 if (**Xdeco(n)** > 4095) then

3293 **Xdeco(n)** = 4095

3294 endif

3295 endif

E.3.6.7 PCM for 12–7–12 Decoder

3296 **Xenco(n)** has the following format:

3297 **Xenco(n)** = "1 xxxxxxx"

3298 where,

3299 "1" is the code word

3300 the **sign** bit is not used

3301 "xxxxxxx" is the six bit **value** field

3302 The codec equation is described as follows:

3303 **value** = 64 * (**Xenco(n)** & 0x3f)

3304 if (**value** > **Xpred(n)**) then

3305 **Xdeco(n)** = **value** + 31

3306 else

3307 **Xdeco(n)** = **value** + 32

3308 endif

10-Nov-2016

E.3.7 Decoder for 12–6–12 Data Compression

3309 Pixels without prediction are decoded using the following formula:

3310 $\text{Xdeco}(n) = 64 * \text{Xenco}(n) + 32$

3311 Pixels with prediction are decoded using the following formula:

```

3312   if (Xenco( n ) & 0x3c == 0x00) then
3313       use DPCM1
3314   else if (Xenco( n ) & 0x3c == 0x04) then
3315       use DPCM3
3316   else if (Xenco( n ) & 0x38 == 0x10) then
3317       use DPCM4
3318   else if (Xenco( n ) & 0x3c == 0x08) then
3319       use DPCM5
3320   else if (Xenco( n ) & 0x38 == 0x18) then
3321       use DPCM6
3322   else if (Xenco( n ) & 0x3c == 0x0c) then
3323       use DPCM7
3324   else
3325       use PCM
3326   endif

```

3327 *Note: DPCM2 is not used.*

E.3.7.1 DPCM1 for 12–6–12 Decoder

3328 $\text{Xenco}(n)$ has the following format:

3329 $\text{Xenco}(n) = \text{"0000 s x"}$

3330 where,

```

3331   "0000" is the code word
3332   "s" is the sign bit
3333   "x" is the one bit value field

```

3334 The codec equation is described as follows:

```

3335   sign = Xenco( n ) & 0x2
3336   value = Xenco( n ) & 0x1
3337   if (sign > 0) then
3338       Xdeco( n ) = Xpred( n ) - value
3339   else
3340       Xdeco( n ) = Xpred( n ) + value
3341   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

```

10-Nov-2016

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 xxxxxx"
```

where,

"1" is the code word

the **sign** bit is not used

"xxxxxx" 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 177 illustrates interleaving JPEG image data with YUV422 image data using Data Type values.

Figure 178 illustrates interleaving JPEG image data with YUV422 image data using both Data Type values and Virtual Channel Identifiers.

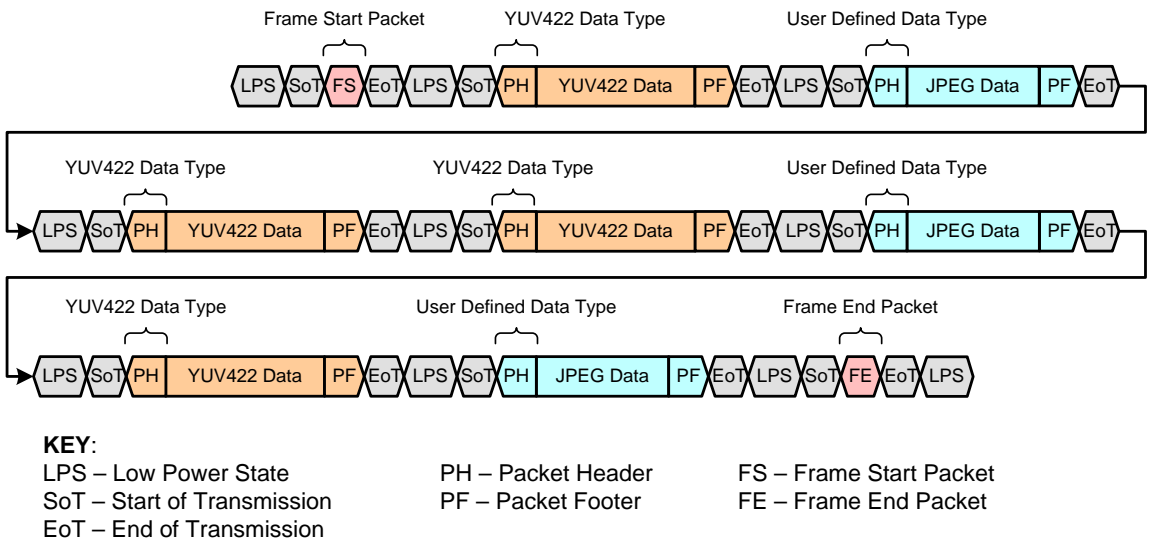


Figure 177 Data Type Interleaving: Concurrent JPEG and YUV Image Data

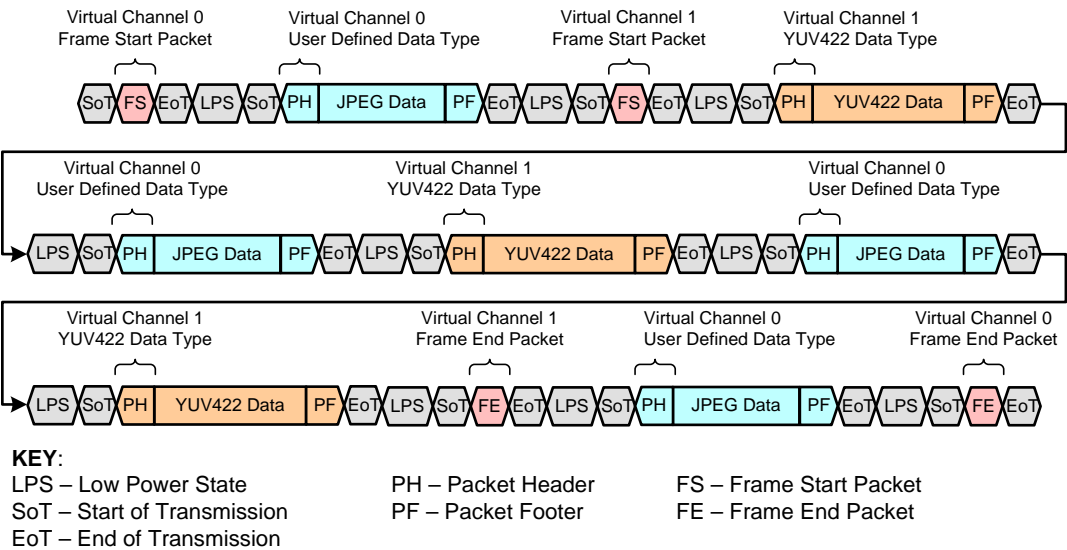


Figure 178 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data

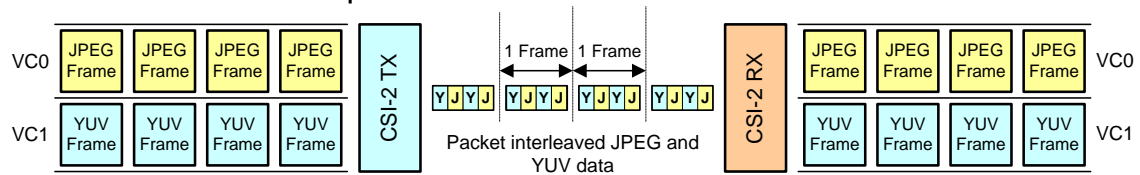
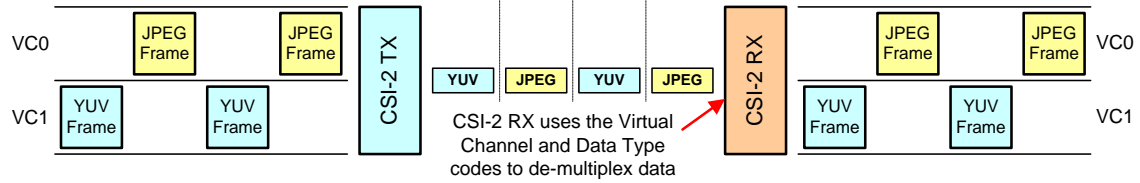
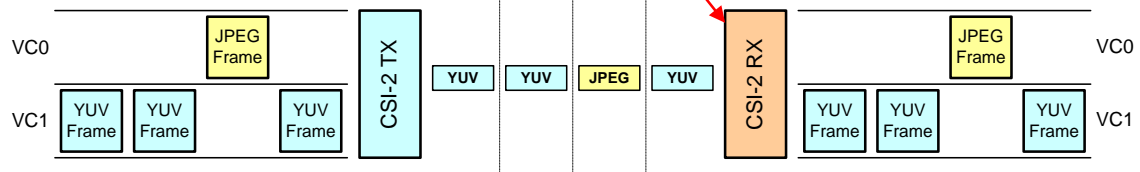
Both *Figure 177* and *Figure 178* can be similarly extended to the interleaving of JPEG image data with any other type of image data, e.g. RGB565.

Figure 179 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.

10-Nov-2016

Use Case 1: Concurrent JPEG output with YUV data**Use Case 2: Alternating JPEG and YUV output – one frame JPEG, then one frame YUV****Use Case 3: Streaming YUV with occasional JPEG still capture****Figure 179 Example JPEG and YUV Interleaving Use Cases**

3483

This page intentionally left blank

10-Nov-2016

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 41*.

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 41*.)

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 41 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).