



## **Specification for Camera Serial Interface 2 (CSI-2)**

**Version 1.2**

**22 January 2014**

MIPI Board Adopted 10 September 2014

This document is subject to further editorial and technical development.

## **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

## Contents

3	Contents .....	iii
4	Tables .....	xii
5	Release History.....	xiii
6	1 Overview .....	1
7	1.1 Scope .....	1
8	1.2 Purpose .....	1
9	2 Terminology .....	2
10	2.1 Definitions .....	2
11	2.2 Abbreviations .....	2
12	2.3 Acronyms .....	3
13	3 References .....	5
14	4 Overview of CSI-2 .....	6
15	5 CSI-2 Layer Definitions .....	7
16	6 Camera Control Interface (CCI).....	9
17	6.1 Data Transfer Protocol .....	9
18	6.1.1 Message Type .....	9
19	6.1.2 Read/Write Operations .....	10
20	6.2 CCI Slave Addresses .....	13
21	6.3 CCI Multi-Byte Registers.....	13
22	6.3.1 Overview .....	13
23	6.3.2 The Transmission Byte Order for Multi-byte Register Values.....	15
24	6.3.3 Multi-Byte Register Protocol.....	16
25	6.4 Electrical Specifications and Timing for I/O Stages .....	20
26	7 Physical Layer .....	23
27	8 Multi-Lane Distribution and Merging .....	24
28	8.1 Multi-Lane Interoperability .....	29
29	9 Low Level Protocol .....	32
30	9.1 Low Level Protocol Packet Format .....	32
31	9.1.1 Low Level Protocol Long Packet Format .....	32
32	9.1.2 Low Level Protocol Short Packet Format.....	34
33	9.2 Data Identifier (DI).....	34
34	9.3 Virtual Channel Identifier.....	34
35	9.4 Data Type (DT) .....	35
36	9.5 Packet Header Error Correction Code .....	36
37	9.5.1 General Hamming Code Applied to Packet Header .....	37

38	9.5.2	Hamming-Modified Code.....	37
39	9.5.3	ECC Generation on TX Side .....	40
40	9.5.4	Applying ECC on RX Side.....	41
41	9.6	Checksum Generation .....	42
42	9.7	Packet Spacing .....	44
43	9.8	Synchronization Short Packet Data Type Codes .....	45
44	9.8.1	Frame Synchronization Packets.....	45
45	9.8.2	Line Synchronization Packets.....	46
46	9.9	Generic Short Packet Data Type Codes.....	46
47	9.10	Packet Spacing Examples.....	47
48	9.11	Packet Data Payload Size Rules .....	49
49	9.12	Frame Format Examples.....	50
50	9.13	Data Interleaving .....	52
51	9.13.1	Data Type Interleaving .....	52
52	9.13.2	Virtual Channel Identifier Interleaving.....	55
53	10	Color Spaces.....	57
54	10.1	RGB Color Space Definition.....	57
55	10.2	YUV Color Space Definition .....	57
56	11	Data Formats .....	58
57	11.1	Generic 8-bit Long Packet Data Types.....	59
58	11.1.1	Null and Blanking Data .....	59
59	11.1.2	Embedded Information .....	59
60	11.2	YUV Image Data.....	60
61	11.2.1	Legacy YUV420 8-bit .....	61
62	11.2.2	YUV420 8-bit.....	63
63	11.2.3	YUV420 10-bit.....	65
64	11.2.4	YUV422 8-bit.....	67
65	11.2.5	YUV422 10-bit.....	69
66	11.3	RGB Image Data .....	70
67	11.3.1	RGB888.....	71
68	11.3.2	RGB666.....	72
69	11.3.3	RGB565.....	73
70	11.3.4	RGB555.....	74
71	11.3.5	RGB444.....	75
72	11.4	RAW Image Data .....	75
73	11.4.1	RAW6.....	76
74	11.4.2	RAW7.....	77

75	11.4.3	RAW8.....	78
76	11.4.4	RAW10.....	79
77	11.4.5	RAW12.....	80
78	11.4.6	RAW14.....	81
79	11.5	User Defined Data Formats .....	82
80	12	Recommended Memory Storage .....	84
81	12.1	General/Arbitrary Data Reception.....	84
82	12.2	RGB888 Data Reception .....	85
83	12.3	RGB666 Data Reception .....	85
84	12.4	RGB565 Data Reception .....	86
85	12.5	RGB555 Data Reception .....	86
86	12.6	RGB444 Data Reception .....	87
87	12.7	YUV422 8-bit Data Reception .....	87
88	12.8	YUV422 10-bit Data Reception .....	88
89	12.9	YUV420 8-bit (Legacy) Data Reception.....	88
90	12.10	YUV420 8-bit Data Reception .....	90
91	12.11	YUV420 10-bit Data Reception .....	91
92	12.12	RAW6 Data Reception .....	92
93	12.13	RAW7 Data Reception .....	92
94	12.14	RAW8 Data Reception .....	93
95	12.15	RAW10 Data Reception .....	93
96	12.16	RAW12 Data Reception .....	94
97	12.17	RAW14 Data Reception .....	94
98	Annex A	JPEG8 Data Format (informative).....	95
99	A.1	Introduction .....	95
100	A.2	JPEG Data Definition .....	96
101	A.3	Image Status Information .....	97
102	A.4	Embedded Images .....	98
103	A.5	JPEG8 Non-standard Markers .....	99
104	A.6	JPEG8 Data Reception .....	100
105	Annex B	CSI-2 Implementation Example (informative) .....	101
106	B.1	Overview .....	101
107	B.2	CSI-2 Transmitter Detailed Block Diagram .....	101
108	B.3	CSI-2 Receiver Detailed Block Diagram .....	102
109	B.4	Details on the D-PHY implementation.....	103
110	B.4.1	CSI-2 Clock Lane Transmitter.....	105
111	B.4.2	CSI-2 Clock Lane Receiver .....	106

112	B.4.3	CSI-2 Data Lane Transmitter.....	107
113	B.4.4	CSI-2 Data Lane Receiver .....	108
114	Annex C	CSI-2 Recommended Receiver Error Behavior (informative).....	111
115	C.1	Overview .....	111
116	C.2	D-PHY Level Error .....	111
117	C.3	Packet Level Error.....	112
118	C.4	Protocol Decoding Level Error.....	113
119	Annex D	CSI-2 Sleep Mode (informative) .....	114
120	D.1	Overview .....	114
121	D.2	SLM Command Phase.....	114
122	D.3	SLM Entry Phase.....	114
123	D.4	SLM Exit Phase.....	115
124	Annex E	Data Compression for RAW Data Types (normative).....	116
125	E.1	Predictors.....	117
126	E.1.1	Predictor1 .....	118
127	E.1.2	Predictor2 .....	118
128	E.2	Encoders .....	119
129	E.2.1	Coder for 10–8–10 Data Compression .....	119
130	E.2.2	Coder for 10–7–10 Data Compression .....	121
131	E.2.3	Coder for 10–6–10 Data Compression .....	123
132	E.2.4	Coder for 12–8–12 Data Compression .....	126
133	E.2.5	Coder for 12–7–12 Data Compression .....	129
134	E.2.6	Coder for 12–6–12 Data Compression .....	132
135	E.3	Decoders.....	135
136	E.3.1	Decoder for 10–8–10 Data Compression.....	136
137	E.3.2	Decoder for 10–7–10 Data Compression.....	138
138	E.3.3	Decoder for 10–6–10 Data Compression.....	140
139	E.3.4	Decoder for 12–8–12 Data Compression.....	143
140	E.3.5	Decoder for 12–7–12 Data Compression.....	146
141	E.3.6	Decoder for 12–6–12 Data Compression.....	150
142	Annex F	JPEG Interleaving (informative).....	155

## 143 Figures

144	Figure 1 CSI-2 and CCI Transmitter and Receiver Interface .....	6
145	Figure 2 CSI-2 Layer Definitions .....	7
146	Figure 3 CCI Message Types .....	10
147	Figure 4 CCI Single Read from Random Location .....	10
148	Figure 5 CCI Single Read from Current Location.....	11
149	Figure 6 CCI Sequential Read Starting from a Random Location .....	11
150	Figure 7 CCI Sequential Read Starting from the Current Location.....	12
151	Figure 8 CCI Single Write to a Random Location .....	12
152	Figure 9 CCI Sequential Write Starting from a Random Location.....	13
153	Figure 10 Corruption of a 32-bit Wide Register during a Read Message.....	14
154	Figure 11 Corruption of a 32-bit Wide Register during a Write Message.....	15
155	Figure 12 Example 16-bit Register Write.....	15
156	Figure 13 Example 32-bit Register Write (address not shown).....	16
157	Figure 14 Example 64-bit Register Write (address not shown).....	16
158	Figure 15 Example 16-bit Register Read.....	17
159	Figure 16 Example 32-bit Register Read.....	18
160	Figure 17 Example 16-bit Register Write.....	19
161	Figure 18 Example 32-bit Register Write.....	20
162	Figure 19 CCI Timing .....	22
163	Figure 20 Conceptual Overview of the Lane Distributor Function .....	24
164	Figure 21 Conceptual Overview of the Lane Merging Function .....	25
165	Figure 22 Two Lane Multi-Lane Example .....	26
166	Figure 23 Three Lane Multi-Lane Example .....	27
167	Figure 24 N Lane Multi-Lane Example.....	28
168	Figure 25 N Lane Multi-Lane Example for Short Packet Transmission .....	29
169	Figure 26 One Lane Transmitter and N Lane Receiver Example.....	30
170	Figure 27 M Lane Transmitter and N Lane Receiver Example (M<N).....	30
171	Figure 28 M Lane Transmitter and One Lane Receiver Example .....	30
172	Figure 29 M Lane Transmitter and N Lane Receiver Example (N<M).....	31
173	Figure 30 Low Level Protocol Packet Overview.....	32
174	Figure 31 Long Packet Structure .....	33
175	Figure 32 Short Packet Structure.....	34
176	Figure 33 Data Identifier Byte.....	34
177	Figure 34 Logical Channel Block Diagram (Receiver) .....	35
178	Figure 35 Interleaved Video Data Streams Examples .....	35

179	Figure 36 24-bit ECC Generation Example.....	36
180	Figure 37 64-bit ECC Generation on TX Side .....	40
181	Figure 38 24-bit ECC Generation on TX Side .....	40
182	Figure 39 64-bit ECC on RX Side Including Error Correction .....	41
183	Figure 40 24-bit ECC on RX Side Including Error Correction .....	42
184	Figure 41 Checksum Transmission .....	42
185	Figure 42 Checksum Generation for Packet Data .....	43
186	Figure 43 Definition of 16-bit CRC Shift Register.....	43
187	Figure 44 16-bit CRC Software Implementation Example.....	44
188	Figure 45 Packet Spacing .....	45
189	Figure 46 Multiple Packet Example .....	47
190	Figure 47 Single Packet Example.....	47
191	Figure 48 Line and Frame Blanking Definitions .....	48
192	Figure 49 Vertical Sync Example.....	49
193	Figure 50 Horizontal Sync Example .....	49
194	Figure 51 General Frame Format Example .....	50
195	Figure 52 Digital Interlaced Video Example.....	51
196	Figure 53 Digital Interlaced Video with Accurate Synchronization Timing Information .....	52
197	Figure 54 Interleaved Data Transmission using Data Type Value .....	53
198	Figure 55 Packet Level Interleaved Data Transmission .....	54
199	Figure 56 Frame Level Interleaved Data Transmission.....	55
200	Figure 57 Interleaved Data Transmission using Virtual Channels .....	56
201	Figure 58 Frame Structure with Embedded Data at the Beginning and End of the Frame.....	60
202	Figure 59 Legacy YUV420 8-bit Transmission .....	61
203	Figure 60 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration .....	62
204	Figure 61 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1 .....	62
205	Figure 62 Legacy YUV420 8-bit Frame Format .....	63
206	Figure 63 YUV420 8-bit Data Transmission Sequence .....	63
207	Figure 64 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration .....	64
208	Figure 65 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1 .....	64
209	Figure 66 YUV420 Spatial Sampling for MPEG 2 and MPEG 4 .....	65
210	Figure 67 YUV420 8-bit Frame Format.....	65
211	Figure 68 YUV420 10-bit Transmission .....	66
212	Figure 69 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration .....	66
213	Figure 70 YUV420 10-bit Frame Format .....	67
214	Figure 71 YUV422 8-bit Transmission .....	67
215	Figure 72 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration .....	68

216	Figure 73 YUV422 Co-sited Spatial Sampling .....	68
217	Figure 74 YUV422 8-bit Frame Format.....	69
218	Figure 75 YUV422 10-bit Transmitted Bytes .....	69
219	Figure 76 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration .....	70
220	Figure 77 YUV422 10-bit Frame Format.....	70
221	Figure 78 RGB888 Transmission .....	71
222	Figure 79 RGB888 Transmission in CSI-2 Bus Bitwise Illustration.....	71
223	Figure 80 RGB888 Frame Format.....	72
224	Figure 81 RGB666 Transmission with 18-bit BGR Words.....	72
225	Figure 82 RGB666 Transmission on CSI-2 Bus Bitwise Illustration.....	72
226	Figure 83 RGB666 Frame Format.....	73
227	Figure 84 RGB565 Transmission with 16-bit BGR Words.....	73
228	Figure 85 RGB565 Transmission on CSI-2 Bus Bitwise Illustration.....	74
229	Figure 86 RGB565 Frame Format.....	74
230	Figure 87 RGB555 Transmission on CSI-2 Bus Bitwise Illustration .....	74
231	Figure 88 RGB444 Transmission on CSI-2 Bus Bitwise Illustration .....	75
232	Figure 89 RAW6 Transmission .....	76
233	Figure 90 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration .....	76
234	Figure 91 RAW6 Frame Format.....	77
235	Figure 92 RAW7 Transmission .....	77
236	Figure 93 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration .....	77
237	Figure 94 RAW7 Frame Format.....	78
238	Figure 95 RAW8 Transmission .....	78
239	Figure 96 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration .....	78
240	Figure 97 RAW8 Frame Format.....	79
241	Figure 98 RAW10 Transmission .....	79
242	Figure 99 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration .....	79
243	Figure 100 RAW10 Frame Format.....	80
244	Figure 101 RAW12 Transmission .....	80
245	Figure 102 RAW12 Transmission on CSI-2 Bus Bitwise Illustration .....	80
246	Figure 103 RAW12 Frame Format.....	81
247	Figure 104 RAW14 Transmission .....	81
248	Figure 105 RAW14 Transmission on CSI-2 Bus Bitwise Illustration .....	81
249	Figure 106 RAW14 Frame Format.....	82
250	Figure 107 User Defined 8-bit Data (128 Byte Packet).....	82
251	Figure 108 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration .....	82
252	Figure 109 Transmission of User Defined 8-bit Data.....	83

253	Figure 110 General/Arbitrary Data Reception.....	84
254	Figure 111 RGB888 Data Format Reception .....	85
255	Figure 112 RGB666 Data Format Reception .....	85
256	Figure 113 RGB565 Data Format Reception .....	86
257	Figure 114 RGB555 Data Format Reception .....	86
258	Figure 115 RGB444 Data Format Reception .....	87
259	Figure 116 YUV422 8-bit Data Format Reception .....	87
260	Figure 117 YUV422 10-bit Data Format Reception .....	88
261	Figure 118 YUV420 8-bit Legacy Data Format Reception.....	89
262	Figure 119 YUV420 8-bit Data Format Reception .....	90
263	Figure 120 YUV420 10-bit Data Format Reception .....	91
264	Figure 121 RAW6 Data Format Reception .....	92
265	Figure 122 RAW7 Data Format Reception .....	92
266	Figure 123 RAW8 Data Format Reception .....	93
267	Figure 124 RAW10 Data Format Reception .....	93
268	Figure 125 RAW12 Data Format Reception .....	94
269	Figure 126 RAW 14 Data Format Reception .....	94
270	Figure 127 JPEG8 Data Flow in the Encoder.....	95
271	Figure 128 JPEG8 Data Flow in the Decoder .....	96
272	Figure 129 EXIF Compatible Baseline JPEG DCT Format .....	97
273	Figure 130 Status Information Field in the End of Baseline JPEG Frame .....	98
274	Figure 131 Example of TN Image Embedding Inside the Compressed JPEG Data Block .....	99
275	Figure 132 JPEG8 Data Format Reception .....	100
276	Figure 133 Implementation Example Block Diagram and Coverage .....	101
277	Figure 134 CSI-2 Transmitter Block Diagram .....	102
278	Figure 135 CSI-2 Receiver Block Diagram.....	103
279	Figure 136 D-PHY Level Block Diagram .....	104
280	Figure 137 CSI-2 Clock Lane Transmitter .....	105
281	Figure 138 CSI-2 Clock Lane Receiver.....	106
282	Figure 139 CSI-2 Data Lane Transmitter .....	107
283	Figure 140 CSI-2 Data Lane Receiver.....	109
284	Figure 141 SLM Synchronization .....	115
285	Figure 142 Data Compression System Block Diagram .....	117
286	Figure 143 Pixel Order of the Original Image.....	117
287	Figure 144 Example Pixel Order of the Original Image.....	117
288	Figure 145 Data Type Interleaving: Concurrent JPEG and YUV Image Data .....	155
289	Figure 146 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data .....	156

290	Figure 147 Example JPEG and YUV Interleaving Use Cases .....	157
-----	--	-----

## Tables

292	Table 1 CCI I/O Characteristics .....	20
293	Table 2 CCI Timing Specification.....	21
294	Table 3 Data Type Classes .....	36
295	Table 4 ECC Syndrome Association Matrix .....	37
296	Table 5 ECC Parity Generation Rules .....	38
297	Table 6 Synchronization Short Packet Data Type Codes .....	45
298	Table 7 Generic Short Packet Data Type Codes.....	46
299	Table 8 Primary and Secondary Data Formats Definitions .....	58
300	Table 9 Generic 8-bit Long Packet Data Types.....	59
301	Table 10 YUV Image Data Types .....	61
302	Table 11 Legacy YUV420 8-bit Packet Data Size Constraints .....	61
303	Table 12 YUV420 8-bit Packet Data Size Constraints.....	63
304	Table 13 YUV420 10-bit Packet Data Size Constraints.....	66
305	Table 14 YUV422 8-bit Packet Data Size Constraints .....	67
306	Table 15 YUV422 10-bit Packet Data Size Constraints.....	69
307	Table 16 RGB Image Data Types.....	71
308	Table 17 RGB888 Packet Data Size Constraints.....	71
309	Table 18 RGB666 Packet Data Size Constraints.....	72
310	Table 19 RGB565 Packet Data Size Constraints.....	73
311	Table 20 RAW Image Data Types .....	76
312	Table 21 RAW6 Packet Data Size Constraints.....	76
313	Table 22 RAW7 Packet Data Size Constraints.....	77
314	Table 23 RAW8 Packet Data Size Constraints.....	78
315	Table 24 RAW10 Packet Data Size Constraints.....	79
316	Table 25 RAW12 Packet Data Size Constraints.....	80
317	Table 26 RAW14 Packet Data Size Constraints.....	81
318	Table 27 User Defined 8-bit Data Types.....	83
319	Table 28 Status Data Padding.....	98
320	Table 29 JPEG8 Additional Marker Codes Listing .....	100

321

## Release History

Date	Release	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.

322 **1 Overview**

323 **1.1 Scope**

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

327 This Revision of the Camera Serial Interface 2 Specification leverages the improved skew tolerance and  
328 higher data rate of the [MIPI01] D-PHY 1.2 Specification. These enhancements enable higher interface  
329 bandwidth and more flexibility in channel layout.

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

335 **1.2 Purpose**

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

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

345 **2 Terminology**

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

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

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

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

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

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

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

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

364 **2.1 Definitions**

365 **Lane:** A differential conductor pair, used for data transmission. For CSI-2 a data Lane is unidirectional.

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

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

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

372 **Transmission:** The time during which high-speed serial data is actively traversing the bus. A transmission  
373 is comprised of one or more packets. A transmission is bounded by SoT (Start of Transmission) and EoT  
374 (End of Transmission) at beginning and end, respectively.

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

379 **2.2 Abbreviations**

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

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

382 **2.3 Acronyms**

383 BER Bit Error Rate

384 CCI Camera Control Interface

385 CIL Control and Interface Logic

386 CRC Cyclic Redundancy Check

387 CSI Camera Serial Interface

388 CSPS Chroma Sample Pixel Shifted

389 DDR Dual Data Rate

390 DI Data Identifier

391 DT Data Type

392 ECC Error Correction Code

393 EoT End of Transmission

394 EXIF Exchangeable Image File Format

395 FE Frame End

396 FS Frame Start

397 HS High Speed; identifier for operation mode

398 HS-RX High-Speed Receiver (Low-Swing Differential)

399 HS-TX High-Speed Transmitter (Low-Swing Differential)

400 I2C Inter-Integrated Circuit

401 JFIF JPEG File Interchange Format

402 JPEG Joint Photographic Expert Group

403 LE Line End

404 LLP Low Level Protocol

405 LS Line Start

406 LSB Least Significant Bit

407 LP Low-Power; identifier for operation mode

408	LP-RX	Low-Power Receiver (Large-Swing Single Ended)
409	LP-TX	Low-Power Transmitter (Large-Swing Single Ended)
410	MSB	Most Significant Bit
411	PF	Packet Footer
412	PH	Packet Header
413	PI	Packet Identifier
414	PT	Packet Type
415	PHY	Physical Layer
416	PPI	PHY Protocol Interface
417	RGB	Color representation (Red, Green, Blue)
418	RX	Receiver
419	SCL	Serial Clock (for CCI)
420	SDA	Serial Data (for CCI)
421	SLM	Sleep Mode
422	SoT	Start of Transmission
423	TX	Transmitter
424	ULPS	Ultra-low Power State
425	VGA	Video Graphics Array
426	YUV	Color representation (Y for luminance, U & V for chrominance)

427

### 3 References

428  
429

[NXP01] UM10204, *I2C-bus specification and user manual*, Revision 03, NXP B.V., 19 June 2007.

430  
431

[MIPI01] *MIPI Alliance Specification for D-PHY*, version 1.2, MIPI Alliance, Inc., 10 September 2014.

432

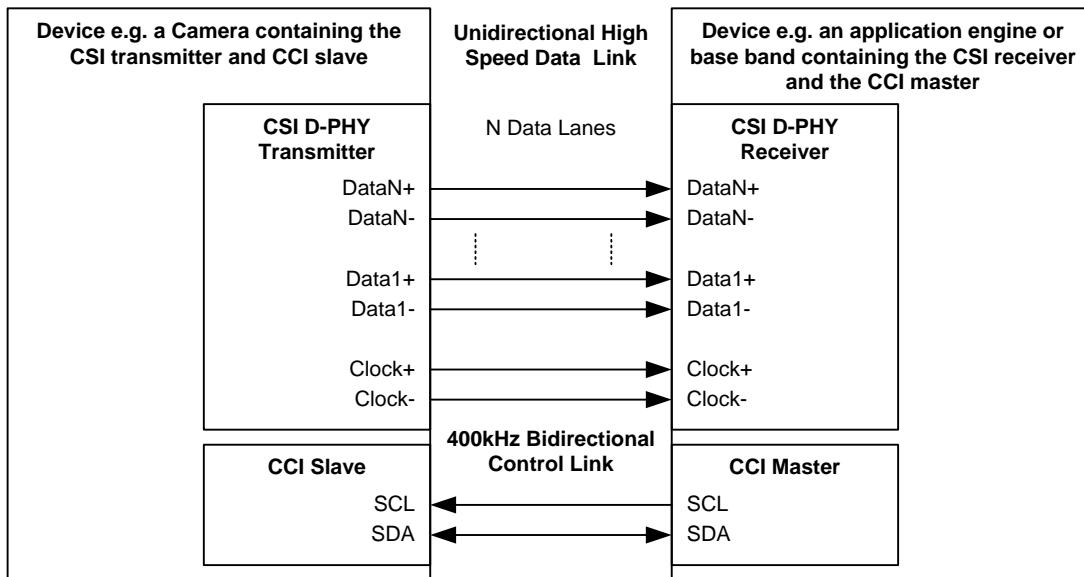
## 4 Overview of CSI-2

433

The CSI-2 Specification defines standard data transmission and control interfaces between transmitter and receiver. Data transmission interface (referred as CSI-2) is unidirectional differential serial interface with data and clock signals; the physical layer of this interface is the *MIPI Alliance Specification for D-PHY* [MIPI01]. Figure 1 illustrates connections between CSI-2 transmitter and receiver, which typically are a camera module and a receiver module, part of the mobile phone engine.

438

The control interface (referred as CCI) is a bi-directional control interface compatible with I2C standard.

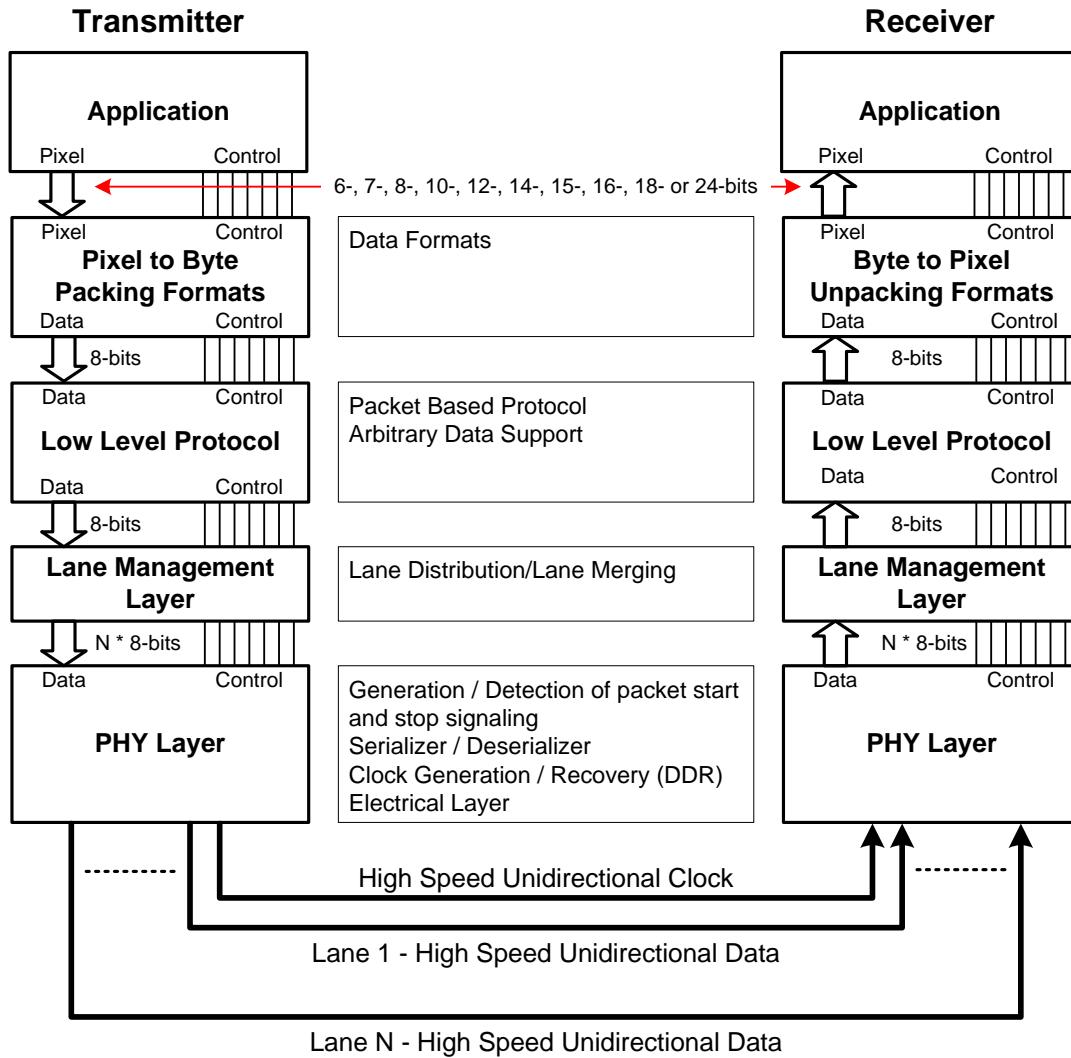


439

440

Figure 1 CSI-2 and CCI Transmitter and Receiver Interface

441 5 CSI-2 Layer Definitions



442  
443      **Figure 2 CSI-2 Layer Definitions**

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

- 445      • **PHY Layer.** The PHY Layer specifies the transmission medium (electrical conductors), the  
446      input/output circuitry and the clocking mechanism that captures “ones” and “zeroes” from the  
447      serial bit stream. This part of the Specification documents the characteristics of the transmission  
448      medium, electrical parameters for signaling and the timing relationship between clock and data  
449      Lanes.

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

454      The PHY layer is described in [MIPI01].

- 455     • **Protocol Layer.** The Protocol layer is composed of several layers, each with distinct  
456        responsibilities. The CSI-2 protocol enables multiple data streams using a single interface on the  
457        host processor. The Protocol layer specifies how multiple data streams may be tagged and  
458        interleaved so each data stream can be properly reconstructed.
  - 459     • **Pixel/Byte Packing/Unpacking Layer.** The CSI-2 supports image applications with varying  
460        pixel formats from six to twenty-four bits per pixels. In the transmitter this layer packs pixels  
461        from the Application layer into bytes before sending the data to the Low Level Protocol layer.  
462        In the receiver this layer unpacks bytes from the Low Level Protocol layer into pixels before  
463        sending the data to the Application layer. Eight bits per pixel data is transferred unchanged by  
464        this layer.
  - 465     • **Low Level Protocol.** The Low Level Protocol (LLP) includes the means of establishing bit-  
466        level and byte-level synchronization for serial data transferred between SoT (Start of  
467        Transmission) and EoT (End of Transmission) events and for passing data to the next layer.  
468        The minimum data granularity of the LLP is one byte. The LLP also includes assignment of  
469        bit-value interpretation within the byte, i.e. the “Endian” assignment.
  - 470     • **Lane Management.** CSI-2 is Lane-scalable for increased performance. The number of data  
471        Lanes may be chosen depending on the bandwidth requirements of the application. The  
472        transmitting side of the interface distributes (“distributor” function) the outgoing data stream  
473        to one or more Lanes. On the receiving side, the interface collects bytes from the Lanes and  
474        merges (“merger” function) them together into a recombined data stream that restores the  
475        original stream sequence.
- 476     Data within the Protocol layer is organized as packets. The transmitting side of the interface appends  
477        header and optional error-checking information on to data to be transmitted at the Low Level  
478        Protocol layer. On the receiving side, the header is stripped off at the Low Level Protocol layer and  
479        interpreted by corresponding logic in the receiver. Error-checking information may be used to test  
480        the integrity of incoming data.
- 481     • **Application Layer.** This layer describes higher-level encoding and interpretation of data  
482        contained in the data stream. The CSI-2 Specification describes the mapping of pixel values to  
483        bytes.
- 484     The normative sections of the Specification only relate to the external part of the Link, e.g. the data and bit  
485        patterns that are transferred across the Link. All internal interfaces and layers are purely informative.

486 **6 Camera Control Interface (CCI)**

487 CCI is a two-wire, bi-directional, half duplex, serial interface for controlling the transmitter. CCI is  
488 compatible with the fast mode variant of the I<sup>2</sup>C interface. CCI shall support 400kHz operation and 7-bit  
489 Slave Addressing.

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

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

496 CCI is a subset of the I<sup>2</sup>C protocol, including the minimum combination of obligatory features for I<sup>2</sup>C slave  
497 devices specified in the I<sup>2</sup>C specification. Therefore, transmitters complying with the CCI specification can  
498 also be connected to the system I<sup>2</sup>C bus. However, care must be taken so that I<sup>2</sup>C masters do not try to utilize  
499 those I<sup>2</sup>C features that are not supported by CCI masters and CCI slaves

500 Each CCI transmitter may have additional features to support I<sup>2</sup>C, but that is dependent on implementation.  
501 Further details can be found on a particular device's data sheet.

502 This Specification does not attempt to define the contents of control messages sent by the CCI master. As  
503 such, it is the responsibility of the CSI-2 implementer to define a set of control messages and corresponding  
504 frame timing and I<sup>2</sup>C latency requirements, if any, that must be met by the CCI master when sending such  
505 control messages to the CCI slave.

506 The CCI defines an additional data protocol layer on top of I<sup>2</sup>C. The data protocol is presented in the  
507 following sections.

508 **6.1 Data Transfer Protocol**

509 The data transfer protocol is according to I<sup>2</sup>C standard. The START, REPEATED START and STOP  
510 conditions as well as data transfer protocol are specified in *The I<sup>2</sup>C Specification* [NXP01].

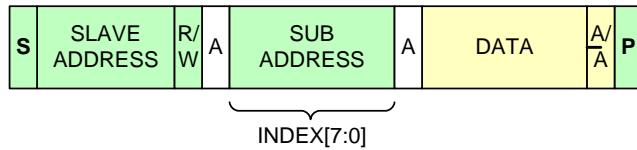
511 **6.1.1 Message Type**

512 A basic CCI message consists of START condition, slave address with read/write bit, acknowledge from  
513 slave, sub address (index) for pointing at a register inside the slave device, acknowledge signal from slave,  
514 in write operation data byte from master, acknowledge/negative acknowledge from slave and STOP  
515 condition. In read operation data byte comes from slave and acknowledge/negative acknowledge from  
516 master. This is illustrated in Figure 3.

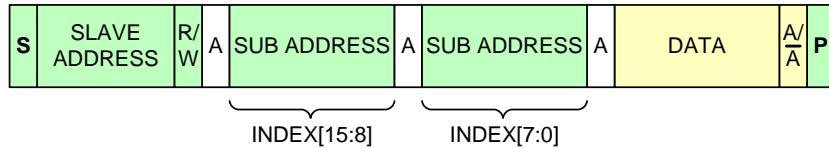
517 The slave address in the CCI is 7-bit.

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

Message type with 8-bit index and 8-bit data (7-bit address)



Message type with 16-bit index and 8-bit data (7-bit address)



- From slave to master      S = START condition      A = Acknowledge
- From master to slave      P = STOP condition       $\bar{A}$  = Negative acknowledge
- Direction dependent on operation

520

521

**Figure 3 CCI Message Types**

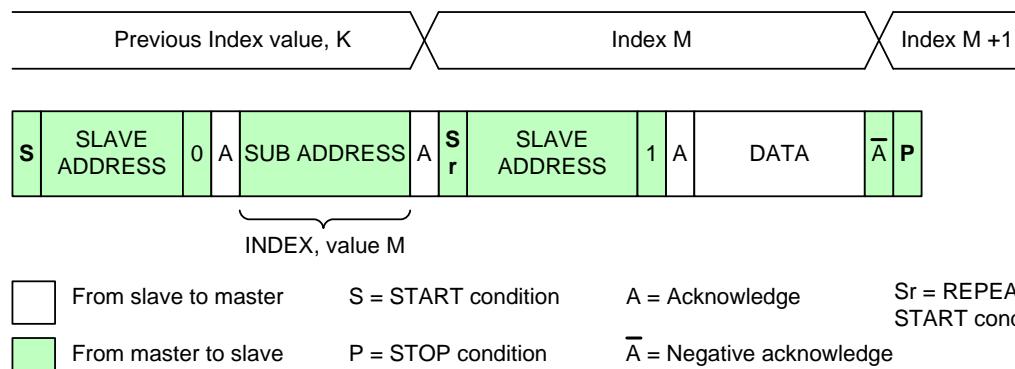
### 522 6.1.2 Read/Write Operations

523 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 524 location, sequential read from current location, single write to random location and sequential write starting 525 from random location. The read/write operations are presented in the following sections.

526 The index in the slave device has to be auto incremented after each read/write operation. This is also 527 explained in the following sections.

#### 529 6.1.2.1 Single Read from Random Location

530 In single read from random location the master does a dummy write operation to desired index, issues a 531 repeated start condition and then addresses the slave again with read operation. After acknowledging its slave 532 address, the slave starts to output data onto SDA line. This is illustrated in Figure 4. The master terminates 533 the read operation by setting a negative acknowledge and stop condition.



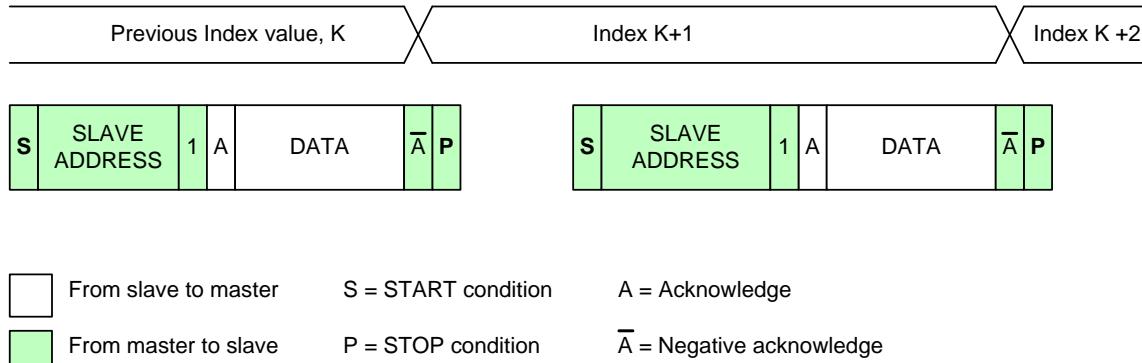
534

535

**Figure 4 CCI Single Read from Random Location**

536 **6.1.2.2 Single Read from the Current Location**

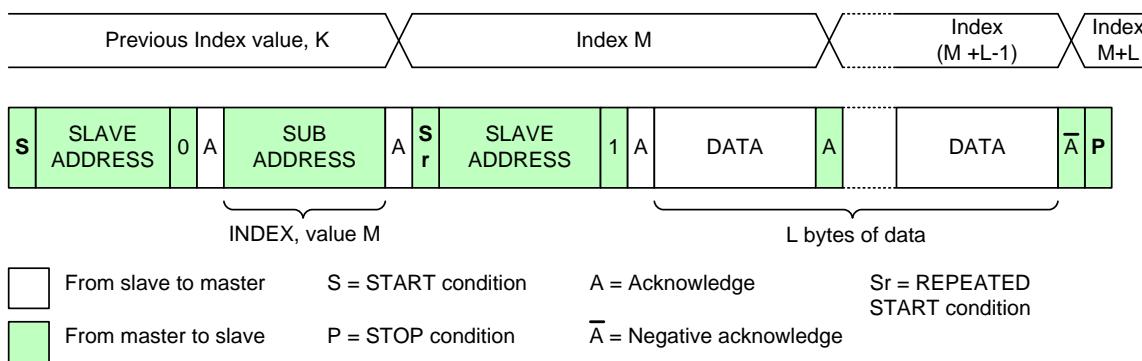
537 It is also possible to read from last used index by addressing the slave with read operation. The slave responds  
538 by setting the data from last used index to SDA line. This is illustrated in Figure 5. The master terminates the  
539 read operation by setting a negative acknowledge and stop condition.



540  
541 **Figure 5 CCI Single Read from Current Location**

542 **6.1.2.3 Sequential Read Starting from a Random Location**

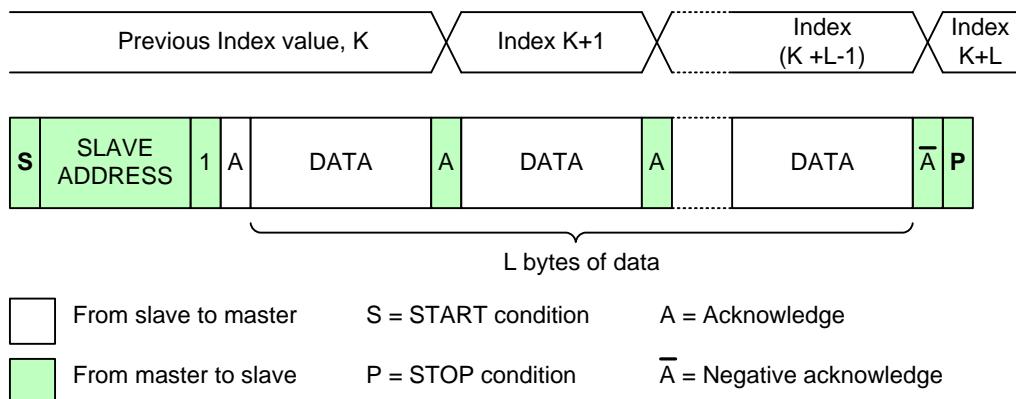
543 The sequential read starting from a random location is illustrated in Figure 6. The master does a dummy write  
544 to the desired index, issues a repeated start condition after an acknowledge from the slave and then addresses  
545 the slave again with a read operation. If a master issues an acknowledge after received data it acts as a signal  
546 to the slave that the read operation continues from the next index. When the master has read the last data byte  
547 it issues a negative acknowledge and stop condition.



548  
549 **Figure 6 CCI Sequential Read Starting from a Random Location**

550 **6.1.2.4 Sequential Read Starting from the Current Location**

552 A sequential read starting from the current location is similar to a sequential read from a random location.  
553 The only exception is there is no dummy write operation. The command sequence is illustrated in Figure 7.  
554 The master terminates the read operation by issuing a negative acknowledge and stop condition.

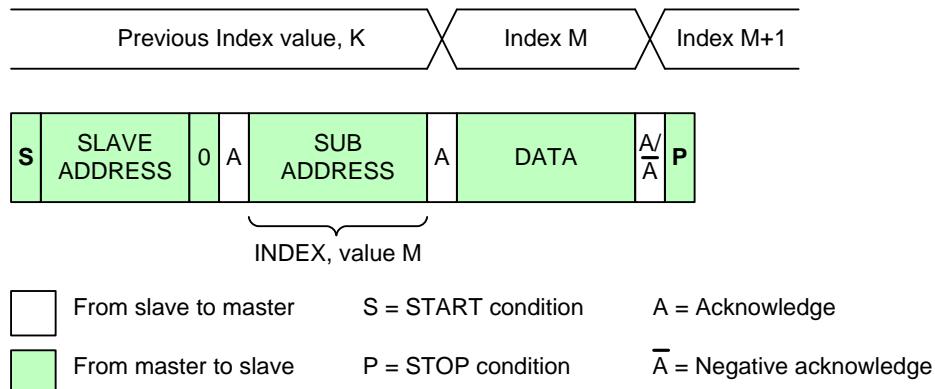


555

556 **Figure 7 CCI Sequential Read Starting from the Current Location**

557 **6.1.2.5 Single Write to a Random Location**

558 A write operation to a random location is illustrated in Figure 8. The master issues a write operation to the  
559 slave then issues the index and data after the slave has acknowledged the write operation. The write operation  
560 is terminated with a stop condition from the master.



561

562 **Figure 8 CCI Single Write to a Random Location**

563 **6.1.2.6 Sequential Write**

564 The sequential write operation is illustrated in Figure 9. The slave auto-increments the index after each data  
565 byte is received. The sequential write operation is terminated with a stop condition from the master.

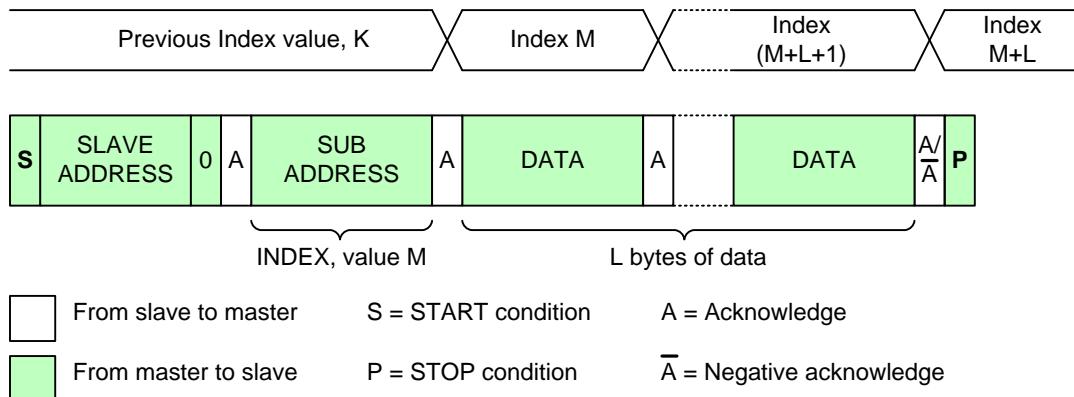


Figure 9 CCI Sequential Write Starting from a Random Location

## 6.2 CCI Slave Addresses

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

## 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.

594 The address of the first byte of a multi-byte register may, or may not be, aligned to the size of the register;  
 595 i.e., a multiple of the number of register bytes. The register alignment is an implementation choice between  
 596 processing optimized and bandwidth optimized organizations. There are no restrictions on the number or mix  
 597 of multi-byte registers within the available 64K by 8-bit index space, with the exception that rules for the  
 598 valid locations for the MS bytes and LS bytes of registers are followed.

599 Partial access to multi-byte registers is not allowed. A multi-byte register shall only be accessed by a single  
 600 sequential message. When a multi-byte register is accessed, its first byte is accessed first; its second byte is  
 601 accessed second, etc.

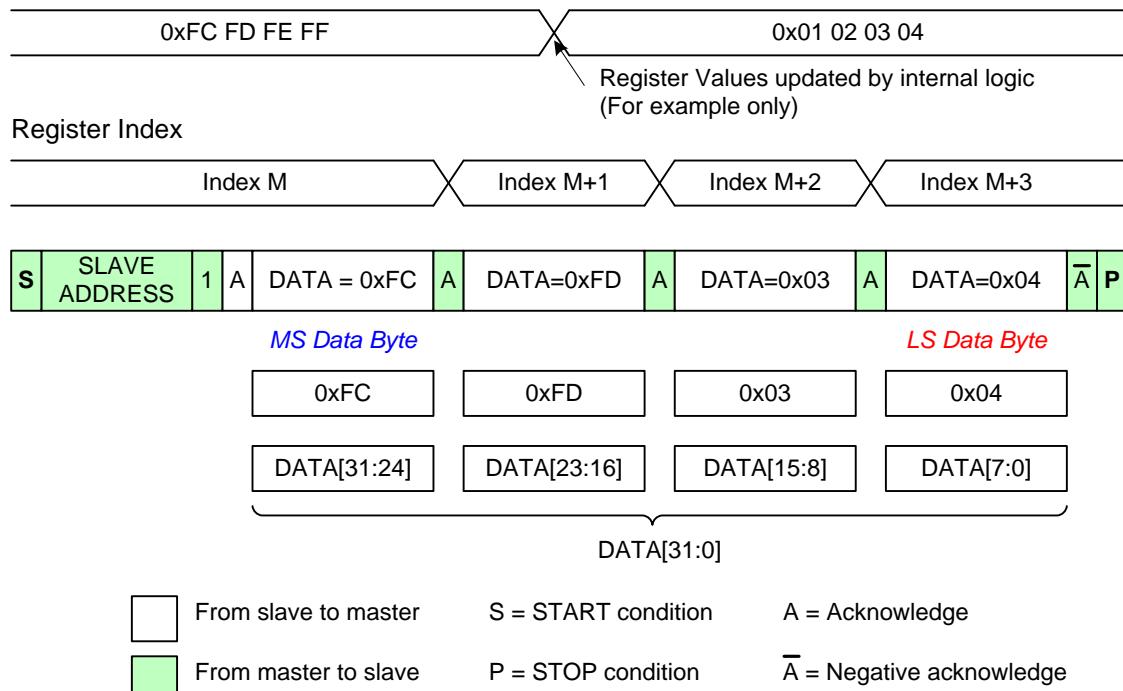
602 When a multi-byte register is accessed, the following re-timing rules must be followed:

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

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

608 Without re-timing, data may be corrupted as illustrated in Figure 10 and Figure 11.

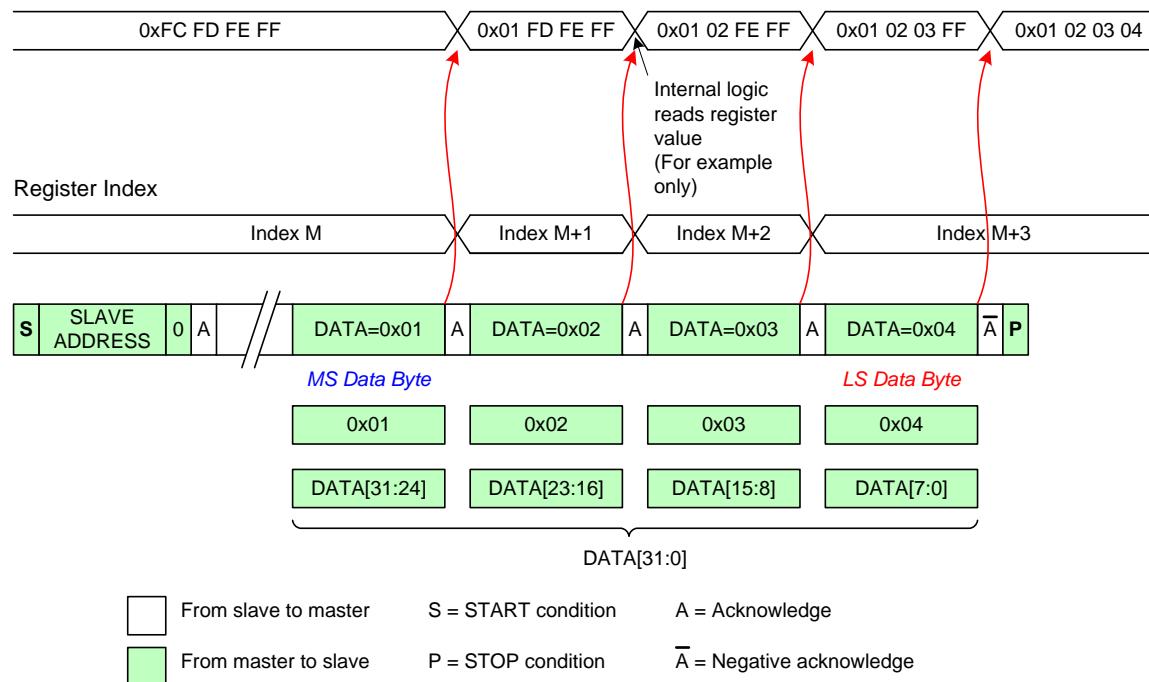
Internal 32-bit Register Value (Locations M, M+1, M+2 and M+3)



609  
610

Figure 10 Corruption of a 32-bit Wide Register during a Read Message

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



611

612

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

613

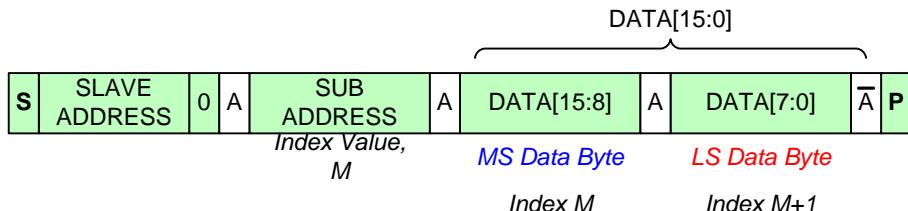
### 6.3.2 The Transmission Byte Order for Multi-byte Register Values

614

This is a normative section.

615  
616

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.

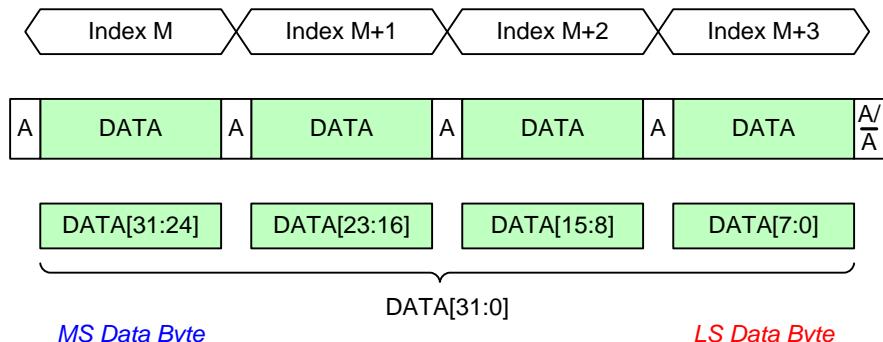


617

618

**Figure 12 Example 16-bit Register Write**

Register Index

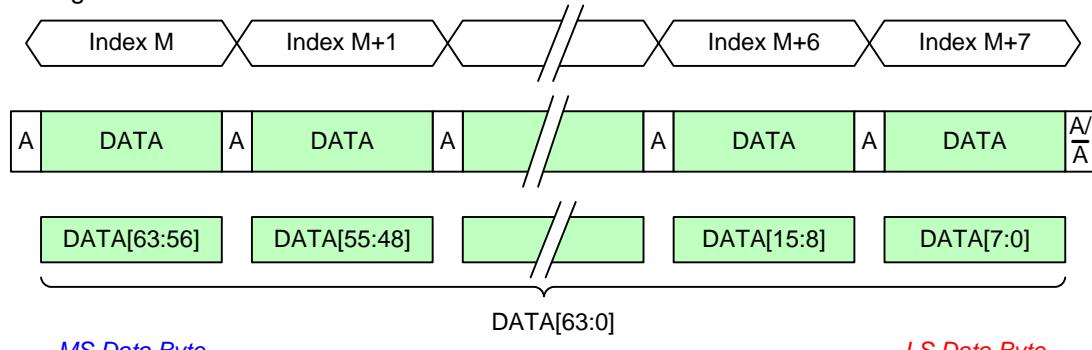


619

620

**Figure 13 Example 32-bit Register Write (address not shown)**

Register Index



621

622

**Figure 14 Example 64-bit Register Write (address not shown)**

### 6.3.3 Multi-Byte Register Protocol

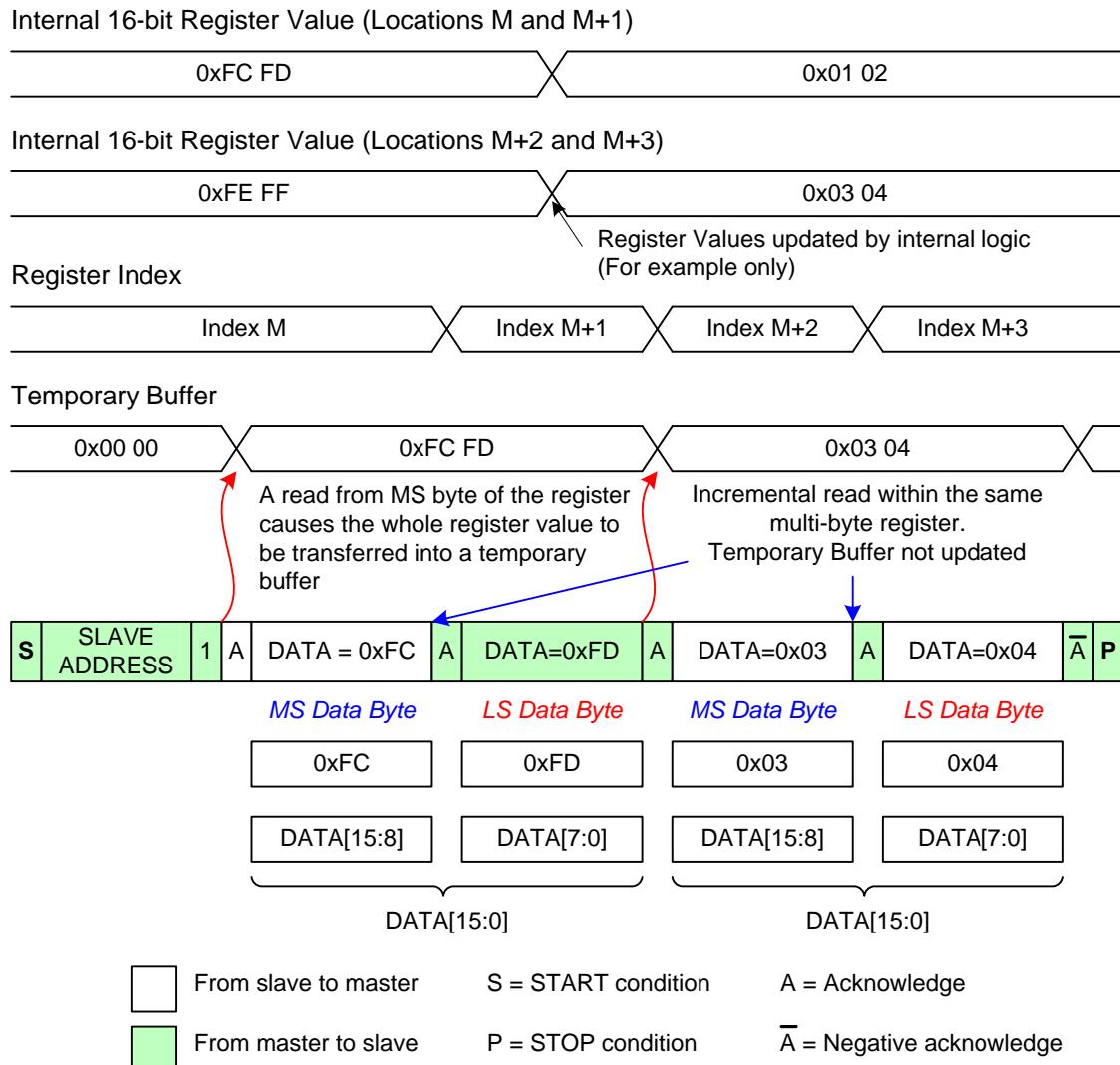
624 This is an informative section.

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

#### 6.3.3.1 Reading Multi-byte Registers

628 To ensure that the value read from a multi-byte register is consistent (i.e. all bytes are temporally coherent),  
629 the device internally transfers the contents of the register into a temporary buffer when the MS byte of the  
630 register is read. The contents of the temporary buffer are then output as a sequence of bytes on the SDA line.  
631 Figure 15 and Figure 16 illustrate multi-byte register read operations.

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



634

635

**Figure 15 Example 16-bit Register Read**

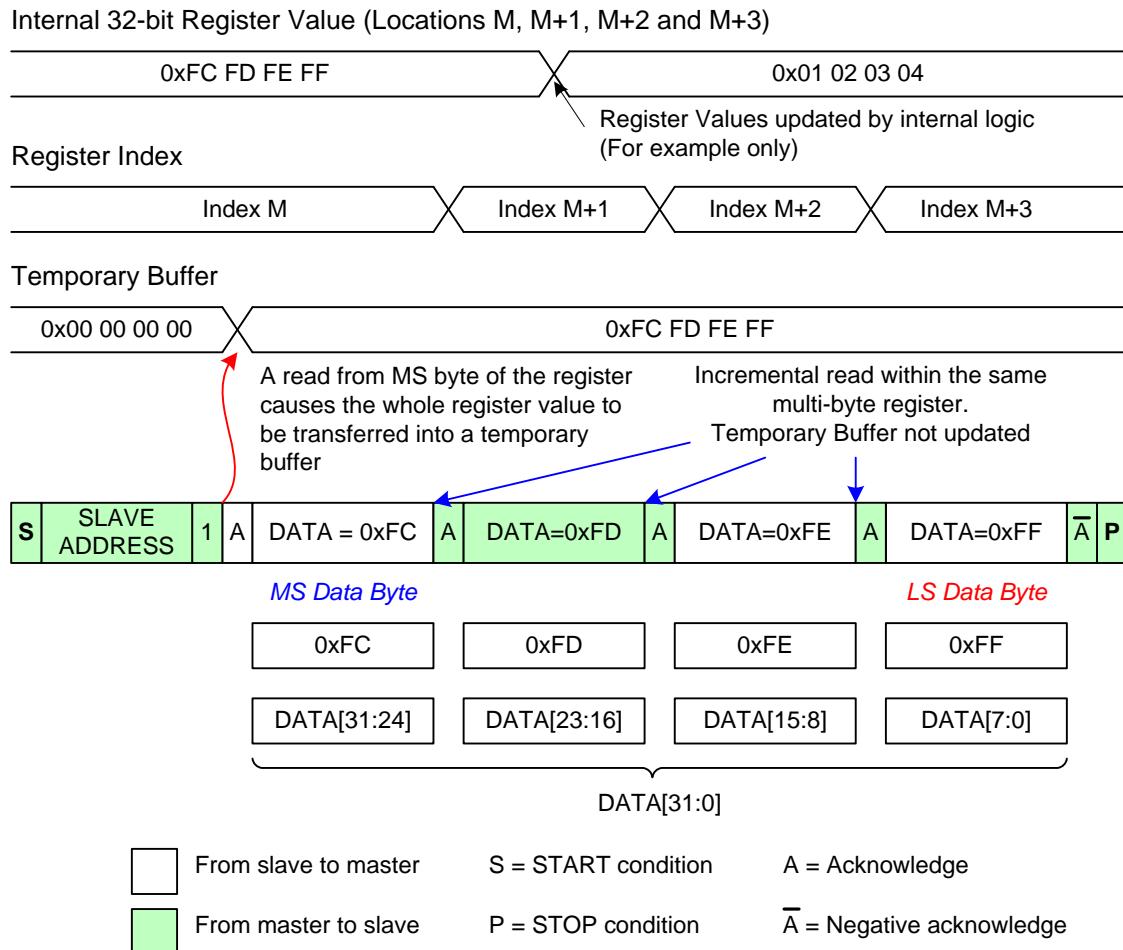
636 In this definition there is no distinction made between whether the register is accessed incrementally via  
637 separate, single byte read messages with no intervening data writes or via a single multi-location read  
638 message. This protocol purely relates to the behavior of the index value.

639 Examples of when the temporary buffer is updated are as follows:

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

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

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



647

648

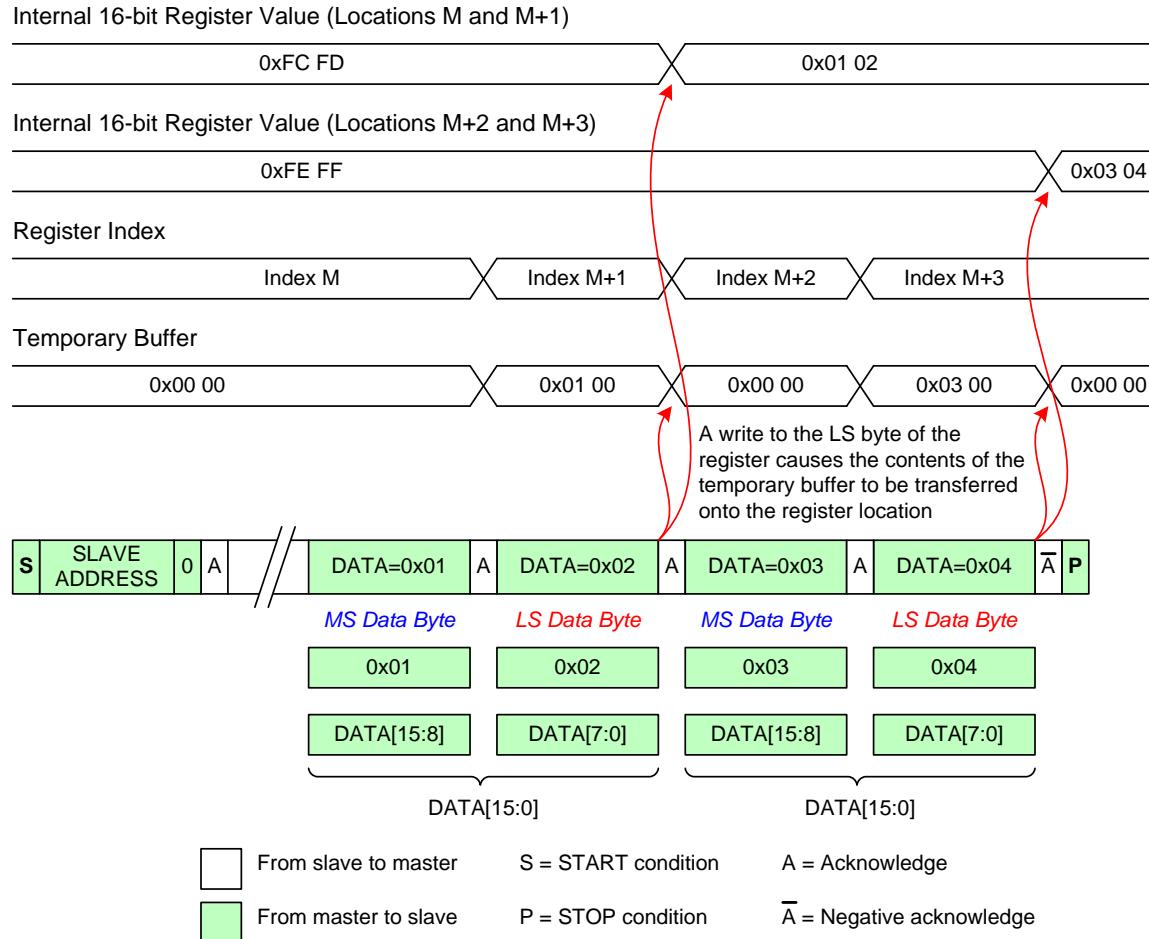
**Figure 16 Example 32-bit Register Read**

649 **6.3.3.2 Writing Multi-byte Registers**

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

653 Figure 17 and Figure 18 illustrate multi-byte register write operations.

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

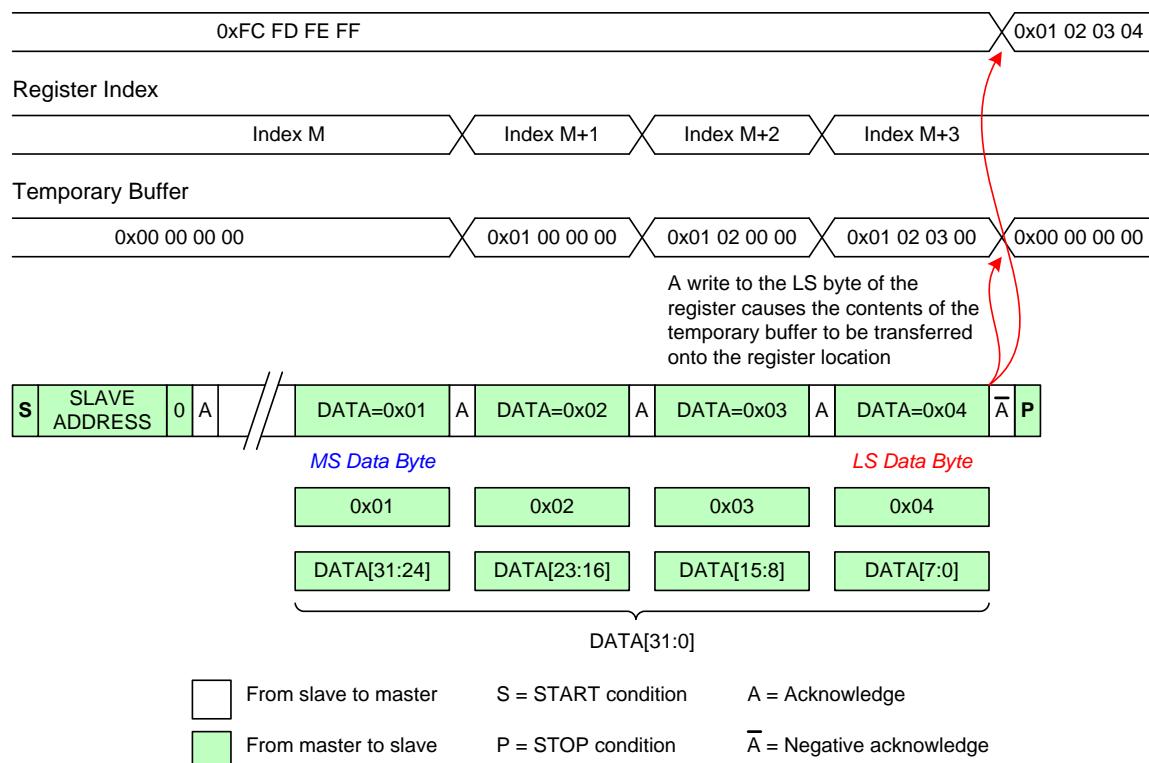


656  
657

658

Figure 17 Example 16-bit Register Write

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



659

660

**Figure 18 Example 32-bit Register Write**

#### 661 6.4 Electrical Specifications and Timing for I/O Stages

662 The electrical specification and timing for I/O stages conform to I<sup>2</sup>C Standard- and Fast-mode devices.  
663 Information presented in Table 1 is from [NXP01].

664

**Table 1 CCI I/O Characteristics**

Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
LOW level input voltage	V <sub>IL</sub>	-0.5	0.3V <sub>DD</sub>	-0.5	0.3 V <sub>DD</sub>	V
HIGH level input voltage	V <sub>IH</sub>	0.7V <sub>DD</sub>	Note 1	0.7V <sub>DD</sub>	Note 1	V
Hysteresis of Schmitt trigger inputs V <sub>DD</sub> > 2V V <sub>DD</sub> < 2V	V <sub>HYS</sub>	N/A N/A	N/A N/A	0.05V <sub>DD</sub> 0.1V <sub>DD</sub>	- -	V
LOW level output voltage (open drain) at 3mA sink current V <sub>DD</sub> > 2V V <sub>DD</sub> < 2V	V <sub>OL1</sub> V <sub>OL3</sub>	0 N/A	0.4 N/A	0 0	0.4 0.2V <sub>DD</sub>	V

Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
HIGH level output voltage	V <sub>OH</sub>	N/A	N/A	0.8V <sub>DD</sub>		V
Output fall time from V <sub>IHmin</sub> to V <sub>ILmax</sub> with bus capacitance from 10 pF to 400 pF	t <sub>OF</sub>	-	250	20+0.1C <sub>B</sub> Note 2	250	ns
Pulse width of spikes which shall be suppressed by the input filter	t <sub>SP</sub>	N/A	N/A	0	50	ns
Input current each I/O pin with an input voltage between 0.1 V <sub>DD</sub> and 0.9 V <sub>DD</sub>	I <sub>I</sub>	-10	10	-10 Note 3	10 Note 3	μA
Input/Output capacitance (SDA)	C <sub>I/O</sub>	-	8	-	8	pF
Input capacitance (SCL)	C <sub>I</sub>	-	6	-	6	pF

665 Notes:

- 666 1. Maximum VIH = V<sub>DDmax</sub> + 0.5V  
 667 2. C<sub>B</sub> = capacitance of one bus line in pF  
 668 3. I/O pins of Fast-mode devices shall not obstruct the SDA and SCL line if V<sub>DD</sub> is switched off

669 **Table 2 CCI Timing Specification**

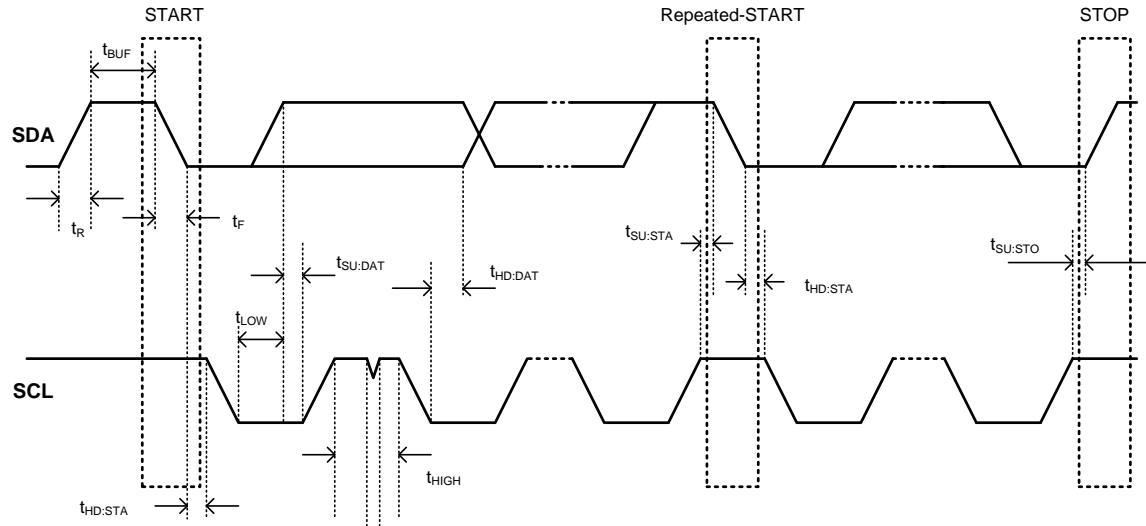
Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
SCL clock frequency	f <sub>SCL</sub>	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	t <sub>HD:STA</sub>	0.4	-	0.6	-	μs
LOW period of the SCL clock	t <sub>LOW</sub>	4.7	-	1.3	-	μs
HIGH period of the SCL clock	t <sub>HIGH</sub>	4.0	-	0.6	-	μs
Setup time for a repeated START condition	t <sub>SU:STA</sub>	4.7	-	0.6	-	μs
Data hold time	t <sub>HD:DAT</sub>	0 Note 2	3.45 Note 3	0 Note 2	0.9 Note 3	μs
Data set-up time	t <sub>SU:DAT</sub>	250	-	100 Note 4	-	ns
Rise time of both SDA and SCL signals	t <sub>R</sub>	-	1000	20+0.1C <sub>B</sub> Note 5	300	ns
Fall time of both SDA and SCL signals	t <sub>F</sub>	-	300	20+0.1C <sub>B</sub> Note 5	300	ns
Set-up time for STOP condition	t <sub>SU:STO</sub>	4.0	-	0.6	-	μs
Bus free time between a STOP and START condition	t <sub>BUF</sub>	4.7	-	1.3	-	μs
Capacitive load for each bus line	C <sub>B</sub>	-	400	-	400	pF

Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
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

670 Notes:

- 671 1. All values referred to  $V_{IHmin} = 0.7V_{DD}$  and  $V_{ILmax} = 0.3V_{DD}$
- 672 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
- 673 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
- 674 4. A Fast-mode I2C-bus device can be used in a Standard-mode I2C-bus system, but the requirement
- 675  $tsu:DAT \geq 250$  ns shall be then met. This will be automatically the case if the device does not stretch
- 676 the LOW period of the SCL signal. If such device does stretch the low period of SCL signal, it shall
- 677 output the next data bit to the SDA line  $t_{MAX} + tsu:DAT = 1000 + 250 = 1250$  ns (according to the
- 678 Standard-mode I2C bus specification) before the SCL line is released.
- 679
- 680 5.  $CB = \text{total capacitance of one bus line in pF}$ .

681 The CCI timing is illustrated in Figure 19.



683  
684

685 **Figure 19 CCI Timing**

686 **7 Physical Layer**

687 CSI-2 uses the physical layer described in [MIPI01].

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

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

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

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

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

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

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

701 All CSI-2 implementations shall support forward escape ULPS on all Data Lanes.

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

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

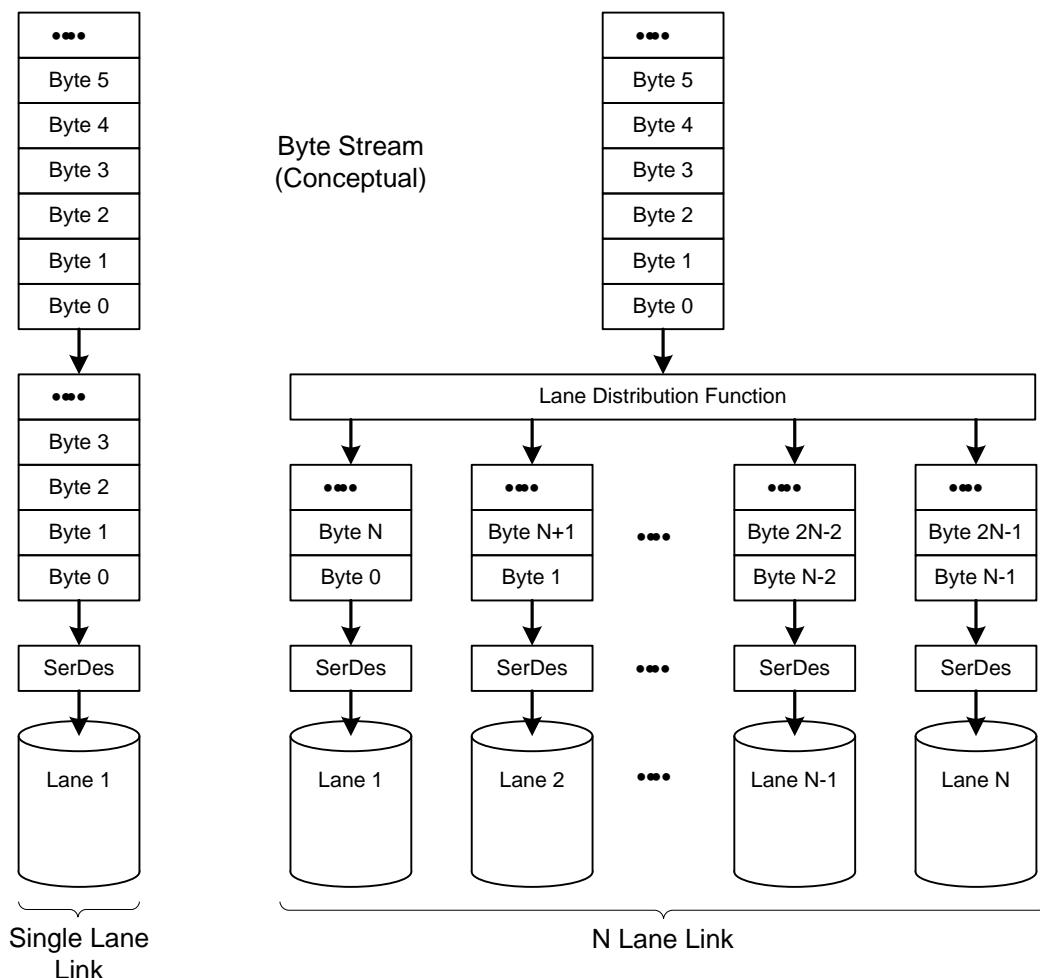
- 706     • For initial calibration sequence the Transmitter shall be programmed with the desired frequency for  
707       calibration. It will then transmit the deskew calibration pattern and the Receiver will autonomously  
708       detect this pattern and tune the deskew function to achieve optimum performance.  
  
709     • For any transmitter frequency changes the deskew calibration shall be rerun.  
  
710     • Some transmitters and/or receiver may require deskew calibration to be rerun periodically and it is  
711       suggested that it can be optimally done within vertical or frame blanking periods.

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

## 715 8 Multi-Lane Distribution and Merging

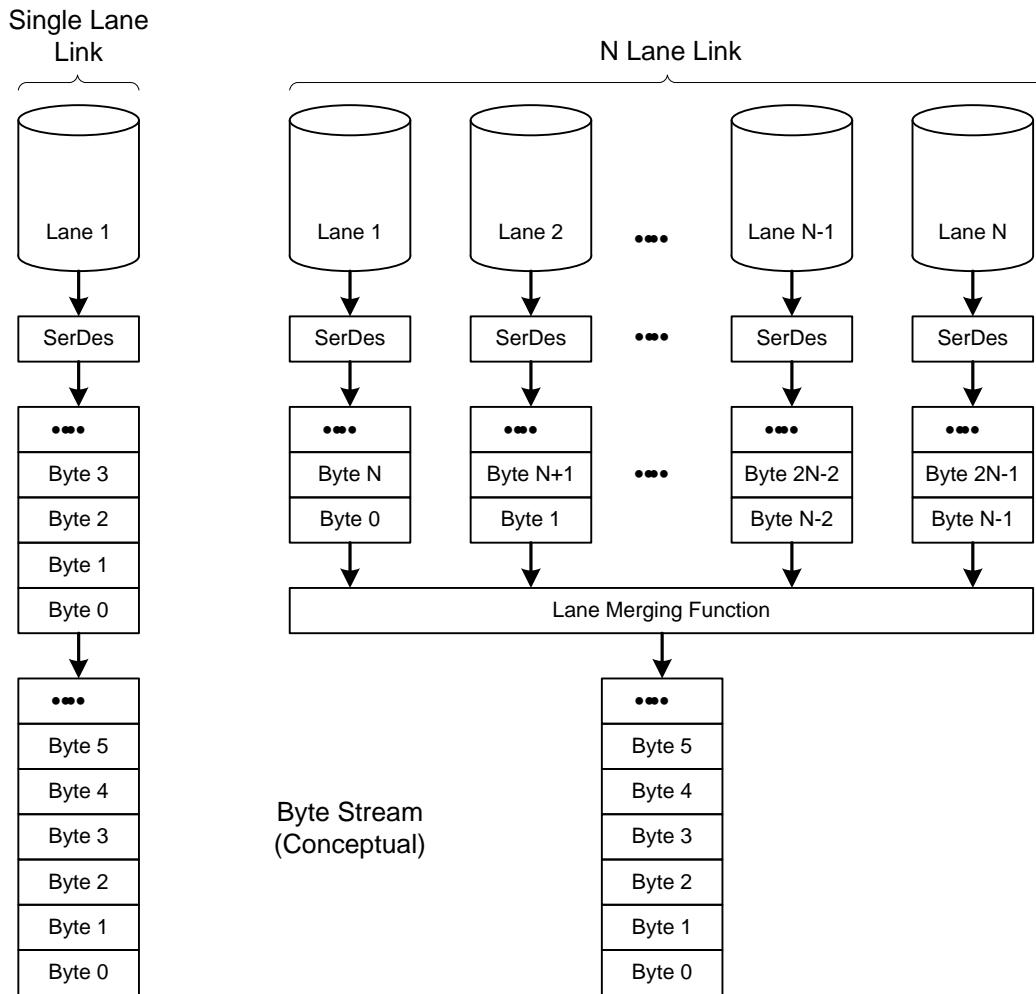
716 CSI-2 is a Lane-scalable specification. Applications requiring more bandwidth than that provided by one data  
717 Lane, or those trying to avoid high clock rates, can expand the data path to a higher number of Lanes and  
718 obtain approximately linear increases in peak bus bandwidth. The mapping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

721 Conceptually, between the PHY and higher functional layers is a layer that handles multi-Lane  
722 configurations. In the transmitter, the layer distributes a sequence of packet bytes across N Lanes, where each  
723 Lane is an independent unit of physical-layer logic (serializers, etc.) and transmission circuitry. In the  
724 receiver, it collects incoming bytes from N Lanes and consolidates (merges) them into complete packets to  
725 pass into the packet decomposer.



726  
727

Figure 20 Conceptual Overview of the Lane Distributor Function



728

729

**Figure 21 Conceptual Overview of the Lane Merging Function**

730 The Lane distributor takes a transmission of arbitrary byte length, buffers up  $N$  bytes (where  $N = \text{number of}$   
731 Lanes), and then sends groups of  $N$  bytes in parallel across  $N$  Lanes. Before sending data, all Lanes perform  
732 the SoT sequence in parallel to indicate to their corresponding receiving units that the first byte of a packet  
733 is beginning. After SoT, the Lanes send groups of successive bytes from the first packet in parallel, following  
734 a round-robin process.

735 Examples:

- 736 • 2-Lane system (Figure 22): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to  
737 Lane 1, byte 3 goes to Lane 2, byte 4 goes to Lane 1, and so on.
- 738 • 3-Lane system (Figure 23): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to  
739 Lane 3, byte 3 goes to Lane 1, byte 4 goes to Lane 2, and so on.
- 740 •  $N$ -Lane system (Figure 24): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte  $N-1$   
741 goes to Lane  $N$ , byte  $N$  goes to Lane 1, byte  $N+1$  goes to Lane 2, and so on.
- 742 •  $N$ -lane system (Figure 25) with  $N > 4$  short packet (4 bytes) transmission: byte 0 of the packet goes  
743 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  $N$   
744 do not receive bytes and stay in LPS state.

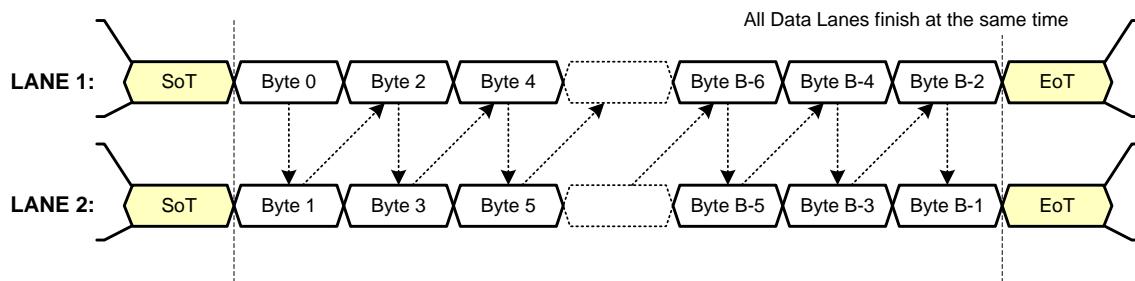
745 At the end of the transmission, there may be “extra” bytes since the total byte count may not be an integer  
 746 multiple of the number of Lanes, N. One or more Lanes may send their last bytes before the others. The Lane  
 747 distributor, as it buffers up the final set of less-than-N bytes in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

751 Each D-PHY data Lane operates autonomously.

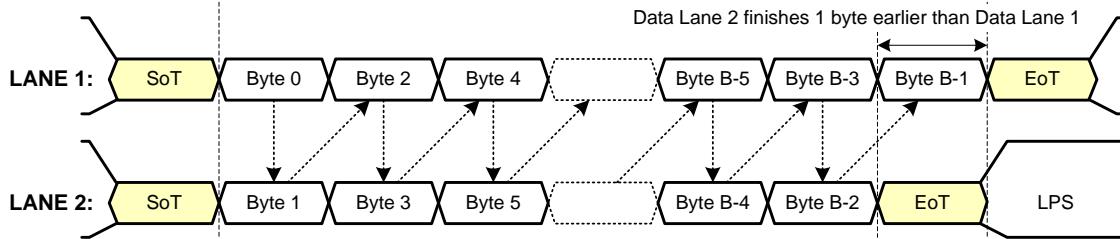
752 Although multiple Lanes all start simultaneously with parallel “start packet” codes, they may complete the  
 753 transaction at different times, sending “end packet” codes one cycle (byte) apart.

754 The N PHYs on the receiving end of the link collect bytes in parallel, and feed them into the Lane-merging  
 755 layer. This reconstitutes the original sequence of bytes in the transmission, which can then be partitioned into  
 756 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

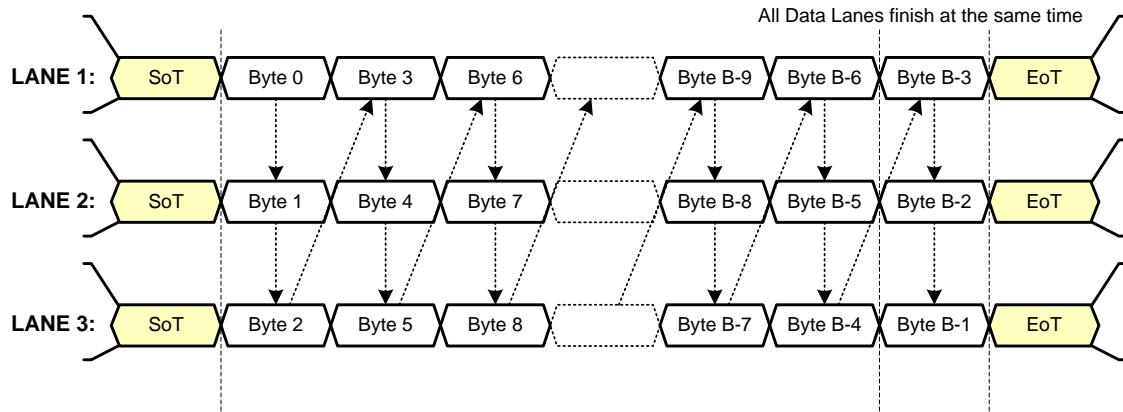
EoT – End of Transmission

757

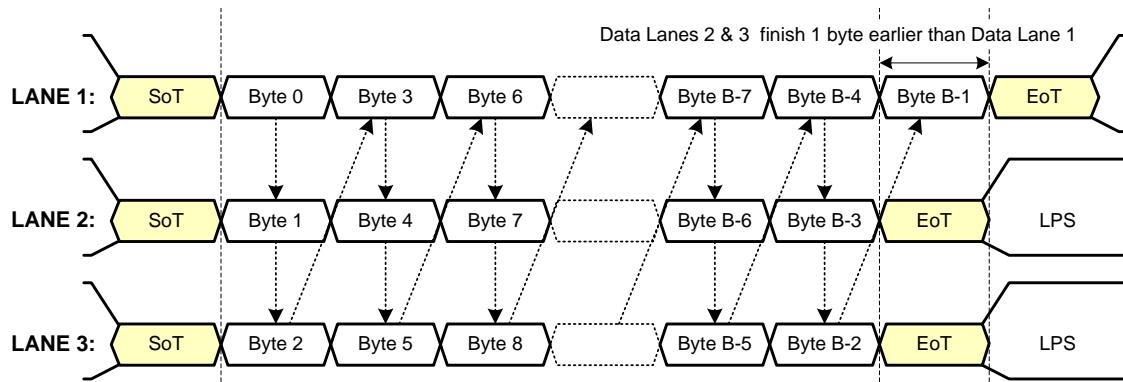
758

**Figure 22 Two Lane Multi-Lane Example**

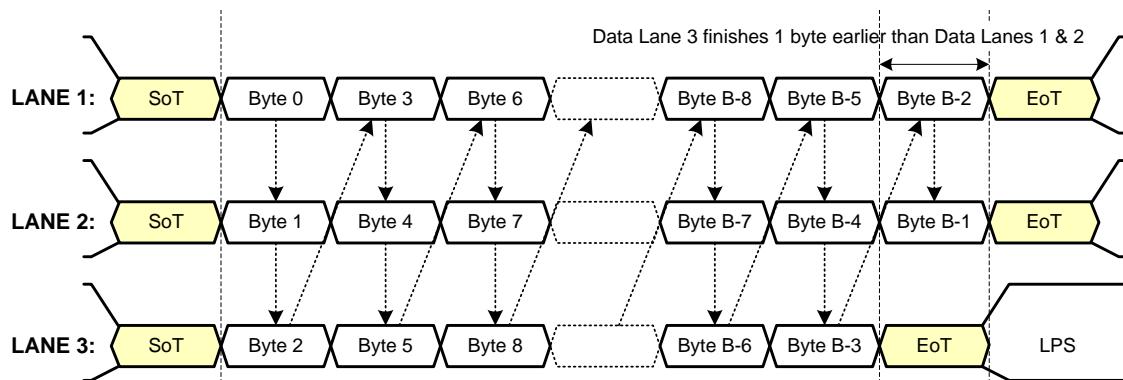
**Number of Bytes, B, transmitted is an integer multiple of the number of lanes:**



**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:**

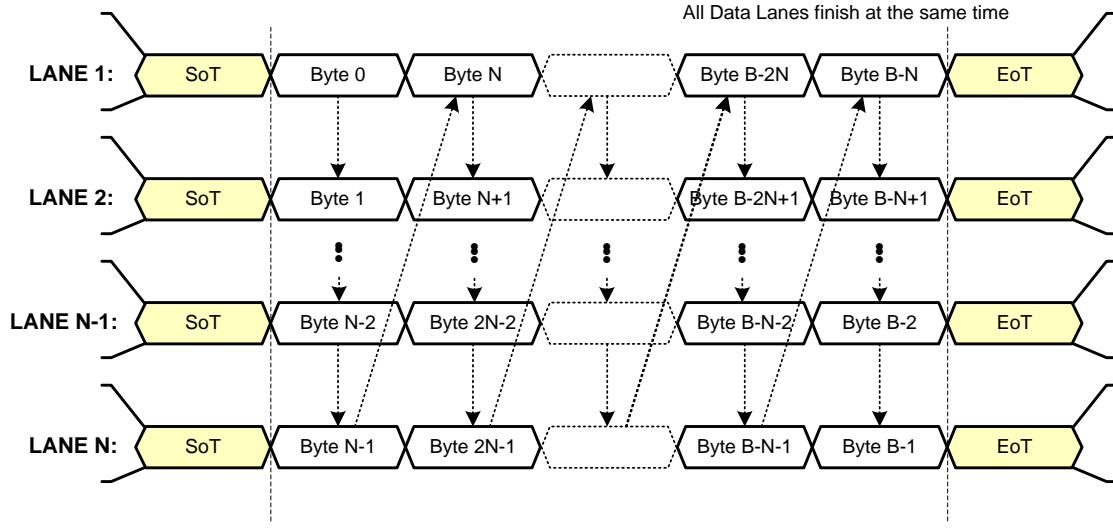
LPS – Low Power State

SoT – Start of Transmission

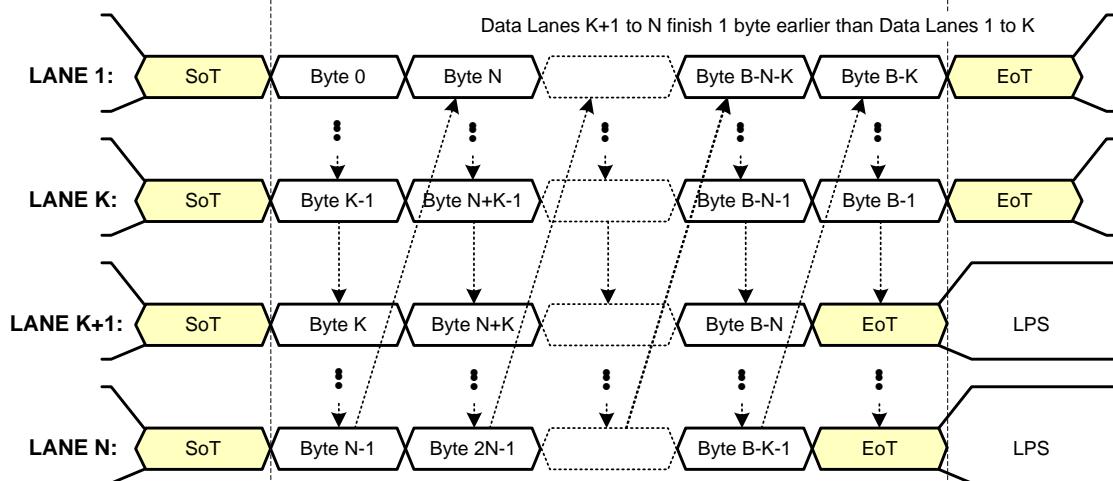
EoT – End of Transmission

**Figure 23 Three Lane Multi-Lane Example**

**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

761  
762

763

**Figure 24 N Lane Multi-Lane Example**

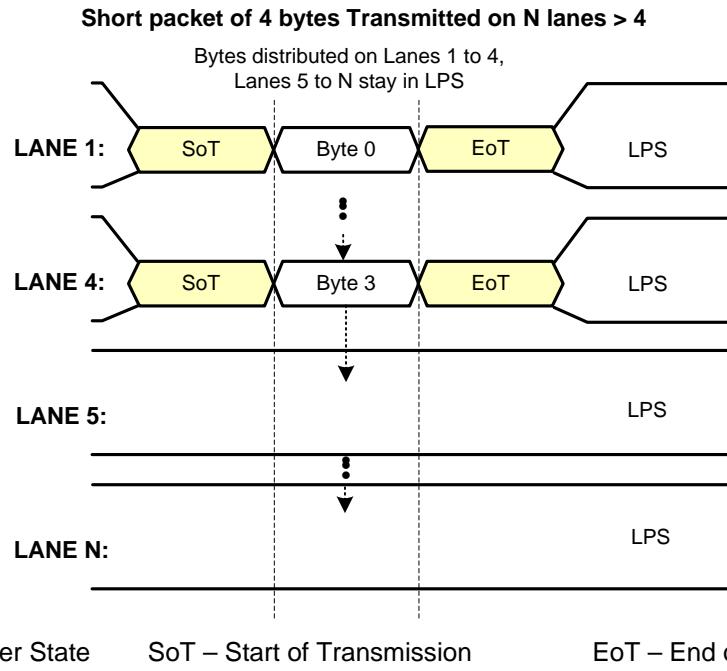


Figure 25 N Lane Multi-Lane Example for Short Packet Transmission

## 8.1 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.

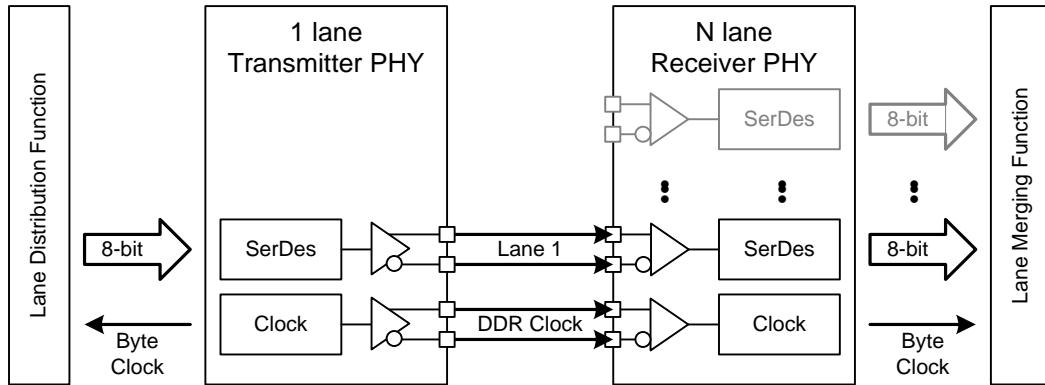
An "N" data Lane receiver shall be connected with an "M" data Lane transmitter, by CCI configuration of the Lane distribution and merging layers within the CSI-2 transmitter and receiver when more than one data Lane is used. Thus, if  $M \leq N$  a receiver with N data Lanes shall work with transmitters 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 26 and Figure 27).
- 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 28 and Figure 29).

779

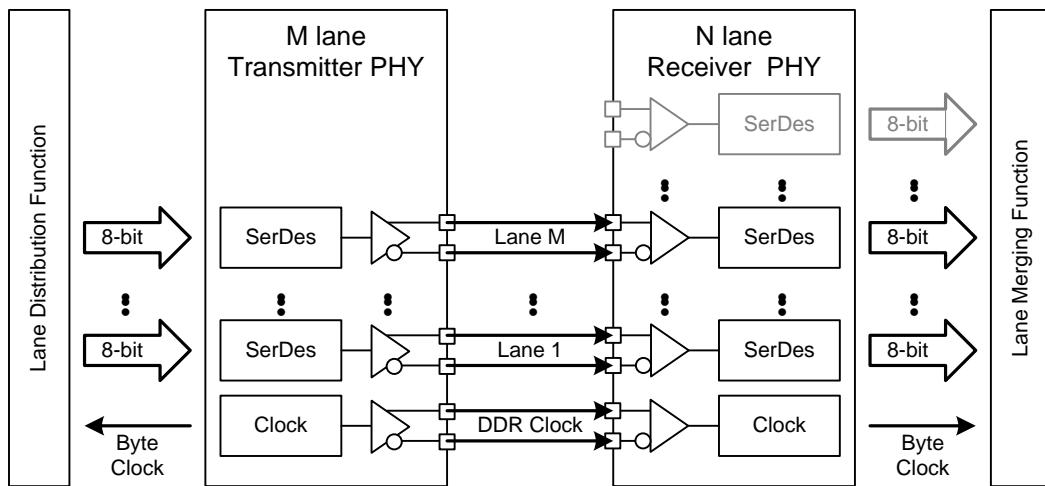
780



**Figure 26 One Lane Transmitter and N Lane Receiver Example**

781

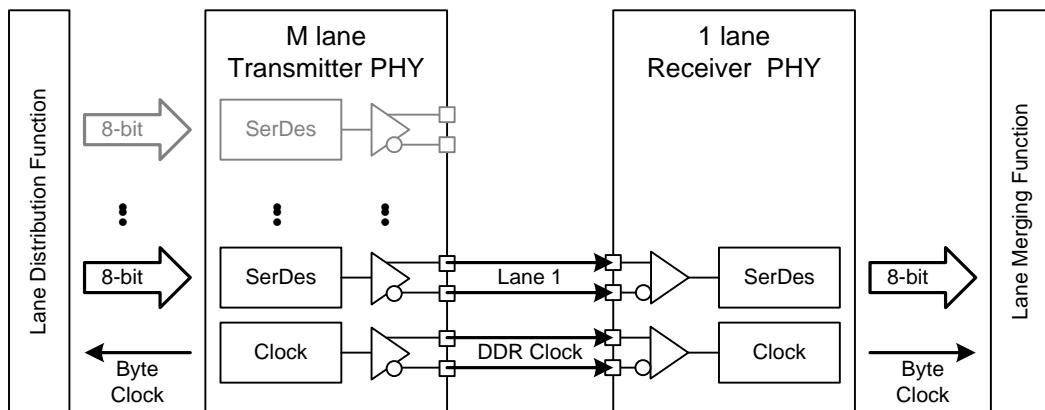
782



**Figure 27 M Lane Transmitter and N Lane Receiver Example (M < N)**

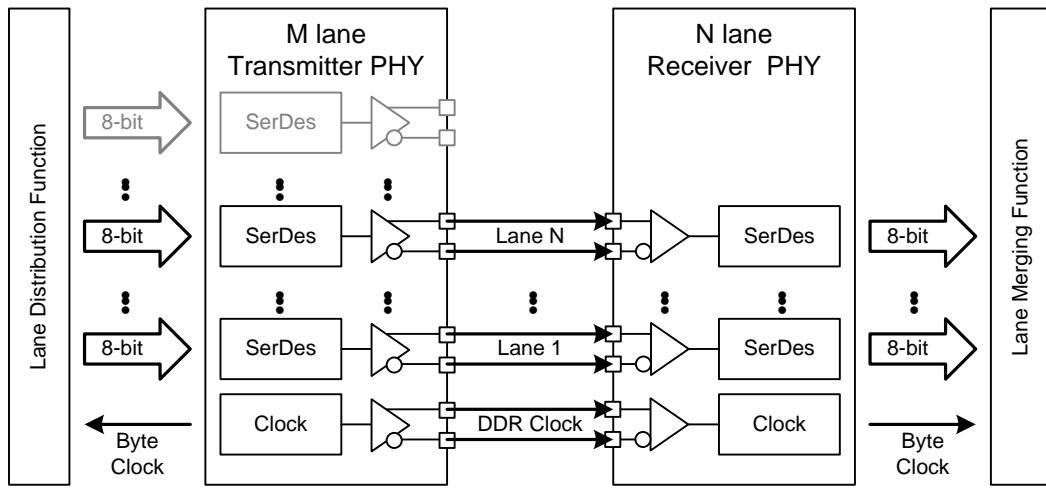
783

784



**Figure 28 M Lane Transmitter and One Lane Receiver Example**

785  
786



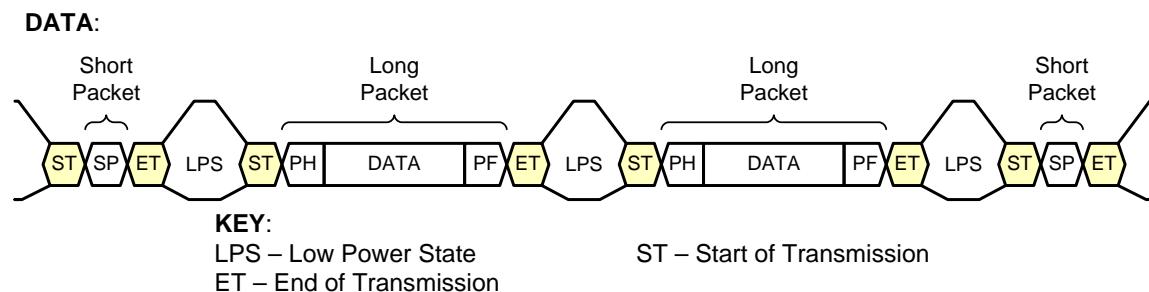
**Figure 29 M Lane Transmitter and N Lane Receiver Example (N<M)**

## 787 9 Low Level Protocol

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

791 Low Level Protocol Features:

- 792 • Transport of arbitrary data (Payload independent)
- 793 • 8-bit word size
- 794 • Support for up to four interleaved virtual channels on the same link
- 795 • Special packets for frame start, frame end, line start and line end information
- 796 • Descriptor for the type, pixel depth and format of the Application Specific Payload data
- 797 • 16-bit Checksum Code for error detection.



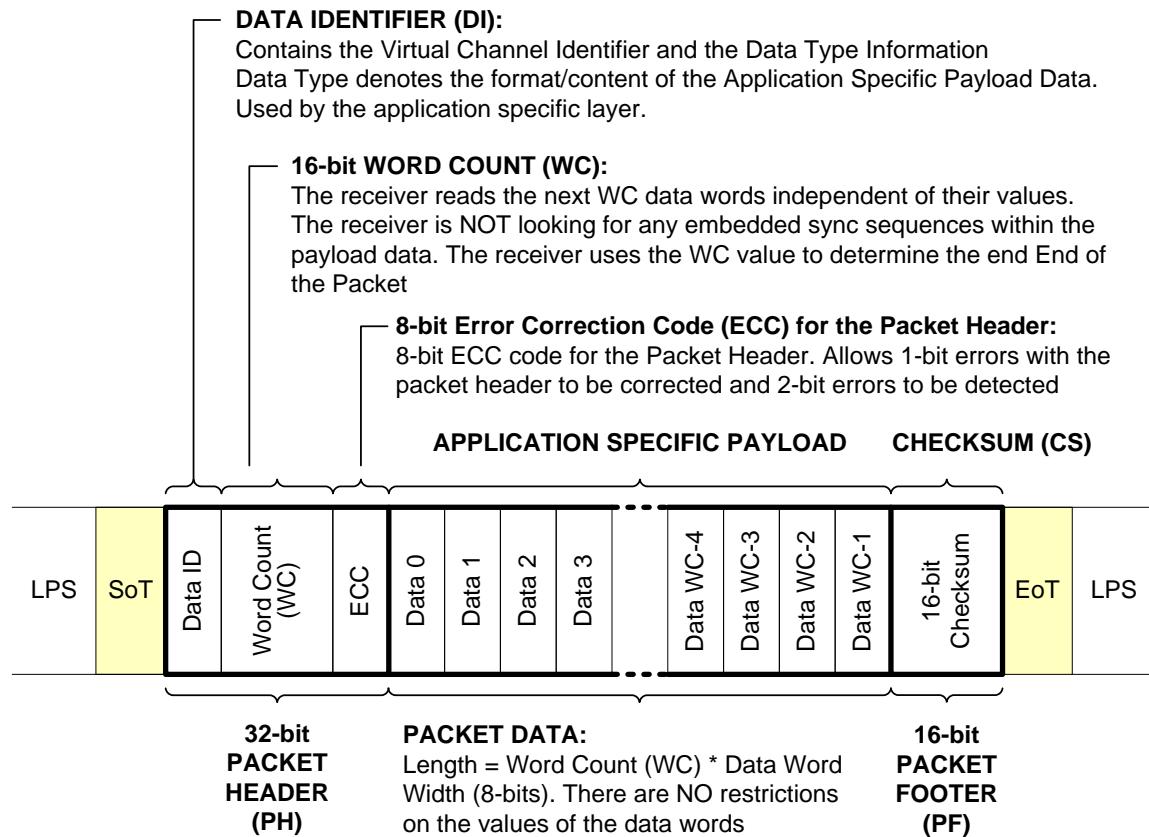
798  
799 **Figure 30 Low Level Protocol Packet Overview**

### 800 9.1 Low Level Protocol Packet Format

801 Two packet structures are defined for low-level protocol communication: Long packets and Short packets.  
802 For each packet structure exit from the low power state followed by the Start of Transmission (SoT) sequence  
803 indicates the start of the packet. The End of Transmission (EoT) sequence followed by the low power state  
804 indicates the end of the packet.

#### 805 9.1.1 Low Level Protocol Long Packet Format

806 Figure 31 shows the structure of the Low Level Protocol Long Packet. A Long Packet shall be identified by  
807 Data Types 0x10 to 0x37. See Table 3 for a description of the Data Types. A Long Packet shall consist of  
808 three elements: a 32-bit Packet Header (PH), an application specific Data Payload with a variable number of  
809 8-bit data words and a 16-bit Packet Footer (PF). The Packet Header is further composed of three elements:  
810 an 8-bit Data Identifier, a 16-bit Word Count field and an 8-bit ECC. The Packet footer has one element, a  
811 16-bit checksum. See Section 9.2 through Section 9.5 for further descriptions of the packet elements.



**Figure 31 Long Packet Structure**

- 812 The Data Identifier defines the Virtual Channel for the data and the Data Type for the application specific payload data.
- 813 The Word Count 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. Neither the Packet Header nor the Packet Footer shall be included in the Word Count.
- 814 The Error Correction Code (ECC) byte allows single-bit errors to be corrected and 2-bit errors to be detected in the packet header. This includes both the data identifier value and the word count value.
- 815 After the end of the Packet Header, the receiver reads the next Word Count \* 8-bit data words of the Data Payload. While reading the Data Payload the receiver shall not look for any embedded sync codes. Therefore, there are no limitations on the value of a data word.
- 816 Once the receiver has read the Data Payload it reads the checksum in the Packet Footer. In the generic case, the length of the Data Payload shall be a multiple of 8-bit data words. In addition, each data format may impose additional restrictions on the length of the payload data, e.g. multiple of four bytes.
- 817 Each byte shall be transmitted least significant bit first. Payload data may be transmitted in any byte order restricted only by data format requirements. Multi-byte elements such as Word Count, Checksum and the Short packet 16-bit Data Field shall be transmitted least significant byte first.
- 818 After the EoT sequence the receiver begins looking for the next SoT sequence.

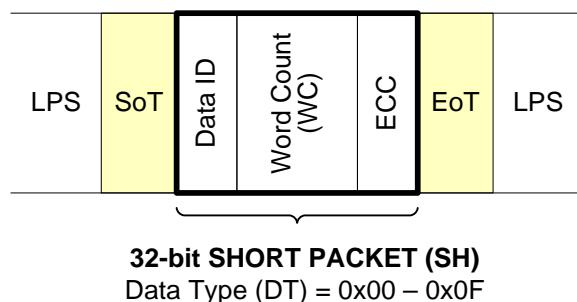
### 831    9.1.2    Low Level Protocol Short Packet Format

832    Figure 32 shows the structure of the Low Level Protocol Short Packet. A Short Packet shall be identified by  
833    Data Types 0x00 to 0x0F. See Table 3 for a description of the Data Types. A Short Packet shall contain only  
834    a Packet Header; a Packet Footer shall not be present. The Word Count field in the Packet Header shall be  
835    replaced by a Short Packet Data Field.

836    For Frame Synchronization Data Types the Short Packet Data Field shall be the frame number. For Line  
837    Synchronization Data Types the Short Packet Data Field shall be the line number. See Table 6 for a  
838    description of the Frame and Line synchronization Data Types.

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

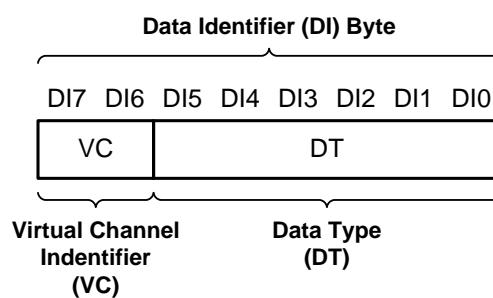
840    The Error Correction Code (ECC) byte allows single-bit errors to be corrected and 2-bit errors to be detected  
841    in the Short Packet.



842                                  **Figure 32 Short Packet Structure**

### 844    9.2    Data Identifier (DI)

845    The Data Identifier byte contains the Virtual Channel Identifier (VC) value and the Data Type (DT) value as  
846    illustrated in Figure 33. The Virtual Channel Identifier is contained in the two MS bits of the Data Identifier  
847    Byte. The Data Type value is contained in the six LS bits of the Data Identifier Byte.



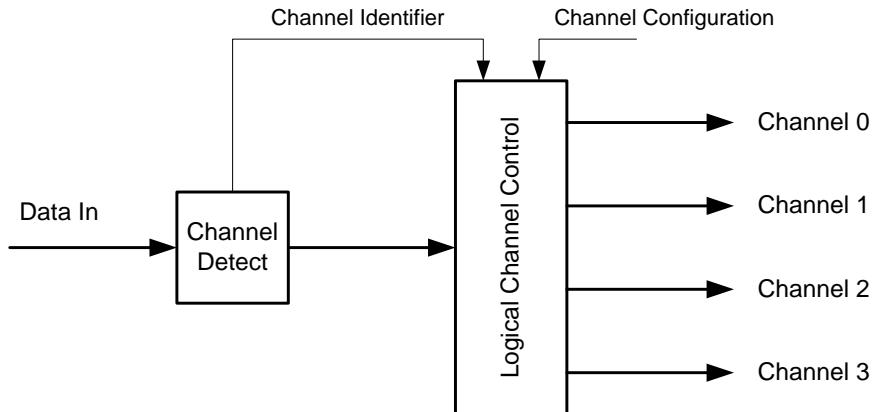
848                                  **Figure 33 Data Identifier Byte**

### 850    9.3    Virtual Channel Identifier

851    The purpose of the Virtual Channel Identifier is to provide separate channels for different data flows that are  
852    interleaved in the data stream.

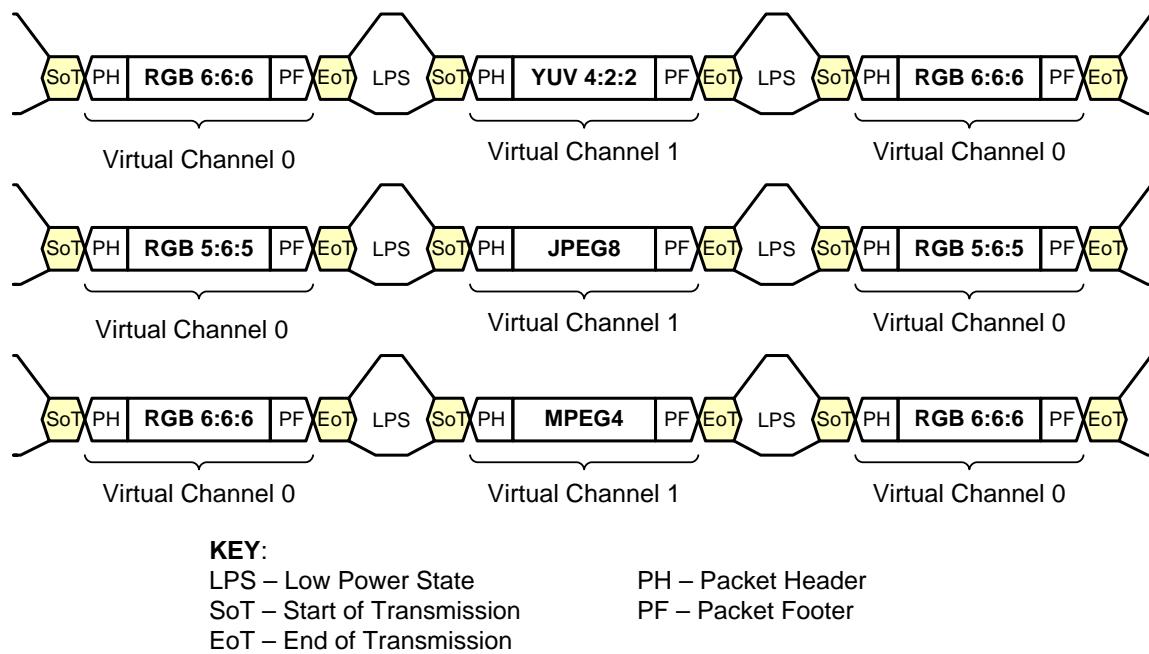
853    The Virtual channel identifier number is in the top two bits of the Data Identifier Byte. The Receiver will  
854    monitor the virtual channel identifier and de-multiplex the interleaved video streams to their appropriate

855 channel. A maximum of four data streams is supported; valid channel identifiers are 0 to 3. The virtual  
856 channel identifiers in the peripherals should be programmable to allow the host processor to control how the  
857 data streams are de-multiplexed. The principle of logical channels is presented in the Figure 34.



858  
859 **Figure 34 Logical Channel Block Diagram (Receiver)**

860 Figure 35 illustrates an example of data streams utilizing virtual channel support.



861  
862 **Figure 35 Interleaved Video Data Streams Examples**

#### 863 **9.4 Data Type (DT)**

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

866 There are eight different data type classes as shown in Table 3. Within each class there are up to eight different  
867 data type definitions. The first two classes denote short packet data types. The remaining six classes denote  
868 long packet data types.

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

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

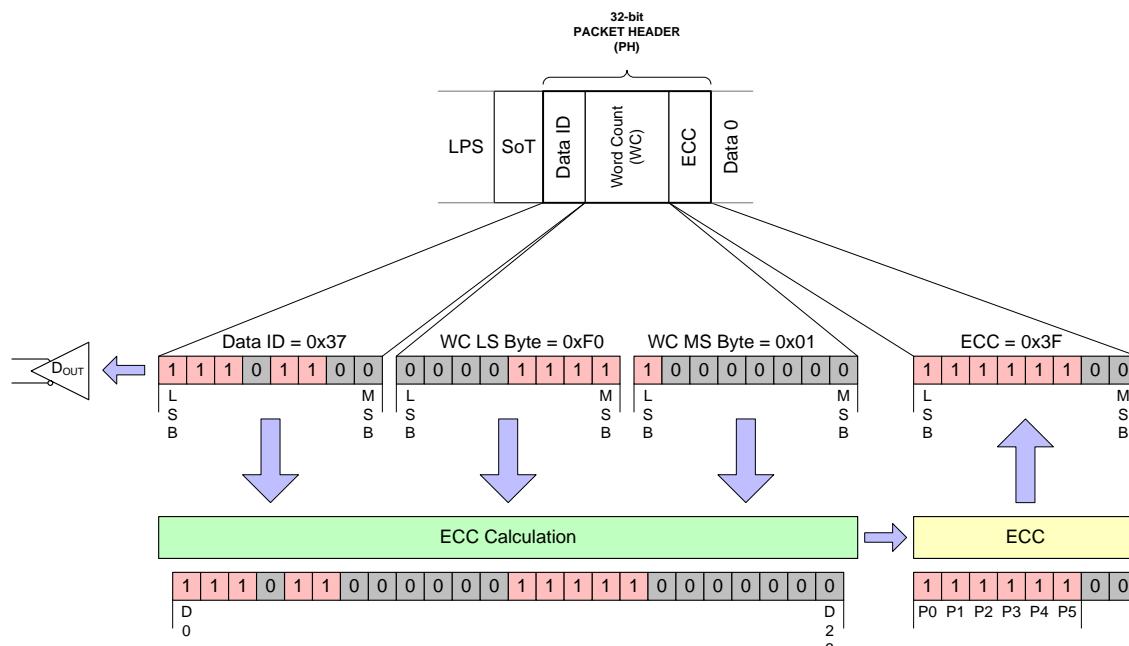
871 **Table 3 Data Type Classes**

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

872 **9.5 Packet Header Error Correction Code**

873 The correct interpretation of the data identifier and word count values is vital to the packet structure. The  
874 Packet Header Error Correction Code byte allows single-bit errors in the data identifier and the word count  
875 to be corrected and two-bit errors to be detected. The 24-bit subset of the code described in Section 9.5.2  
876 shall be used. Therefore, bits 7 and 6 of the ECC byte shall be zero. The error state based on ECC decoding  
877 shall be available at the Application layer in the receiver.

878 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])  
879 to D[15:8] and the Word Count MS Byte (WC[15:8]) to D[23:16]. This mapping is shown in Figure 36,  
880 which also serves as an ECC calculation example.



881

882

**Figure 36 24-bit ECC Generation Example**

883      **9.5.1     General Hamming Code Applied to Packet Header**

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

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

887      The result of appending the computed parity bits to the data bits is called the Hamming code word. The size  
888      of the code word  $c$  is obviously  $d + p$ , and a Hamming code word is described by the ordered set  $(c, d)$ . A  
889      Hamming code word is generated by multiplying the data bits by a generator matrix  $\mathbf{G}$ . The resulting product  
890      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.  
891      The generator matrix  $\mathbf{G}$  used in constructing Hamming codes consists of  $\mathbf{I}$  (the identity matrix) and a parity  
892      generation matrix  $\mathbf{A}$ :

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

894      The packet header plus the ECC code can be obtained as:  $\text{PH} = p * \mathbf{G}$  where  $p$  represents the header (24 or 64  
895      bits) and  $\mathbf{G}$  is the corresponding generator matrix.

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

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

899      If all elements of  $s$  are zero, the code word was received correctly. If  $s$  contains non-zero elements, then at  
900      least one error is present. If a single bit error is encountered then the syndrome  $s$  is one of the elements of  $\mathbf{H}$   
901      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  
902      syndrome will be one of the elements on  $\mathbf{I}$ , else it will be the data bit identified by the position of the syndrome  
903      in  $\mathbf{A}^T$ .

904      **9.5.2     Hamming-Modified Code**

905      The error correcting code used is a 7+1 bits Hamming-modified code (72,64) and the subset of it is 5+1bits  
906      or (30,24). Hamming codes use parity to correct one error or detect two errors, but they are not capable of  
907      doing both simultaneously, thus one extra parity bit needs to be added. The code used allows the same 6-bit  
908      syndromes to correct the first 24-bits of a 64-bit sequence. To specify a compact encoding of parity and  
909      decoding of syndromes, the following matrix is used:

910

911      **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	0x43	0x45	0x46	0x49	0x4A	0x4C	0x51	0x52	
0b100	0x54	0x58	0x61	0x62	0x64	0x68	0x70	0x83	
0b101	0x85	0x86	0x89	0x8A	0x3D	0x3E	0x4F	0x57	
0b110	0x8C	0x91	0x92	0x94	0x98	0xA1	0xA2	0xA4	

		d2d1d0							
d5d4d3		0b000	0b001	0b010	0b011	0b100	0b101	0b110	0b111
0b111		0xA8	0xB0	0xC1	0xC2	0xC4	0xC8	0xD0	0xE0

912     Each cell in the matrix represents a syndrome and the first twenty-four cells (the orange rows) are using the  
913     first three or five bits to build the syndrome. Each syndrome in the matrix is MSB left aligned:

914               e.g. 0x07=0b0000\_0111=P7P6P5P4P3P2P1P0

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

917               e.g. 37th bit position is encoded 0b100\_101 and has the syndrome 0x68.

918     To derive the parity P0 for 24-bits, the P0's in the orange rows will define if the corresponding bit position  
919     is used in P0 parity or not.

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

921     Similar, to derive the parity P0 for 64-bits, all P0's in Table 5 will define the corresponding bit positions to  
922     be used.

923     To correct a single-bit error, the syndrome has to be one of the syndromes Table 4, which will identify the  
924     bit position in error. The syndrome is calculated as:

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

927     Table 5 represents the same information as the matrix in Table 4, organized such that will give a better insight  
928     on the way parity bits are formed out of data bits. The orange area of the table has to be used to form the  
929     ECC to protect a 24-bit header, whereas the whole table has to be used to protect a 64-bit header.

930               **Table 5 ECC Parity Generation Rules**

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
<b>0</b>	0	0	0	0	0	1	1	1	0x07
<b>1</b>	0	0	0	0	1	0	1	1	0x0B
<b>2</b>	0	0	0	0	1	1	0	1	0x0D
<b>3</b>	0	0	0	0	1	1	1	0	0x0E
<b>4</b>	0	0	0	1	0	0	1	1	0x13
<b>5</b>	0	0	0	1	0	1	0	1	0x15
<b>6</b>	0	0	0	1	0	1	1	0	0x16
<b>7</b>	0	0	0	1	1	0	0	1	0x19
<b>8</b>	0	0	0	1	1	0	1	0	0x1A
<b>9</b>	0	0	0	1	1	1	0	0	0x1C
<b>10</b>	0	0	1	0	0	0	1	1	0x23
<b>11</b>	0	0	1	0	0	1	0	1	0x25
<b>12</b>	0	0	1	0	0	1	1	0	0x26

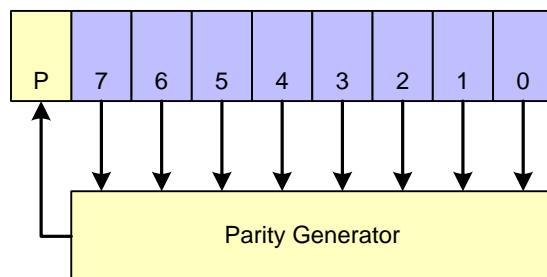
Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
13	0	0	1	0	1	0	0	1	0x29
14	0	0	1	0	1	0	1	0	0x2A
15	0	0	1	0	1	1	0	0	0x2C
16	0	0	1	1	0	0	0	1	0x31
17	0	0	1	1	0	0	1	0	0x32
18	0	0	1	1	0	1	0	0	0x34
19	0	0	1	1	1	0	0	0	0x38
20	0	0	0	1	1	1	1	1	0x1F
21	0	0	1	0	1	1	1	1	0x2F
22	0	0	1	1	0	1	1	1	0x37
23	0	0	1	1	1	0	1	1	0x3B
24	0	1	0	0	0	0	1	1	0x43
25	0	1	0	0	0	1	0	1	0x45
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
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	0	1	1	1	1	0	1	0x3D
45	0	0	1	1	1	1	1	0	0x3E
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

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
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

931 **9.5.3 ECC Generation on TX Side**

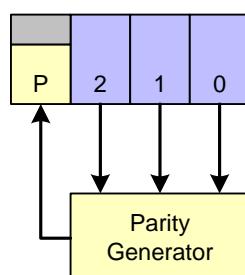
932 This is an informative section.

933 The ECC can be easily implemented using a parallel approach as depicted in Figure 37 for a 64-bit header.



934 **Figure 37 64-bit ECC Generation on TX Side**

935 And Figure 38 for a 24-bit header:



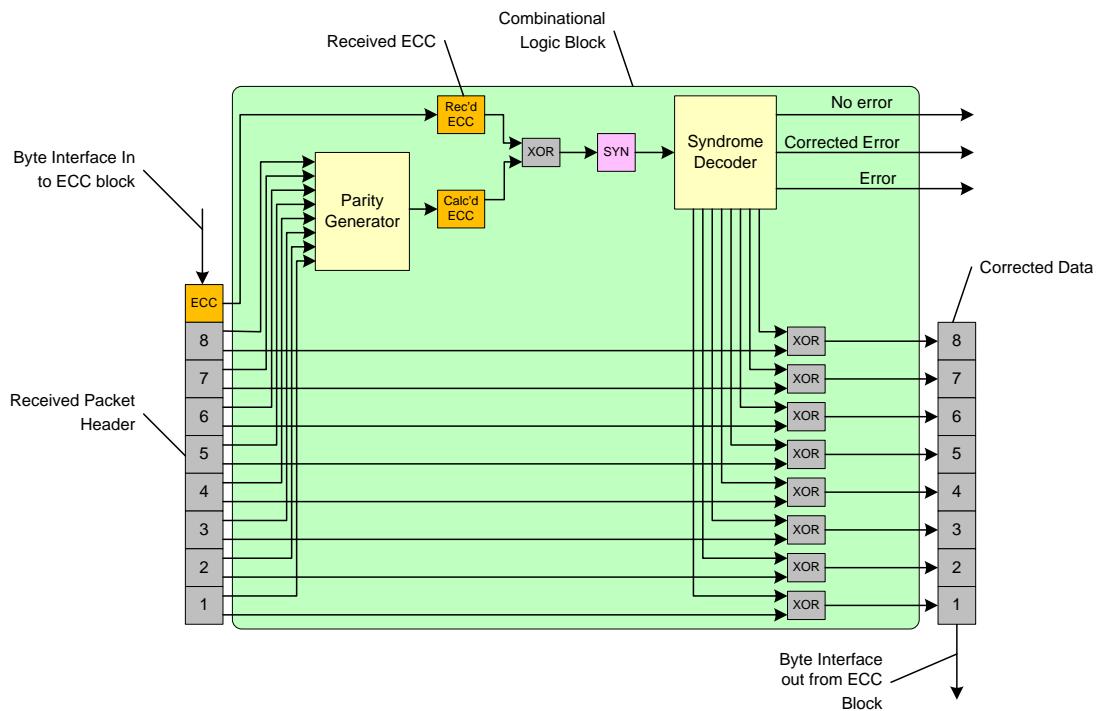
936 **Figure 38 24-bit ECC Generation on TX Side**

939 The parity generators are based on Table 5.

940 e.g.  $P_{324\text{-bit}} = D_1 \wedge D_2 \wedge D_3 \wedge D_7 \wedge D_8 \wedge D_9 \wedge D_{13} \wedge D_{14} \wedge D_{15} \wedge D_{19} \wedge D_{20} \wedge D_{21} \wedge D_{23}$

#### 941 9.5.4 Applying ECC on RX Side

942 Applying ECC on RX side involves generating a new ECC for the received packet, computing the syndrome  
943 using the new ECC and the received ECC, decoding the syndrome to find if a single-error has occurred and  
944 if so, correct it.



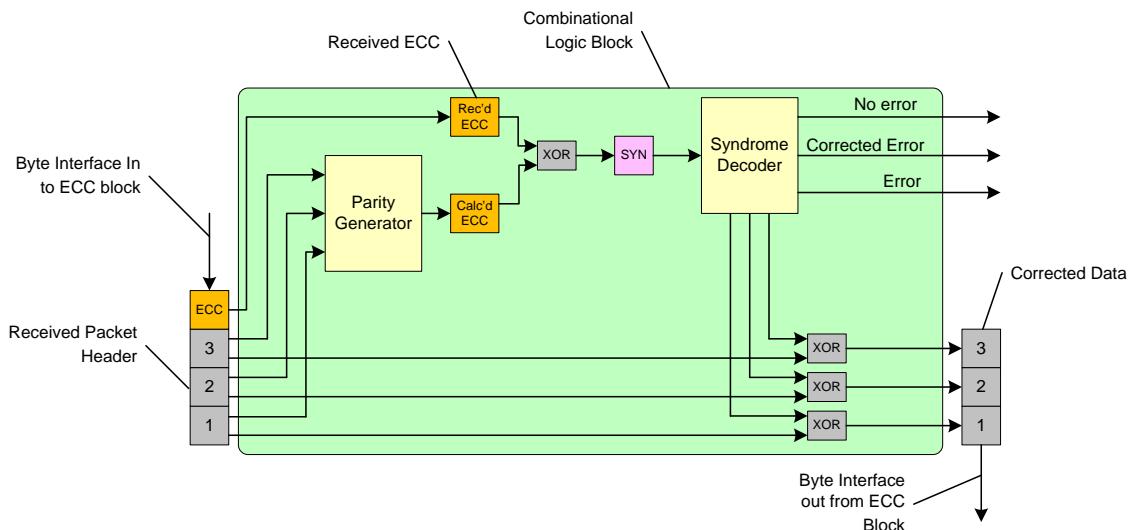
945

946 **Figure 39 64-bit ECC on RX Side Including Error Correction**

947 Decoding the syndrome has three aspects:

- 948 • Finding if the packet has any errors (if syndrome is 0, no errors are present)
- 949 • Checking if a single error has occurred by searching Table 5, if the syndrome is one of the entries  
950 in the table, then a single bit error has occurred and the corresponding bit is affected, thus this  
951 position in the data stream needs to be complemented. Also, if the syndrome is one of the rows of  
952 the identity matrix I, then one of the parity bits are in error. If the syndrome cannot be identified,  
953 then a higher order error has occurred and the error flag will be set (the stream is corrupted and  
954 cannot be restored).
- 955 • Correcting the single error detected, as previously indicated.

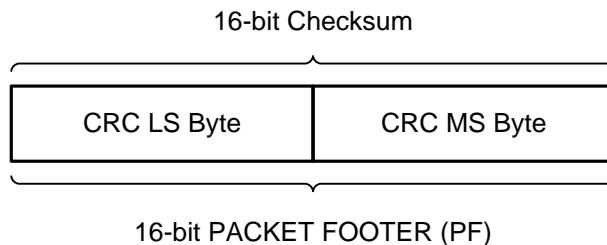
956 The 24-bit implementation uses fewer terms to calculate the parity and thus the syndrome decoding block is  
957 much simpler than the 64-bit implementation.



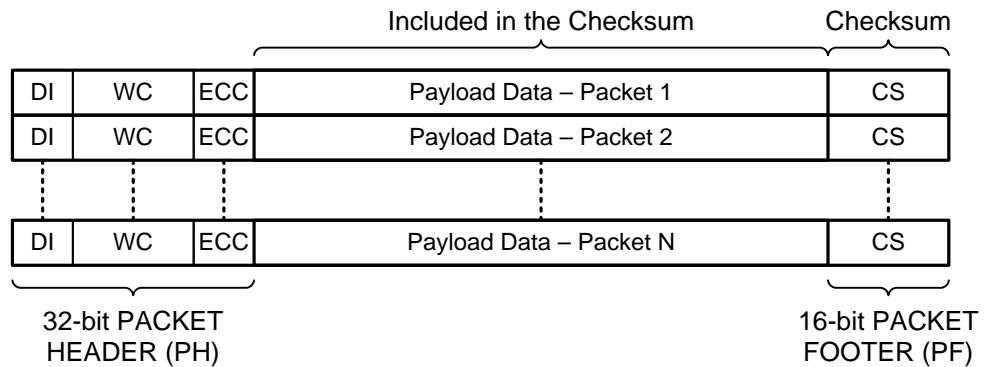
## 960 **9.6 Checksum Generation**

961 To detect possible errors in transmission, a checksum is calculated over each data packet. The checksum is  
962 realized as 16-bit CRC. The generator polynomial is  $x^{16}+x^{12}+x^5+x^0$ .

963 The transmission of the checksum is illustrated in Figure 41.



966 The 16-bit checksum sequence is transmitted as part of the Packet Footer. When the Word Count is zero, the  
967 CRC shall be 0xFFFF.

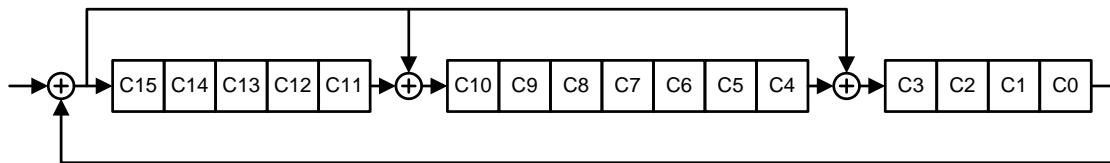


968

969

**Figure 42 Checksum Generation for Packet Data**

970 The definition of a serial CRC implementation is presented in Figure 43. The CRC implementation shall be  
971 functionally equivalent with the C code presented in Figure 44. The CRC shift register is initialized to  
972 0xFFFF at the beginning of each packet. After all payload data has passed through the CRC circuitry, the  
973 CRC circuitry contains the checksum. The 16-bit checksum produced by the C code in Figure 44 equals the  
974 final contents of the C[15:0] shift register shown in Figure 43. The checksum is then sent over CSI-2 bus to  
975 the receiver to verify that no errors have occurred in the transmission.



**Polynomial:**  $x^{16} + x^{12} + x^5 + x^0$   
Note: C15 represents  $x^0$ , C0 represents  $x^{15}$

976

977

**Figure 43 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);
}
```

978

979

#### Figure 44 16-bit CRC Software Implementation Example

980 The data and checksum are transmitted least significant byte first. Each bit within a byte is transmitted least  
981 significant bit first.  
982

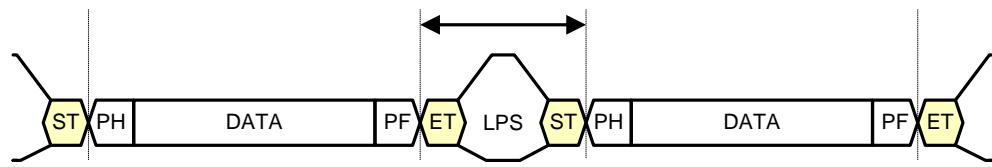
983 Data:  
984 FF 00 00 02 B9 DC F3 72 BB D4 B8 5A C8 75 C2 7C 81 F8 05 DF FF 00 00 01  
985 Checksum LS byte and MS byte:  
986 F0 00  
987  
988 Data:  
989 FF 00 00 00 1E F0 1E C7 4F 82 78 C5 82 E0 8C 70 D2 3C 78 E9 FF 00 00 01  
990 Checksum LS byte and MS byte:  
991 69 E5

## 992 9.7 Packet Spacing

993 Between Low Level Protocol packets there must always be a transition into and out of the Low Power State  
994 (LPS). Figure 45 illustrates the packet spacing with the LPS.

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

**SHORT / LONG PACKET SPACING:**  
Variable - always a LPS between packets



**KEY:**

LPS – Low Power State  
ST – Start of Transmission  
ET – End of Transmission  
PH – Packet Header  
PF – Packet Footer  
SP – Short Packet

997

998

**Figure 45 Packet Spacing**

999 **9.8 Synchronization Short Packet Data Type Codes**

1000 Short Packet Data Types shall be transmitted using only the Short Packet format. See Section 9.1.2 for a  
1001 format description.

1002 **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

1003 **9.8.1 Frame Synchronization Packets**

1004 Each image frame shall begin with a Frame Start (FS) Packet containing the Frame Start Code. The FS Packet  
1005 shall be followed by one or more long packets containing image data and zero or more short packets  
1006 containing synchronization codes. Each image frame shall end with a Frame End (FE) Packet containing the  
1007 Frame End Code. See Table 6 for a description of the synchronization code data types.

1008 For FS and FE synchronization packets the Short Packet Data Field shall contain a 16-bit frame number. This  
1009 frame number shall be the same for the FS and FE synchronization packets corresponding to a given frame.

1010 The 16-bit frame number, when used, shall be non-zero to distinguish it from the use-case where frame  
1011 number is inoperative and remains set to zero.

1012 The behavior of the 16-bit frame number shall be as one of the following

- 1013 • Frame number is always zero – frame number is inoperative.

- 1014     • Frame number increments by 1 for every FS packet with the same Virtual Channel and is  
1015       periodically reset to one e.g. 1, 2, 1, 2, 1, 2, 1, 2 or 1, 2, 3, 4, 1, 2, 3, 4
- 1016     The frame number must be a non-zero value.
- 1017     **9.8.2 Line Synchronization Packets**
- 1018     Line synchronization packets are optional.
- 1019     For Line Start (LS) and Line End (LE) synchronization packets the Short Packet Data Field shall contain a  
1020       16-bit line number. This line number shall be the same for the LS and LE packets corresponding to a given  
1021       line. Line numbers are logical line numbers and are not necessarily equal to the physical line numbers
- 1022     The 16-bit line number, when used, shall be non-zero to distinguish it from the case where line number is  
1023       inoperative and remains set to zero.
- 1024     The behavior of the 16-bit line number shall be as one of the following:
- 1025       • Line number is always zero – line number is inoperative.
- 1026       • Line number increments by one for every LS packet within the same Virtual Channel and the same  
1027        Data Type. The line number is periodically reset to one for the first LS packet after a FS packet.  
1028        The intended usage is for progressive scan (non-interlaced) video data streams. The line number  
1029        must be a non-zero value.
- 1030       • Line number increments by the same arbitrary step value greater than one for every LS packet  
1031        within the same Virtual Channel and the same Data Type. The line number is periodically reset to  
1032        a non-zero arbitrary start value for the first LS packet after a FS packet. The arbitrary start value  
1033        may be different between successive frames. The intended usage is for interlaced video data  
1034        streams.

1035     **9.9 Generic Short Packet Data Type Codes**

1036     Table 7 lists the Generic Short Packet Data Types.

1037     **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

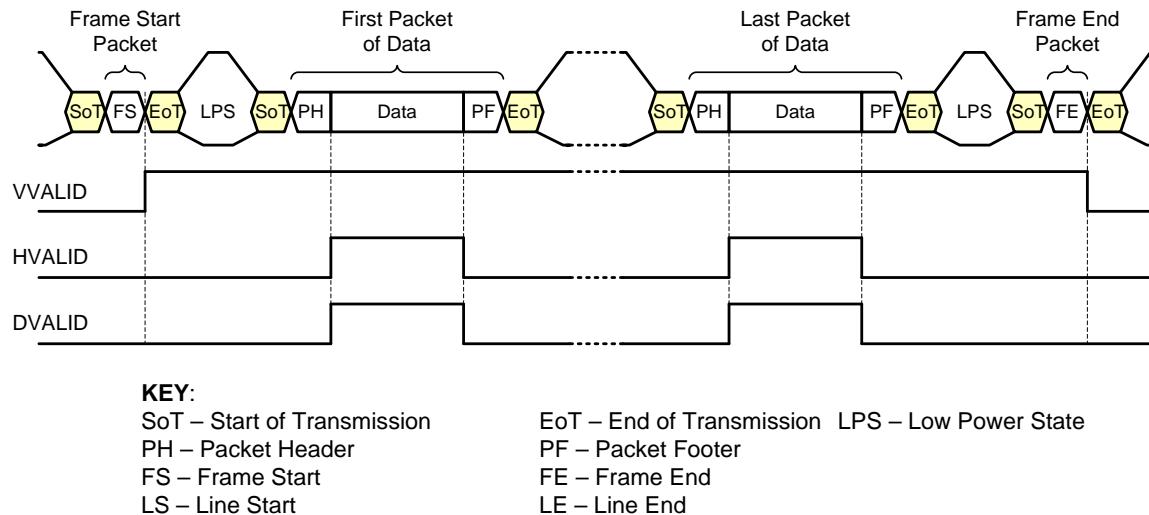
1038     The intention of the Generic Short Packet Data Types is to provide a mechanism for including timing  
1039       information for the opening/closing of shutters, triggering of flashes, etc within the data stream. The intent  
1040       of the 16-bit User defined data field in the generic short packets is to pass a data type value and a 16-bit data  
1041       value from the transmitter to application layer in the receiver. The CSI-2 receiver shall pass the data type  
1042       value and the associated 16-bit data value to the application layer.

## 1043 9.10 Packet Spacing Examples

1044 Packets are separated by an EoT, LPS, SoT sequence as defined in [MIPI01].

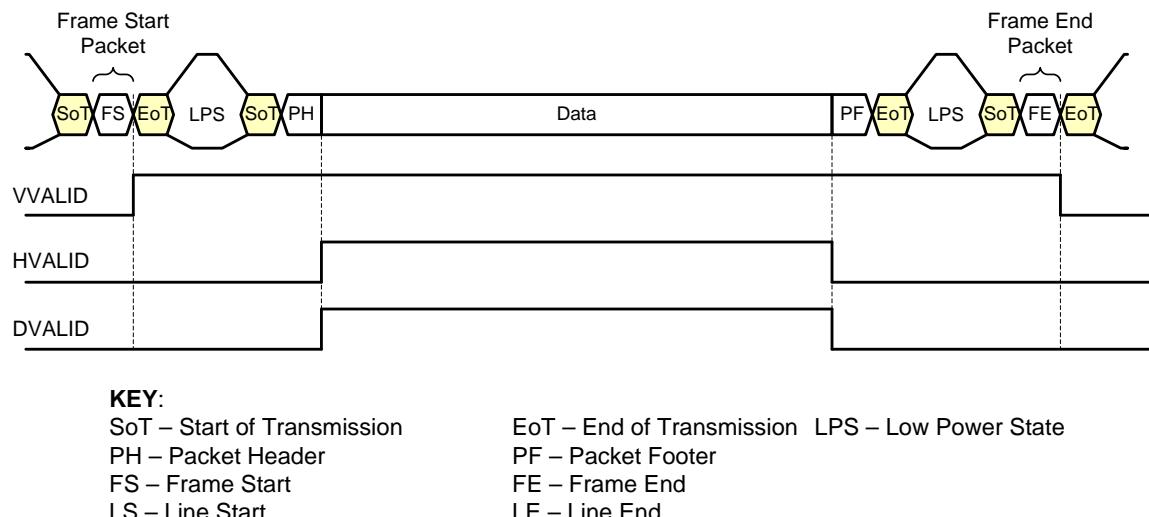
1045 Figure 46 and Figure 47 contain examples of data frames composed of multiple packets, and a single packet,  
1046 respectively.

1047 Note that the VVALID, HVALID and DVALID signals in the figures in this section are only concepts to  
1048 help illustrate the behavior of the frame start/end and line start/end packets. The VVALID, HVALID and  
1049 DVALID signals do not form part of the Specification.



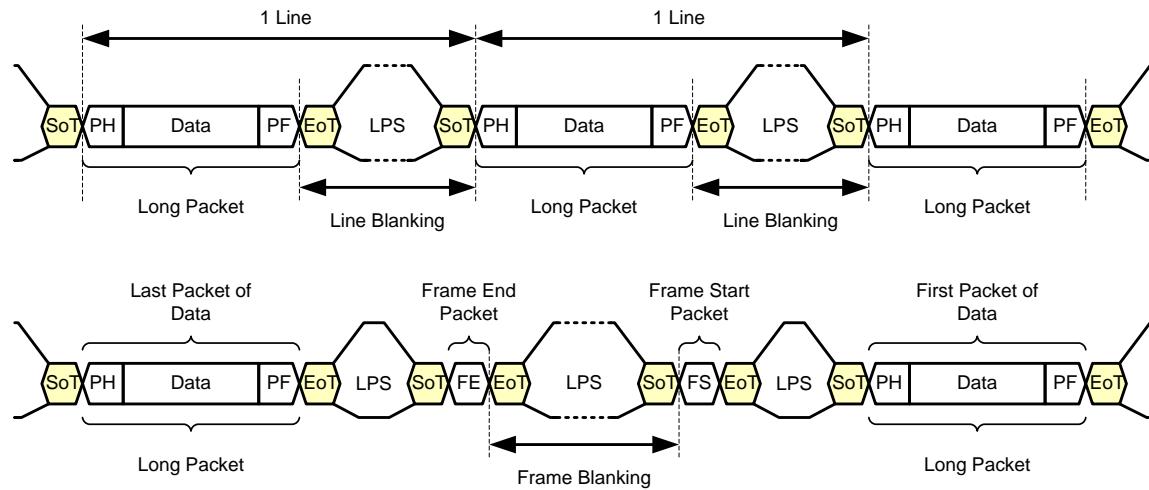
1050  
1051

Figure 46 Multiple Packet Example



1052  
1053

Figure 47 Single Packet Example



**KEY:**

SoT – Start of Transmission  
PH – Packet Header  
FS – Frame Start  
LS – Line Start

EoT – End of Transmission LPS – Low Power State  
PF – Packet Footer  
FE – Frame End  
LE – Line End

1054

1055

**Figure 48 Line and Frame Blanking Definitions**

1056 The period between the Packet Footer of one long packet and the Packet Header of the next long packet is  
1057 called the Line Blanking Period.

1058 The period between the Frame End packet in frame N and the Frame Start packet in frame N+1 is called the  
1059 Frame Blanking Period (Figure 48).

1060 The Line Blanking Period is not fixed and may vary in length. The receiver should be able to cope with a  
1061 near zero Line Blanking Period as defined in [MIPI01]. The transmitter defines the minimum time for the  
1062 Frame Blanking Period. The Frame Blanking Period duration should be programmable in the transmitter.

1063 Frame Start and Frame End packets shall be used.

1064 Recommendations (informative) for frame start and end packet spacing:

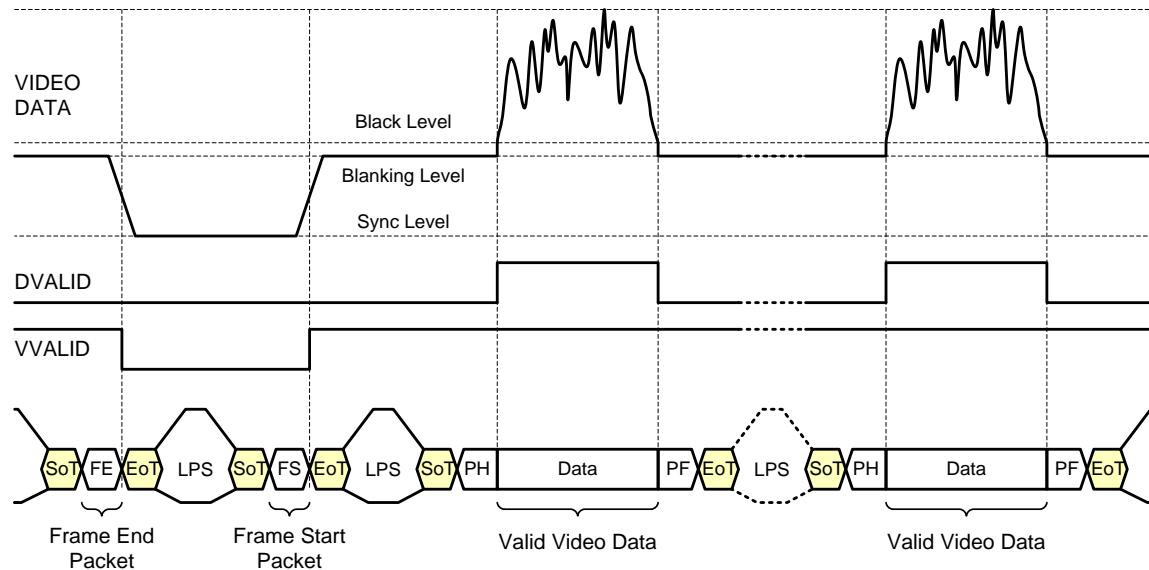
- The Frame Start packet to first data packet spacing should be as close as possible to the minimum packet spacing
- The last data packet to Frame End packet spacing should be as close as possible to the minimum packet spacing

1069 The intention is to ensure that the Frame Start and Frame End packets accurately denote the start and end of  
1070 a frame of image data. A valid exception is when the positions of the Frame Start and Frame End packets are  
1071 being used to convey pixel level accurate vertical synchronization timing information.

1072 The positions of the Frame Start and Frame End packets can be varied within the Frame Blanking Period in  
1073 order to provide pixel level accurate vertical synchronization timing information. See Figure 49.

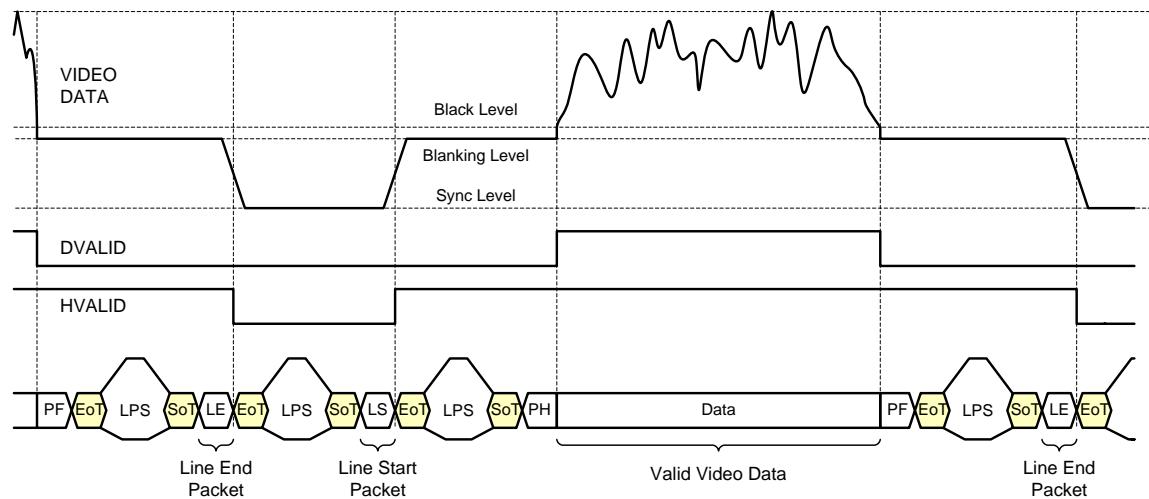
1074 Line Start and Line End packets shall be used for pixel level accurate horizontal synchronization timing  
1075 information.

1076 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 50.



1078  
1079

**Figure 49 Vertical Sync Example**



1080  
1081

**Figure 50 Horizontal Sync Example**

## 1082 9.11 Packet Data Payload Size Rules

1083 For YUV, RGB or RAW data types, one long packet shall contain one line of image data. Each long packet  
1084 of the same Data Type shall have equal length when packets are within the same Virtual Channel and when  
1085 packets are within the same frame. An exception to this rule is the YUV420 data type which is defined in  
1086 Section 11.2.2.

1087 For User Defined Byte-based Data Types, long packets can have arbitrary length. The spacing between  
1088 packets can also vary.

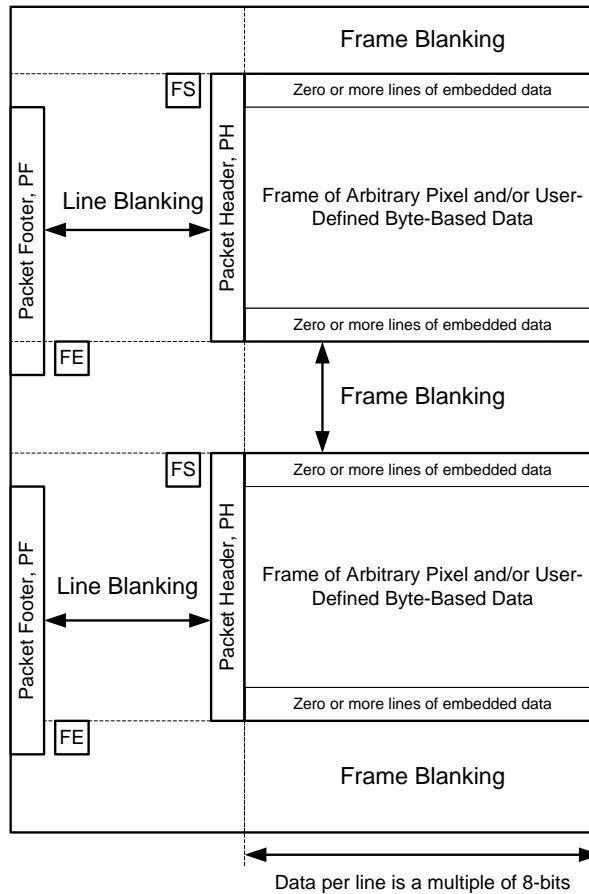
1089 The total size of data within a long packet for all data types shall be a multiple of eight bits. However, it is  
1090 also possible that a data type's payload data transmission format, as defined elsewhere in this Specification,  
1091 imposes additional constraints on payload size. In order to meet these constraints it may sometimes be  
1092 necessary to add some number of "padding" pixels to the end of a payload e.g., when a packet with the  
1093 RAW10 data type contains an image line whose length is not a multiple of four pixels as required by the  
1094 RAW10 transmission format as described in Section 11.4.4. The values of such padding pixels are not  
1095 specified.

1096 **9.12 Frame Format Examples**

1097 This is an informative section.

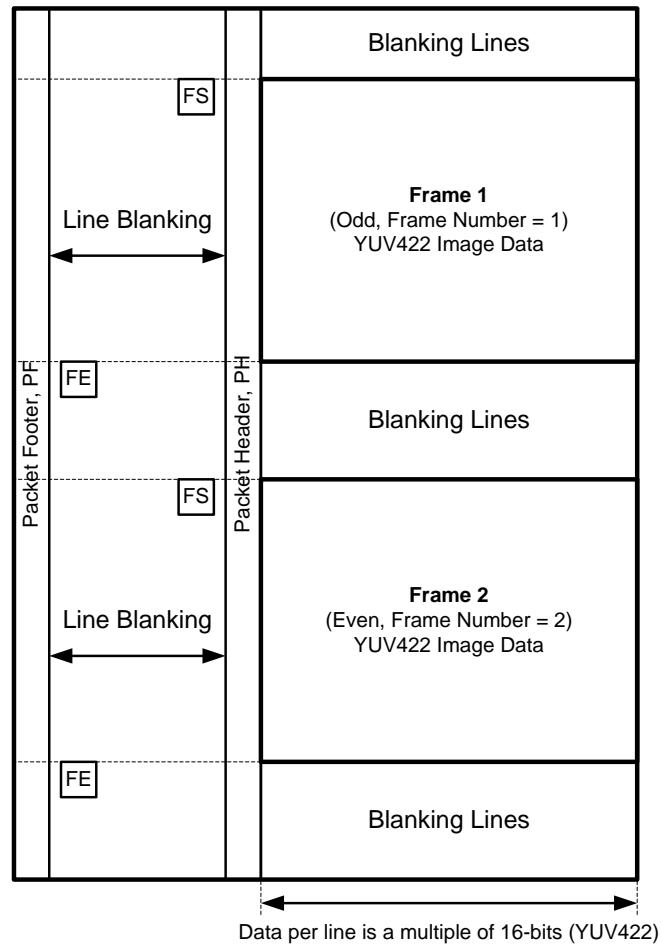
1098 This section contains three examples to illustrate how the CSI-2 features can be used.

- 1099 • General Frame Format Example, Figure 51  
1100 • Digital Interlaced Video Example, Figure 52  
1101 • Digital Interlaced Video with accurate synchronization timing information, Figure 53



1102  
1103

**Figure 51 General Frame Format Example**



**KEY:**

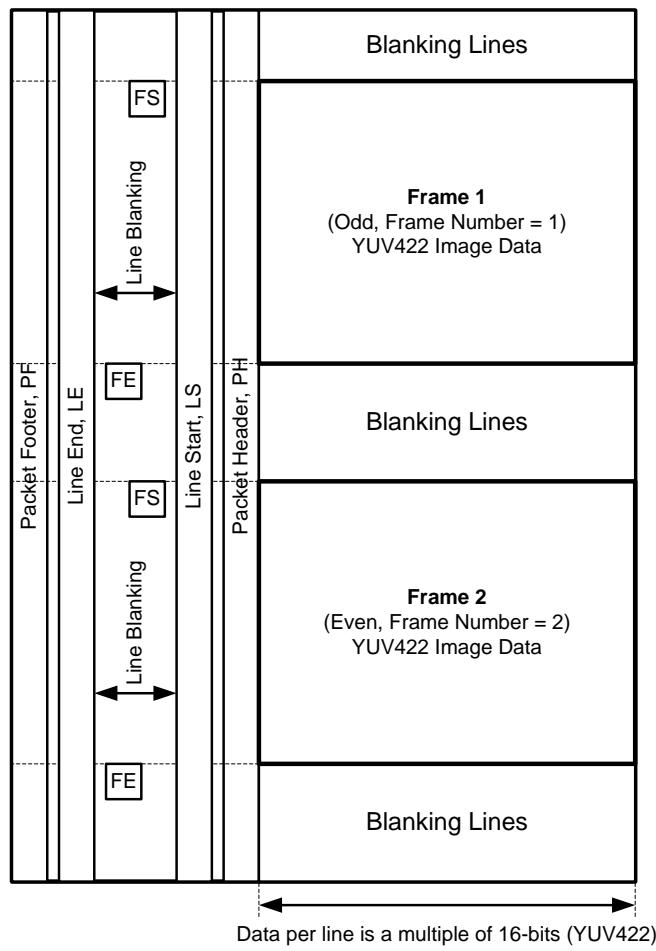
PH – Packet Header  
FS – Frame Start  
LS – Line Start

PF – Packet Footer  
FE – Frame End  
LE – Line End

1104

1105

**Figure 52 Digital Interlaced Video Example**



1106

1107 **Figure 53 Digital Interlaced Video with Accurate Synchronization Timing Information**

1108 **9.13 Data Interleaving**

1109 The CSI-2 supports the interleaved transmission of different image data formats within the same video data  
1110 stream.

1111 There are two methods to interleave the transmission of different image data formats:

- 1112 • Data Type  
1113 • Virtual Channel Identifier

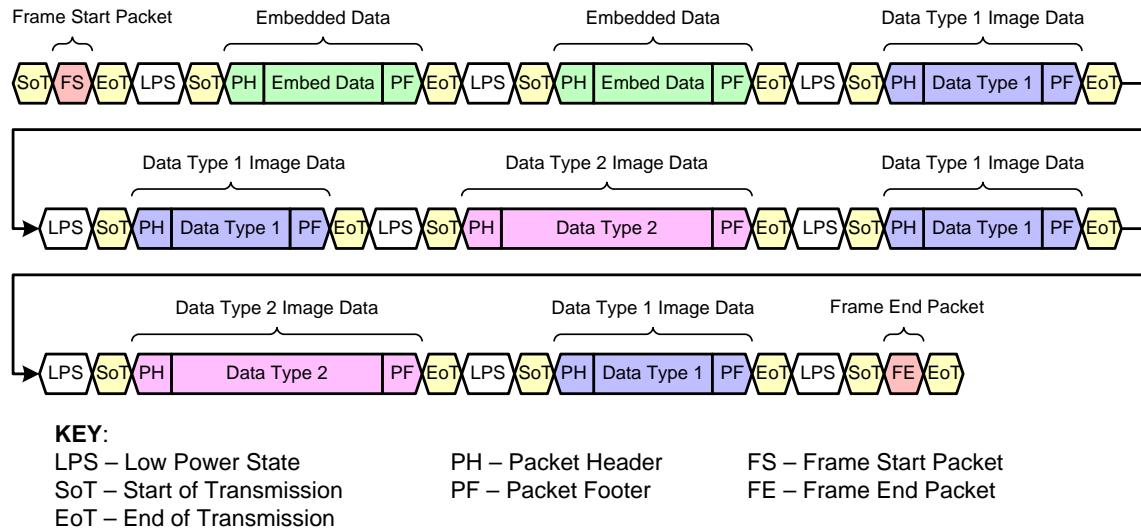
1114 The preceding methods of interleaved data transmission can be combined in any manner.

1115 **9.13.1 Data Type Interleaving**

1116 The Data Type value uniquely defines the data format for that packet of data. The receiver uses the Data  
1117 Type value in the packet header to de-multiplex data packets containing different data formats as illustrated  
1118 in Figure 54. Note, in the figure the Virtual Channel Identifier is the same in each Packet Header.

- 1119 The packet payload data format shall agree with the Data Type code in the Packet Header as follows:
- 1120 • For defined image data types – any non-reserved codes in the range 0x18 to 0x3F – only the single  
1121 corresponding MIPI-defined packet payload data format shall be considered correct
- 1122 • Reserved image data types – any reserved codes in the range 0x18 to 0x3F – shall not be used. No  
1123 packet payload data format shall be considered correct for reserved image data types
- 1124 • For generic long packet data types (codes 0x10 thru 0x17) and user-defined, byte-based (codes  
1125 0x30 – 0x37), any packet payload data format shall be considered correct
- 1126 • Generic long packet data types (codes 0x10 thru 0x17) and user-defined, byte-based (codes 0x30 –  
1127 0x37), should not be used with packet payloads that meet any MIPI image data format definition
- 1128 • Synchronization short packet data types (codes 0x00 thru 0x07) shall consist of only the header  
1129 and shall not include payload data bytes
- 1130 • Generic short packet data types (codes 0x08 thru 0x0F) shall consist of only the header and shall  
1131 not include payload data bytes

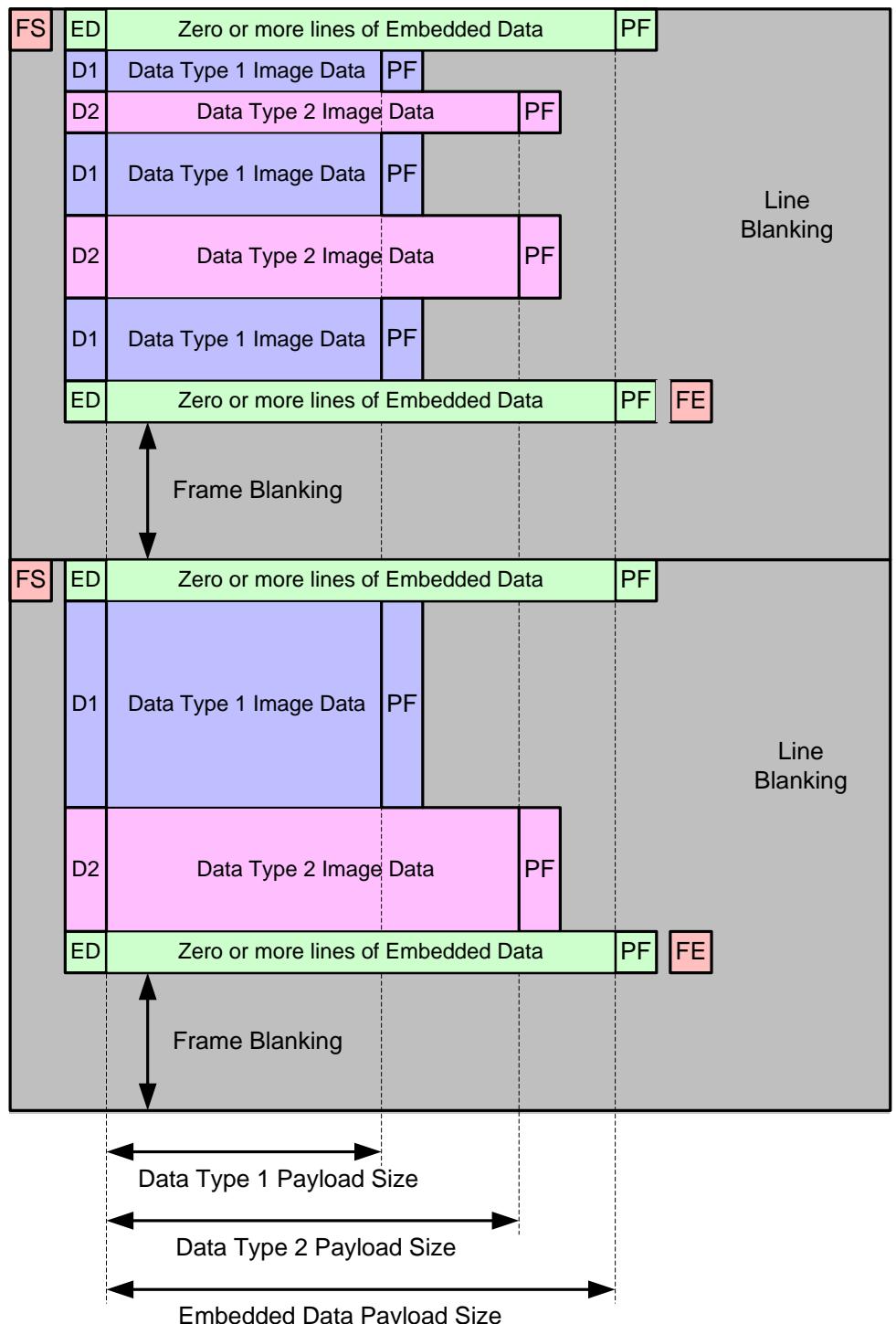
1132 Data formats are defined further in Section 11.



**Figure 54 Interleaved Data Transmission using Data Type Value**

1135 All of the packets within the same virtual channel, independent of the Data Type value, share the same frame  
1136 start/end and line start/end synchronization information. By definition, all of the packets, independent of data  
1137 type, between a Frame Start and a Frame End packet within the same virtual channel belong to the same  
1138 frame.

1139 Packets of different data types may be interleaved at either the packet level as illustrated in Figure 55 or the  
1140 frame level as illustrated in Figure 56. Data formats are defined in Section 11.



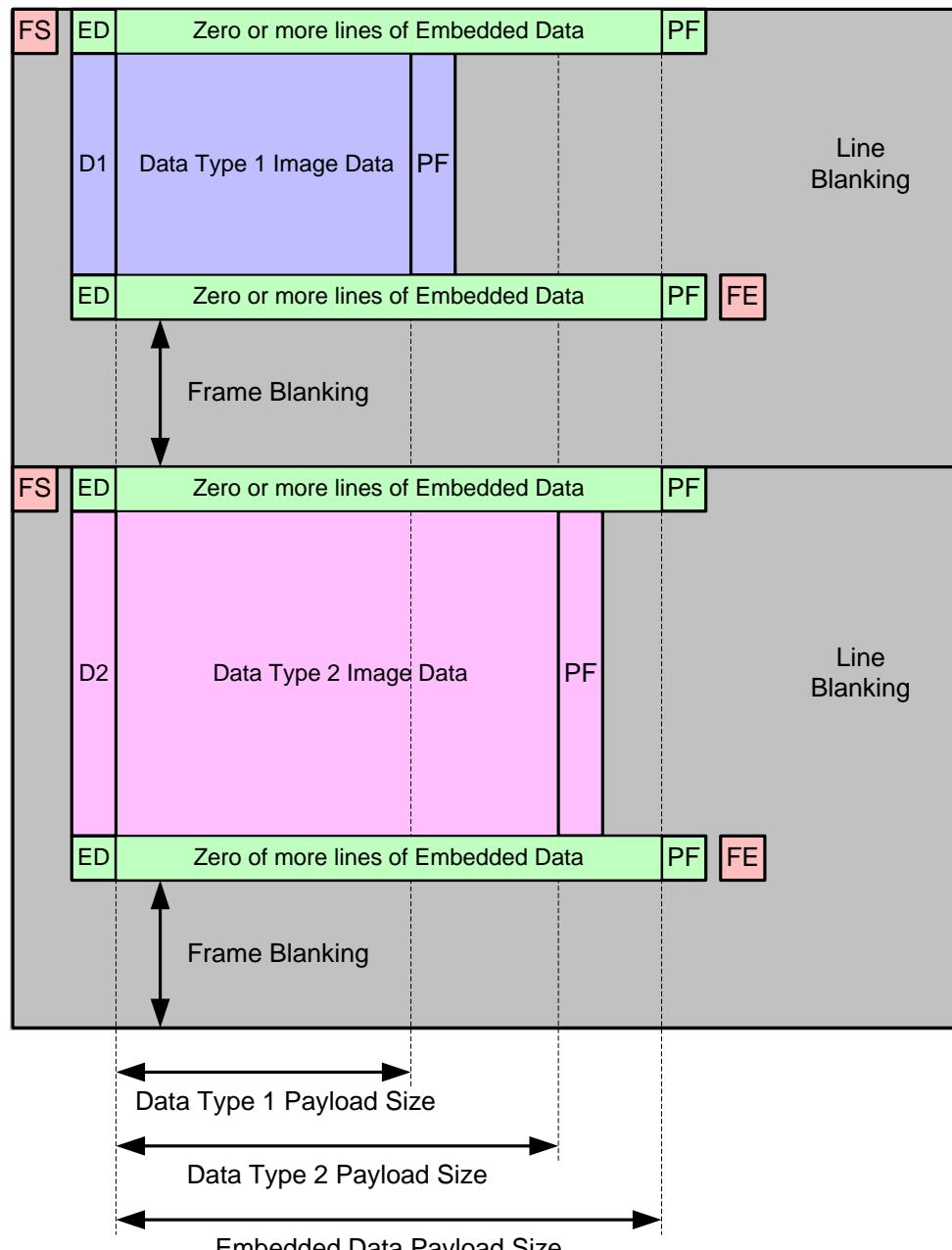
**KEY:**

- |                       |   |
|-----------------------|---|
| LPS – Low Power State | ED – Packet Header containing Embedded Data type code     |
| FS – Frame Start      | D1 – Packet Header containing Data Type 1 Image Data Code |
| FE – Frame End        | D2 – Packet Header containing Data Type 2 Image Data Code |
| PF – Packet Footer    |   |

1141

1142

**Figure 55 Packet Level Interleaved Data Transmission**



**KEY:**

LPS – Low Power State	ED – Packet Header containing Embedded Data type code
FS – Frame Start	D1 – Packet Header containing Data Type 1 Image Data Code
FE – Frame End	D2 – Packet Header containing Data Type 2 Image Data Code
PF – Packet Footer	

1143

1144

**Figure 56 Frame Level Interleaved Data Transmission**

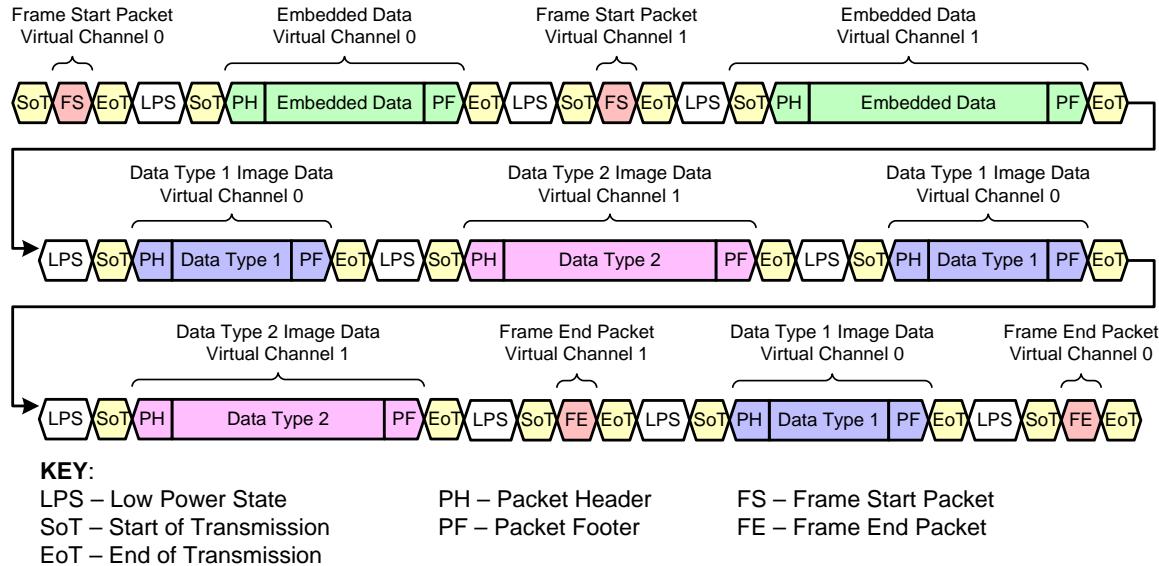
1145    **9.13.2 Virtual Channel Identifier Interleaving**

1146    The Virtual Channel Identifier allows different data types within a single data stream to be logically separated  
1147    from each other. Figure 57 illustrates data interleaving using the Virtual Channel Identifier.

1148 Each virtual channel has its own Frame Start and Frame End packet. Therefore, it is possible for different  
1149 virtual channels to have different frame rates, though the data rate for both channels would remain the same.

1150 In addition, Data Type value Interleaving can be used for each virtual channel, allowing different data types  
1151 within a virtual channel and a second level of data interleaving.

1152 Therefore, receivers should be able to de-multiplex different data packets based on the combination of the  
1153 Virtual Channel Identifier and the Data Type value. For example, data packets containing the same Data  
1154 Type value but transmitted on different virtual channels are considered to belong to different frames (streams)  
1155 of image data.



1156  
1157

**Figure 57 Interleaved Data Transmission using Virtual Channels**

1158 **10 Color Spaces**

1159 The color space definitions in this section are simply references to other standards. The references are  
1160 included only for informative purposes and not for compliance. The color space used is not limited to the  
1161 references given.

1162 **10.1 RGB Color Space Definition**

1163 In this Specification, the abbreviation RGB means the nonlinear sR'G'B' color space in 8-bit representation  
1164 based on the definition of sRGB in IEC 61966.

1165 The 8-bit representation results as RGB888. The conversion to the more commonly used RGB565 format is  
1166 achieved by scaling the 8-bit values to five bits (blue and red) and six bits (green). The scaling can be done  
1167 either by simply dropping the LSBs or rounding.

1168 **10.2 YUV Color Space Definition**

1169 In this Specification, the abbreviation YUV refers to the 8-bit gamma corrected Y'CBCR color space defined  
1170 in ITU-R BT601.4.

1171

## 11 Data Formats

1172  
1173  
1174  
1175

The intent of this section is to provide a definitive reference for data formats typically used in CSI-2 applications. Table 8 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.

1176  
1177  
1178

The formats most widely used in CSI-2 applications are distinguished by a “primary” designation in Table 8. 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.

1179  
1180

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.

1181

**Table 8 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

Data Format	Primary	Secondary
Generic 8-bit Long Packet Data Types	P	
User Defined Byte-based Data (Note 1)	P	

1182 Notes:

1183 1. Compressed image data should use the user defined, byte-based data type codes

1184 For clarity the Start of Transmission and End of Transmission sequences in the figures in this section have  
1185 been omitted.

## 1186 11.1 Generic 8-bit Long Packet Data Types

1187 Table 9 defines the generic 8-bit Long packet data types.

1188 **Table 9 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

### 1189 11.1.1 Null and Blanking Data

1190 For both the null and blanking data types the receiver must ignore the content of the packet payload data.

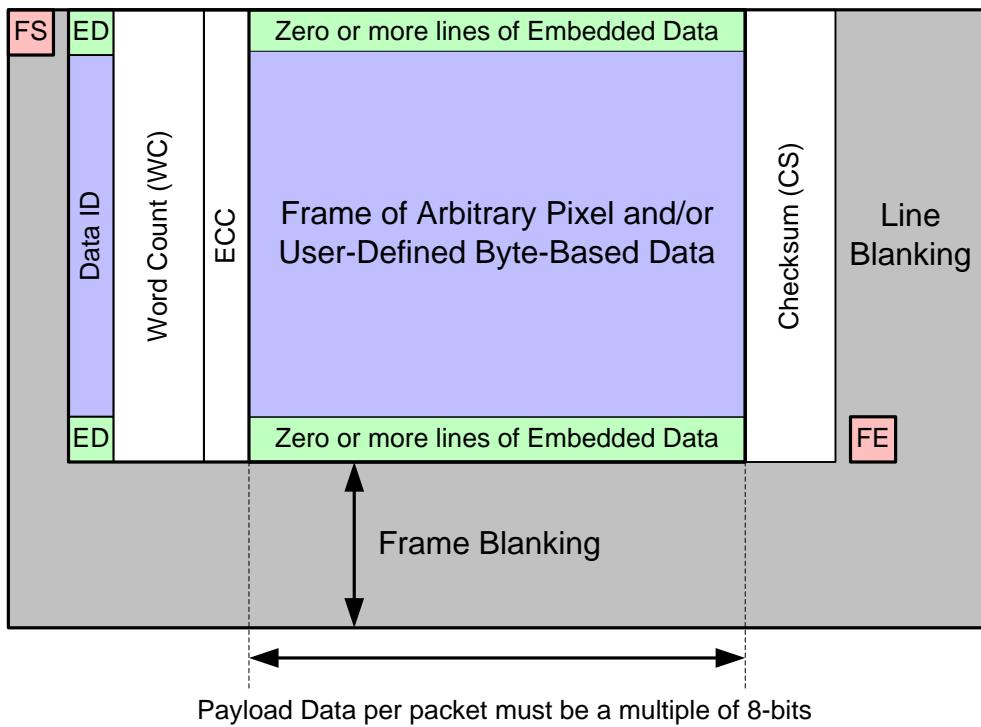
1191 A blanking packet differs from a null packet in terms of its significance within a video data stream. A null  
1192 packet has no meaning whereas the blanking packet may be used, for example, as the blanking lines between  
1193 frames in an ITU-R BT.656 style video stream.

### 1194 11.1.2 Embedded Information

1195 It is possible to embed extra lines containing additional information to the beginning and to the end of each  
1196 picture frame as presented in the Figure 58. If embedded information exists, then the lines containing the  
1197 embedded data must use the embedded data code in the data identifier.

1198 There may be zero or more lines of embedded data at the start of the frame. These lines are termed the frame  
1199 header.

1200 There may be zero or more line of embedded data at the end of the frame. These lines are termed the frame  
1201 footer.



1202 **KEY:**

1203 LPS – Low Power State                  DI – Data Identifier                  WC – Word Count  
1203 ECC – Error Correction Code        CS – Checksum                  ED – Embedded Data  
1203 FS – Frame Start                      FE – Frame End  
1203 LS – Line Start                        LE – Line End

1203 **Figure 58 Frame Structure with Embedded Data at the Beginning and End of the Frame**

1204 **11.2 YUV Image Data**

1205 Table 10 defines the data type codes for YUV data formats described in this section. The number of lines  
1206 transmitted for the YUV420 data type shall be even.

1207 YUV420 data formats are divided into legacy and non-legacy data formats. The legacy YUV420 data format  
1208 is for compatibility with existing systems. The non-legacy YUV420 data formats enable lower cost  
1209 implementations.

1210

**Table 10 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

1211 **11.2.1 Legacy YUV420 8-bit**

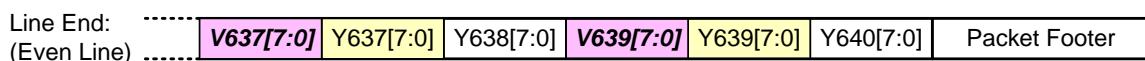
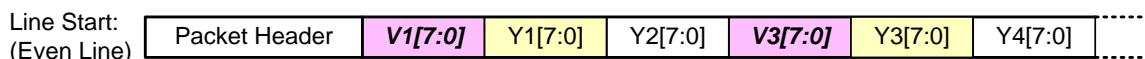
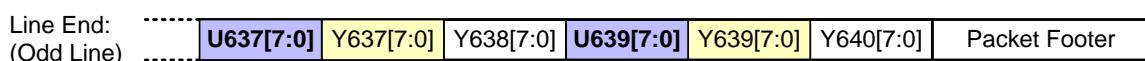
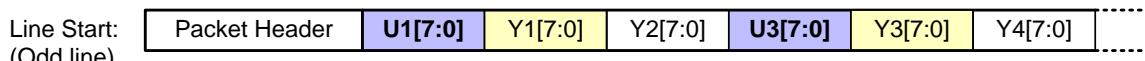
1212 Legacy YUV420 8-bit data transmission is performed by transmitting UYY... / VYY... sequences in odd /  
1213 even lines. U component is transferred in odd lines (1, 3, 5 ...) and V component is transferred in even lines  
1214 (2, 4, 6 ...). This sequence is illustrated in Figure 59.

1215 Table 11 specifies the packet size constraints for YUV420 8-bit packets. Each packet must be a multiple of  
1216 the values in the table.

1217 **Table 11 Legacy YUV420 8-bit Packet Data Size Constraints**

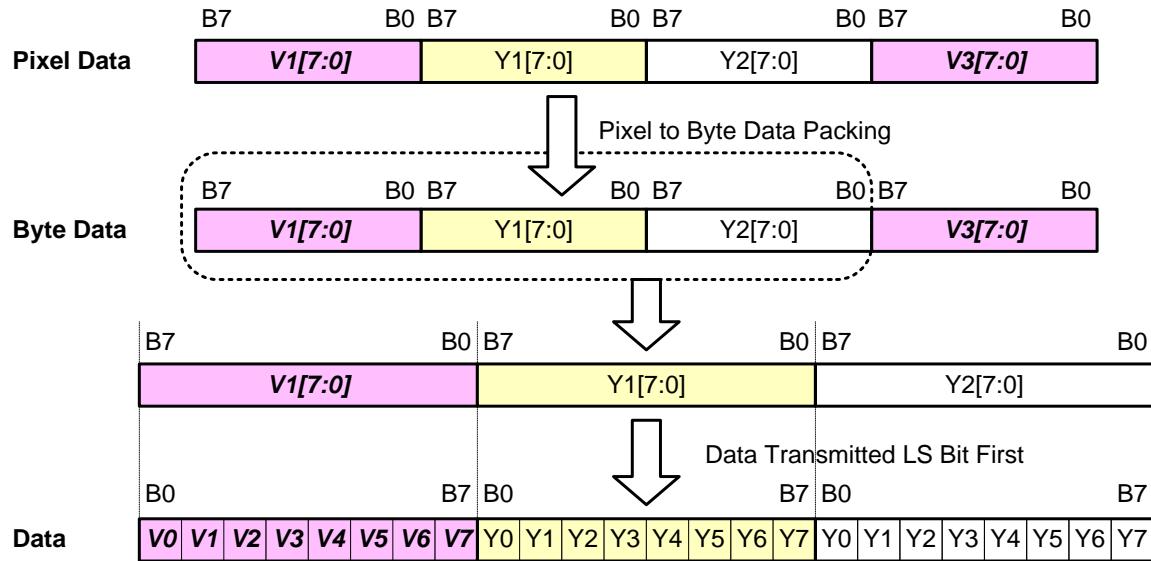
Pixels	Bytes	Bits
2	3	24

1218 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated  
1219 in Figure 60.



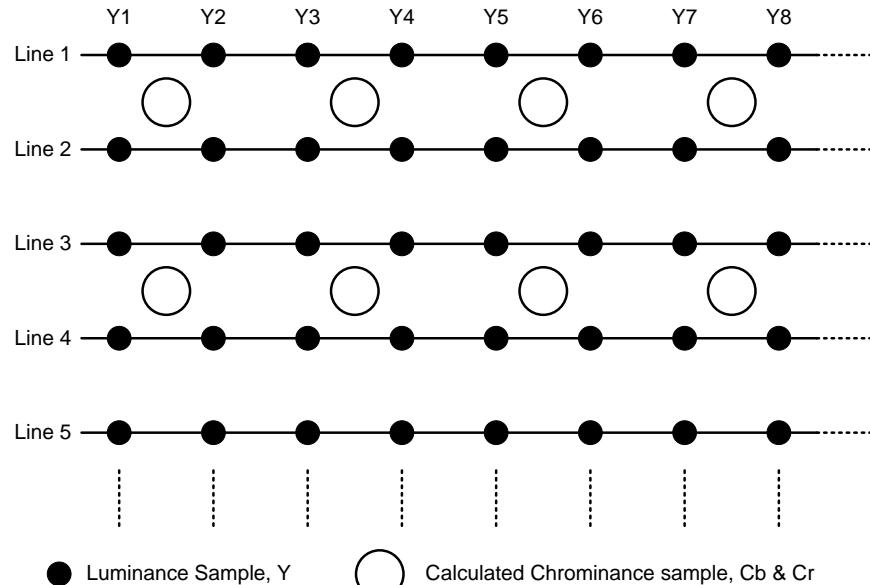
1220

1221 **Figure 59 Legacy YUV420 8-bit Transmission**



1224 There is one spatial sampling option

- H.261, H.263 and MPEG1 Spatial Sampling (Figure 61).



1226  
1227 **Figure 61 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1**

1228

1229

**Figure 62 Legacy YUV420 8-bit Frame Format**

1230 **11.2.2 YUV420 8-bit**

1231 YUV420 8-bit data transmission is performed by transmitting YYYY... / UYVYUYVY... sequences in odd  
1232 / even lines. Only the luminance component (Y) is transferred for odd lines (1, 3, 5...) and both luminance  
1233 (Y) and chrominance (U and V) components are transferred for even lines (2, 4, 6...). The format for the  
1234 even lines (UYVY) is identical to the YUV422 8-bit data format. The data transmission sequence is illustrated  
1235 in Figure 63.

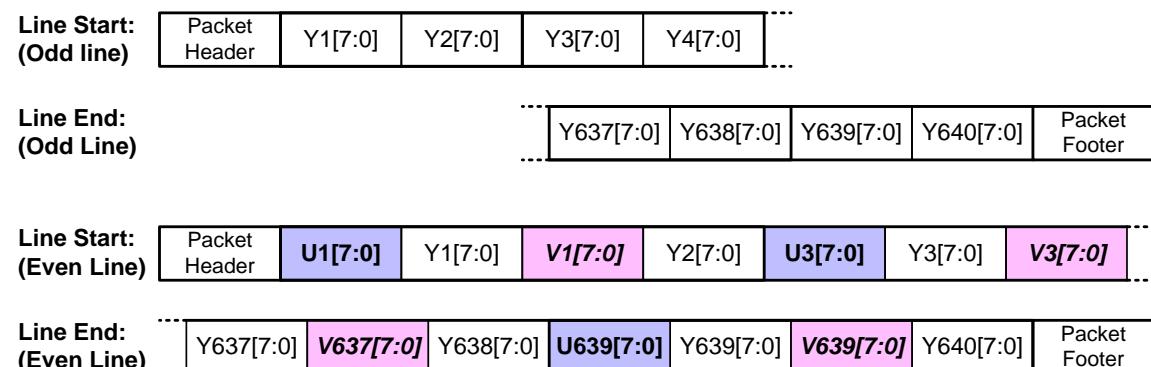
1236 The payload data size, in bytes, for even lines (UYVY) is double the payload data size for odd lines (Y). This  
1237 is exception to the general CSI-2 rule that each line shall have an equal length.

1238 Table 12 specifies the packet size constraints for YUV420 8-bit packets. Each packet must be a multiple of  
1239 the values in the table.

1240 **Table 12 YUV420 8-bit Packet Data Size Constraints**

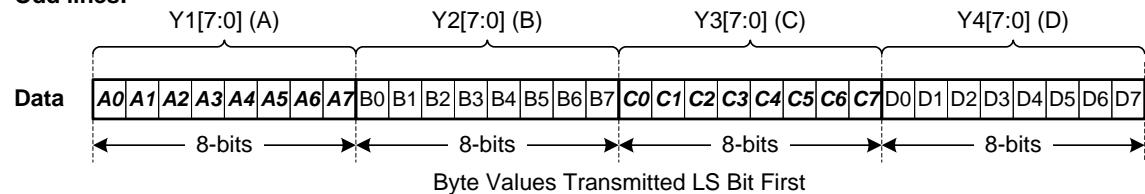
Odd Lines (1, 3, 5...) Luminance Only, Y			Even Lines (2, 4, 6...) Luminance and Chrominance, UYVY		
Pixels	Bytes	Bits	Pixels	Bytes	Bits
2	2	16	2	4	32

1241 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated  
1242 in Figure 64.

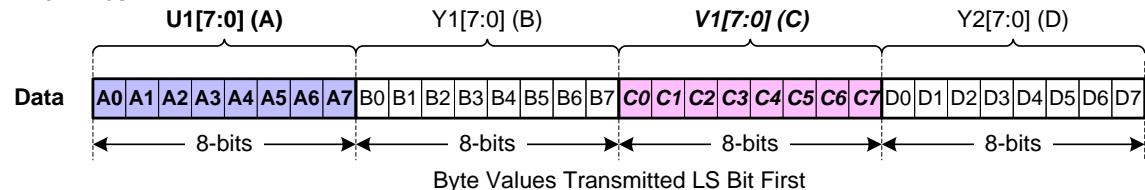


1243  
1244 **Figure 63 YUV420 8-bit Data Transmission Sequence**

**Odd lines:**



**Even lines:**



1245

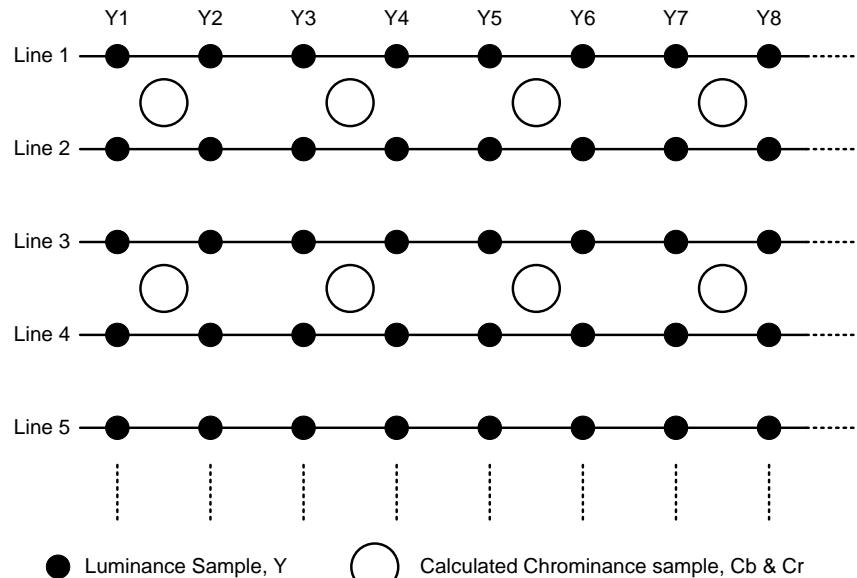
1246

**Figure 64 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration**

1247 There are two spatial sampling options

- 1248 • H.261, H.263 and MPEG1 Spatial Sampling (Figure 65).
- 1249 • Chroma Shifted Pixel Sampling (CSPS) for MPEG2, MPEG4 (Figure 66).

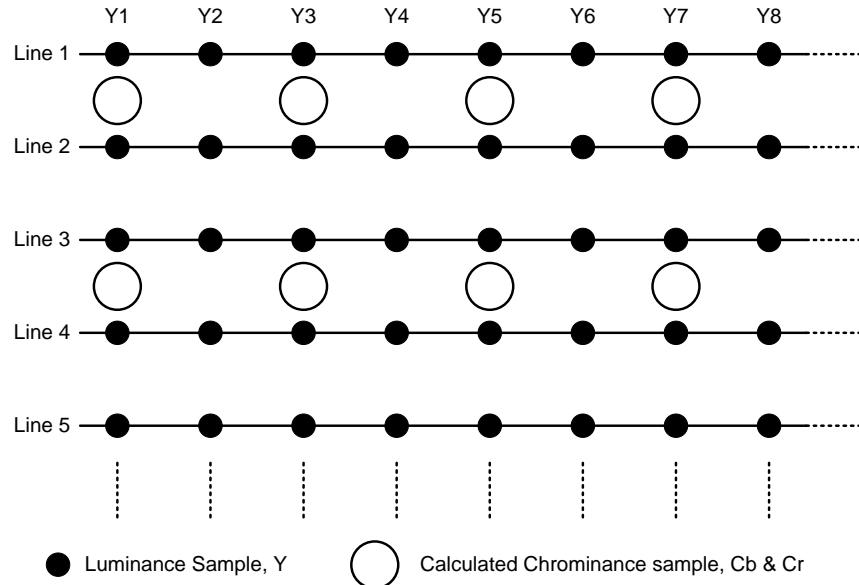
1250 Figure 67 shows the YUV420 frame format.



1251

1252

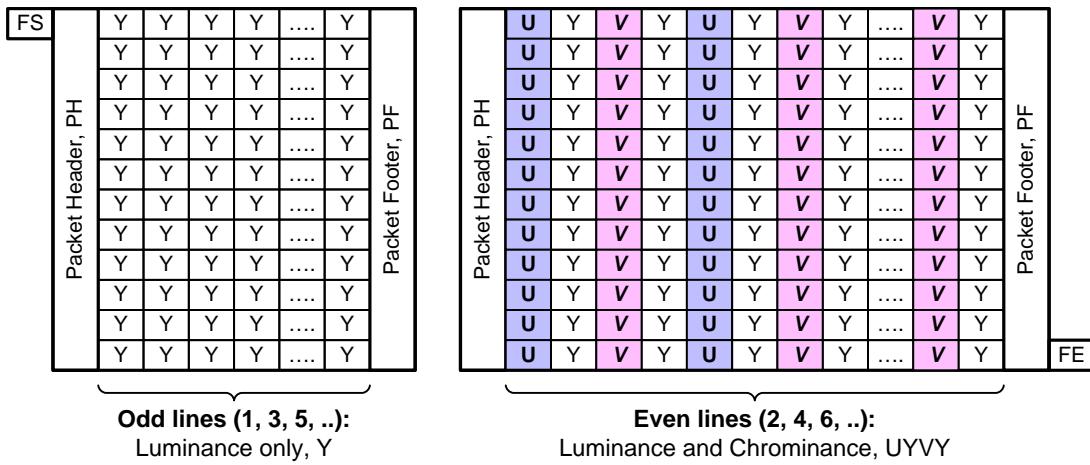
**Figure 65 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1**



1253

1254

**Figure 66 YUV420 Spatial Sampling for MPEG 2 and MPEG 4**



1255

1256

**Figure 67 YUV420 8-bit Frame Format**

### 11.2.3 YUV420 10-bit

1258 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 68.

1262 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.

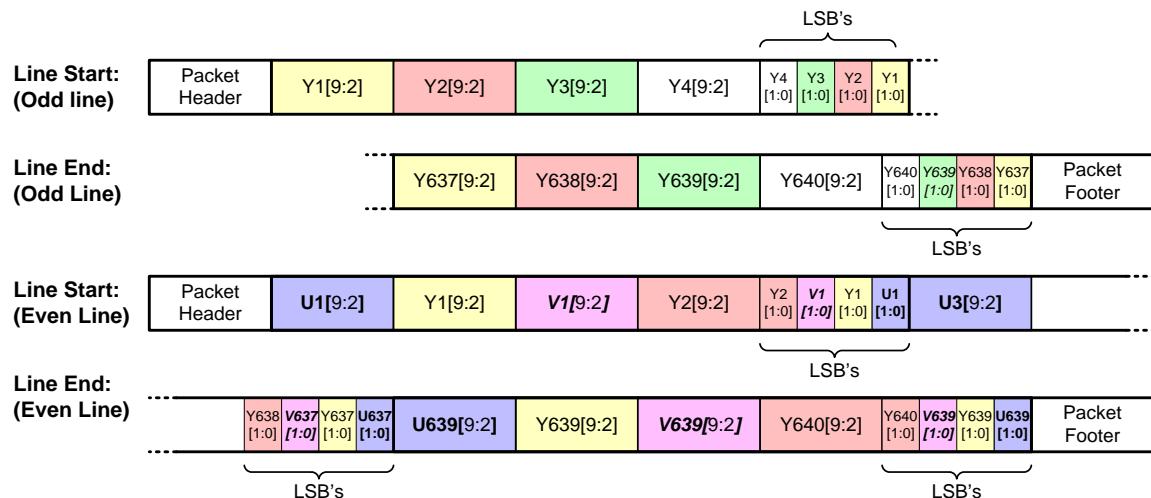
1264 Table 13 specifies the packet size constraints for YUV420 10-bit packets. The length of each packet must be a multiple of the values in the table.

1266

**Table 13 YUV420 10-bit Packet Data Size Constraints**

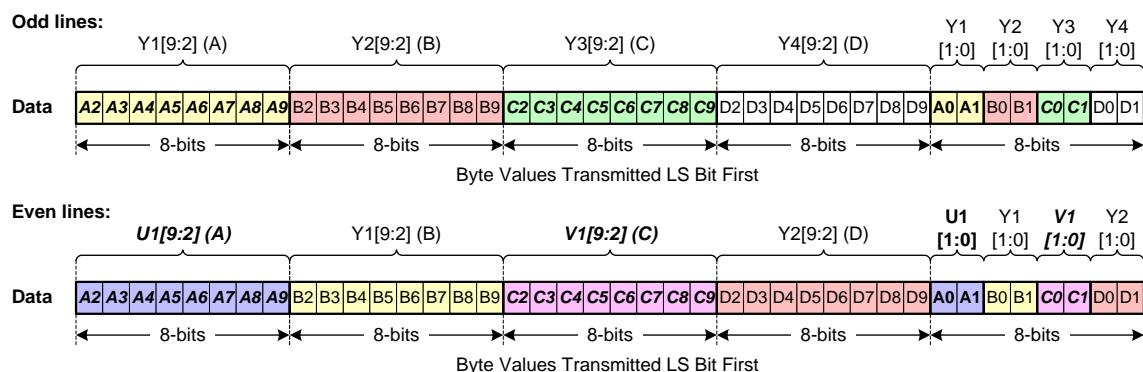
Odd Lines (1, 3, 5...) Luminance Only, Y			Even Lines (2, 4, 6...) Luminance and Chrominance, UYVY		
Pixels	Bytes	Bits	Pixels	Bytes	Bits
4	5	40	4	10	80

1267 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated  
1268 in Figure 69.



1269

**Figure 68 YUV420 10-bit Transmission**

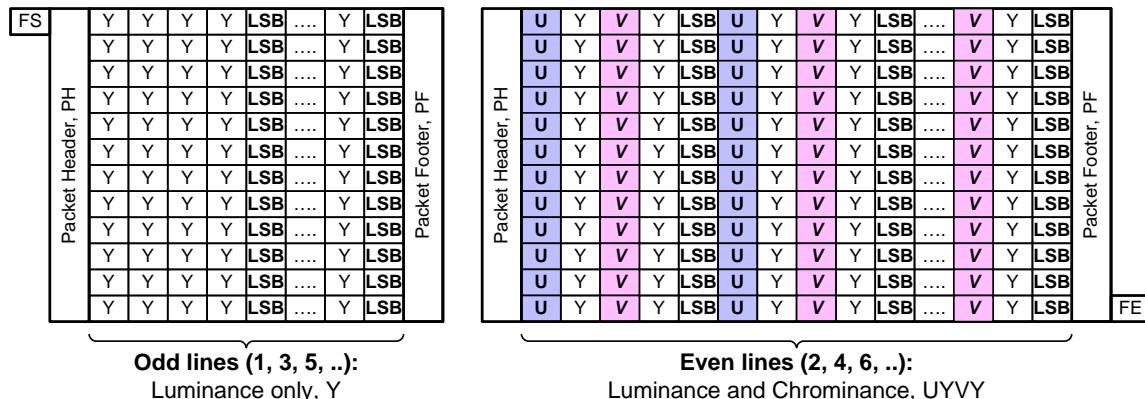


1271

**Figure 69 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration**

1272

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



1274

1275

**Figure 70 YUV420 10-bit Frame Format**

#### 11.2.4 YUV422 8-bit

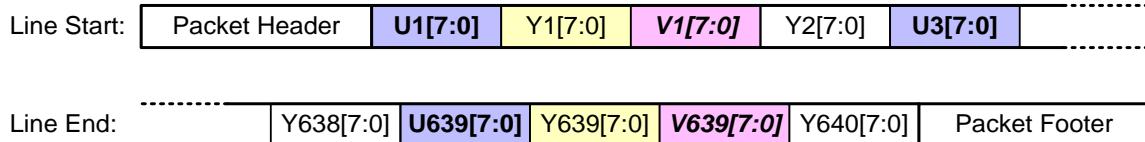
1277 YUV422 8-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in Figure 71.

1279 Table 14 specifies the packet size constraints for YUV422 8-bit packet. The length of each packet must be a  
1280 multiple of the values in the table.

**Table 14 YUV422 8-bit Packet Data Size Constraints**

Pixels	Bytes	Bits
2	4	32

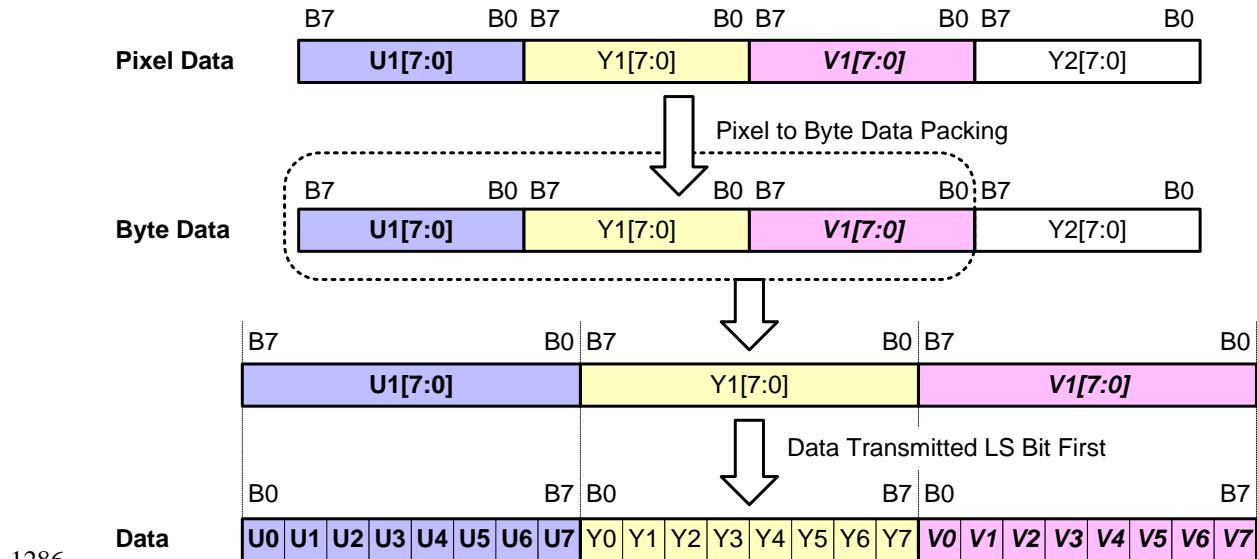
1282 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated  
1283 in Figure 72.



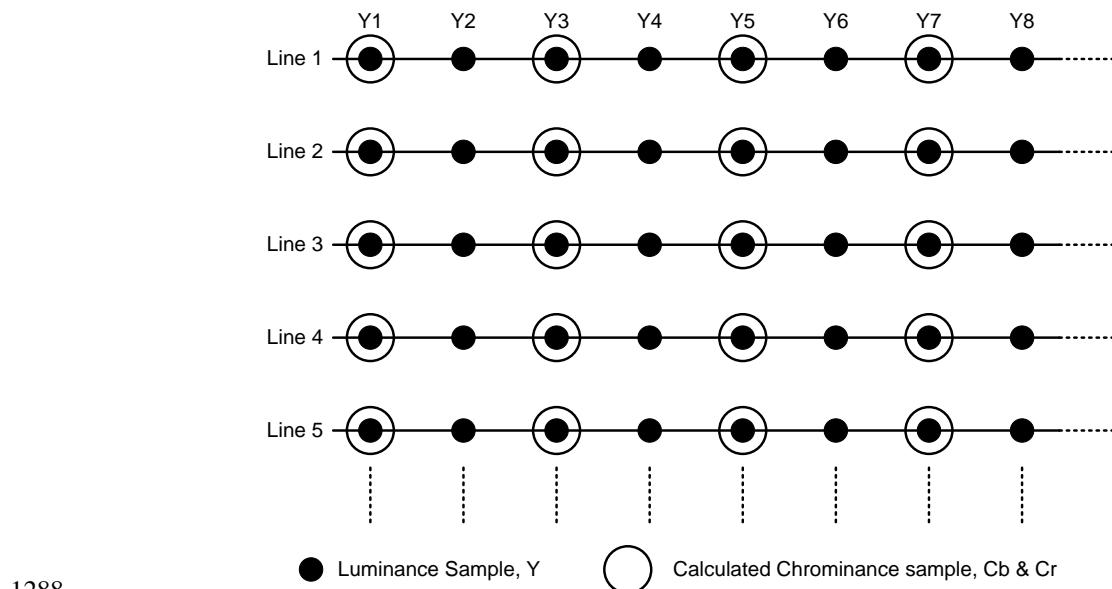
1284

1285

**Figure 71 YUV422 8-bit Transmission**



**Figure 72 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration**



**Figure 73 YUV422 Co-sited Spatial Sampling**

1290 The pixel spatial alignment is the same as in CCIR-656 standard. The frame format for YUV422 is presented  
1291 in Figure 74.

1292

1293

**Figure 74 YUV422 8-bit Frame Format**

### 1294 11.2.5 YUV422 10-bit

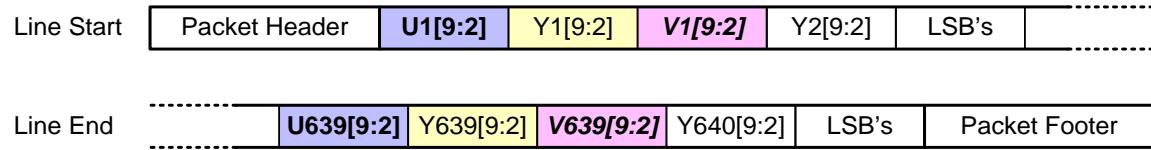
1295 YUV422 10-bit data transmission is performed by transmitting a UYVY sequence. This sequence is  
1296 illustrated in Figure 75.

1297 Table 15 specifies the packet size constraints for YUV422 10-bit packet. The length of each packet must be  
1298 a multiple of the values in the table.

1299 **Table 15 YUV422 10-bit Packet Data Size Constraints**

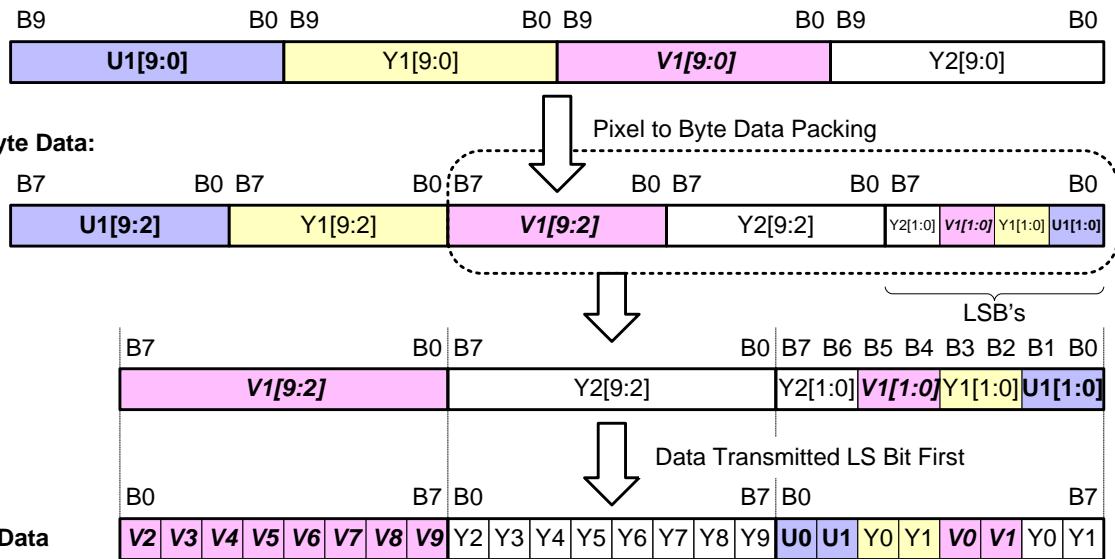
Pixels	Bytes	Bits
2	5	40

1300 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated  
1301 in Figure 76.



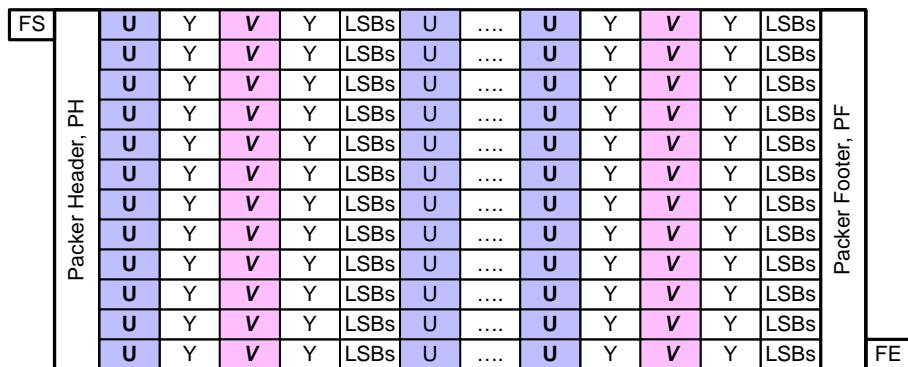
1302 1303 **Figure 75 YUV422 10-bit Transmitted Bytes**

**Pixel Data:**



**Figure 76 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 77.



**Figure 77 YUV422 10-bit Frame Format**

**11.3 RGB Image Data**

Table 16 defines the data type codes for RGB data formats described in this section.

1312

**Table 16 RGB Image Data Types**

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

1313

### 11.3.1 RGB888

1314  
1315

RGB888 data transmission is performed by transmitting a BGR byte sequence. This sequence is illustrated in Figure 78. The RGB888 frame format is illustrated in Figure 80.

1316  
1317

Table 17 specifies the packet size constraints for RGB888 packets. The length of each packet must be a multiple of the values in the table.

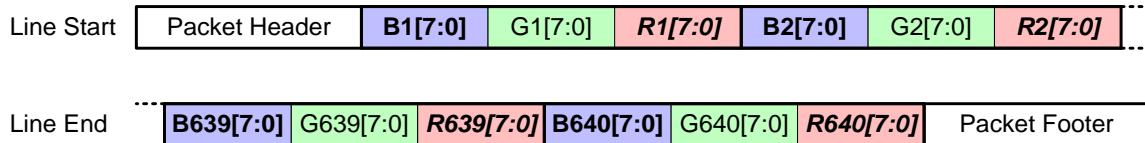
1318

**Table 17 RGB888 Packet Data Size Constraints**

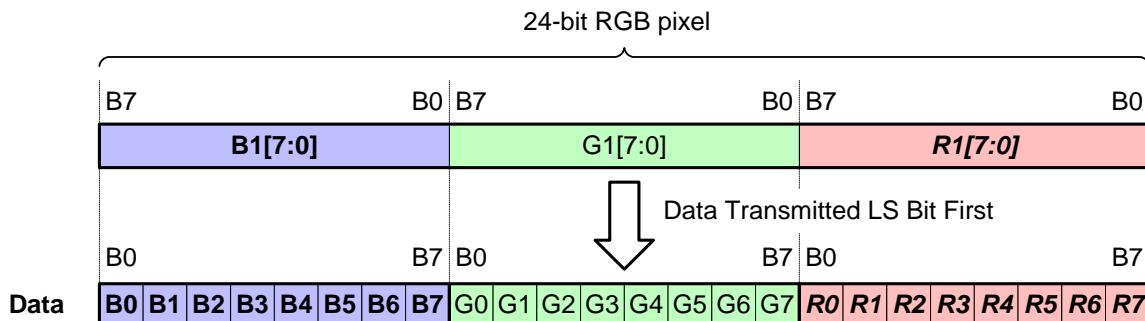
Pixels	Bytes	Bits
1	3	24

1319  
1320

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in Figure 79.



**Figure 78 RGB888 Transmission**



**Figure 79 RGB888 Transmission in CSI-2 Bus Bitwise Illustration**

1323  
1324

1325

1326

**Figure 80 RGB888 Frame Format**

### 1327 11.3.2 RGB666

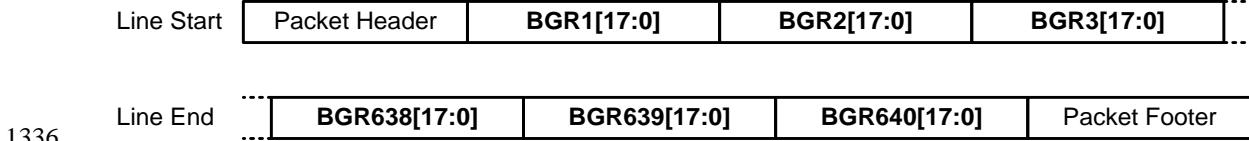
1328 RGB666 data transmission is performed by transmitting a B0...5, G0...5, and R0...5 (18-bit) sequence. This  
1329 sequence is illustrated in Figure 81. The frame format for RGB666 is presented in the Figure 83.

1330 Table 18 specifies the packet size constraints for RGB666 packets. The length of each packet must be a  
1331 multiple of the values in the table.

**Table 18 RGB666 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	9	72

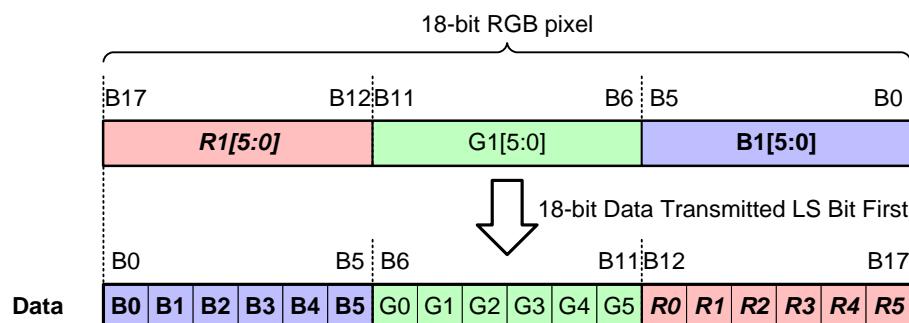
1332 Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB666 case the length of one data  
1333 word is 18-bits, not eight bits. The word-wise flip is done for 18-bit BGR words; i.e. instead of flipping each  
1334 byte (8-bits), each 18-bits pixel value is flipped. This is illustrated in Figure 82.  
1335



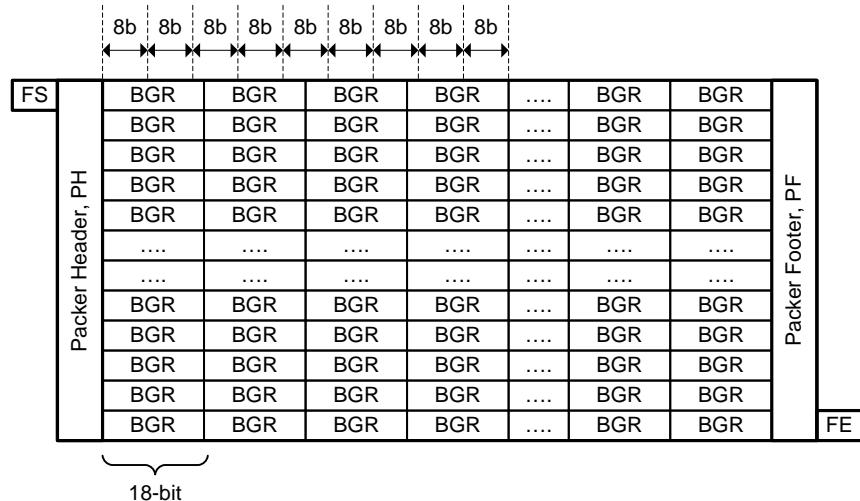
**Figure 81 RGB666 Transmission with 18-bit BGR Words**

1338

1339



**Figure 82 RGB666 Transmission on CSI-2 Bus Bitwise Illustration**



1340

1341

**Figure 83 RGB666 Frame Format**

### 1342 11.3.3 RGB565

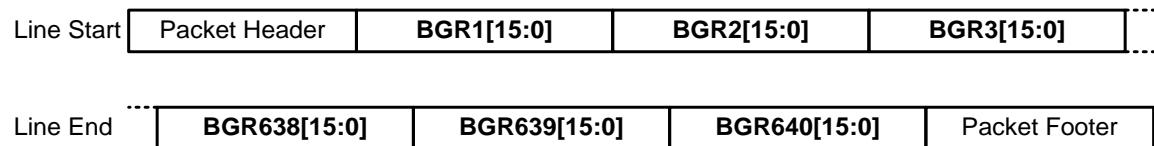
1343 RGB565 data transmission is performed by transmitting B0...B4, G0...G5, R0...R4 in a 16-bit sequence.  
1344 This sequence is illustrated in Figure 84. The frame format for RGB565 is presented in the Figure 86.

1345 Table 19 specifies the packet size constraints for RGB565 packets. The length of each packet must be a  
1346 multiple of the values in the table.

1347 **Table 19 RGB565 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	2	16

1348 Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB565 case the length of one data  
1349 word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words; i.e. instead of flipping each  
1350 byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 85.



1351

1352

**Figure 84 RGB565 Transmission with 16-bit BGR Words**

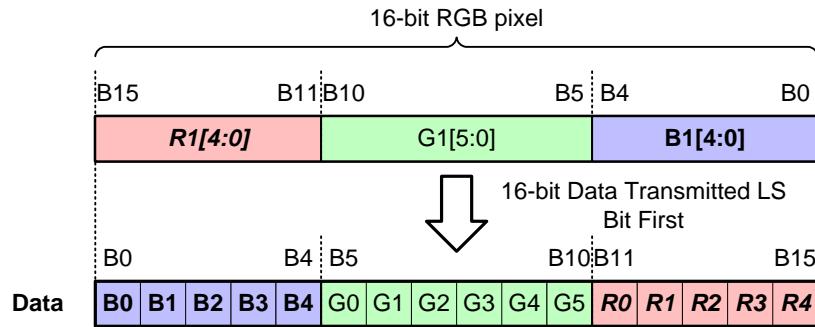


Figure 85 RGB565 Transmission on CSI-2 Bus Bitwise Illustration

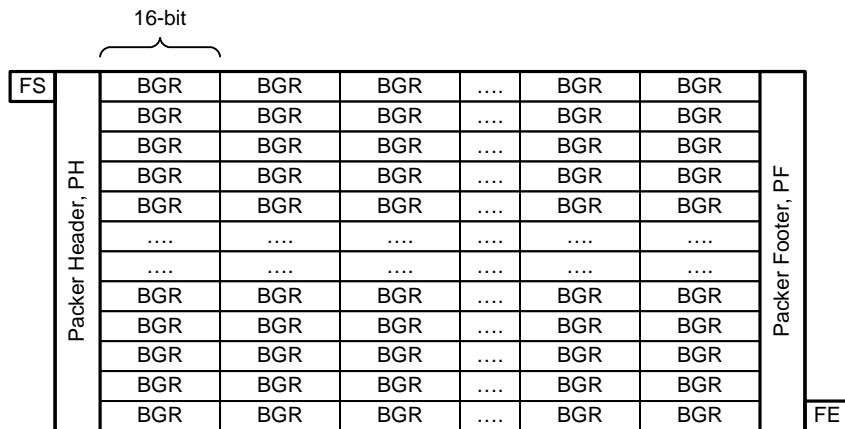


Figure 86 RGB565 Frame Format

#### 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 87.

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 87.

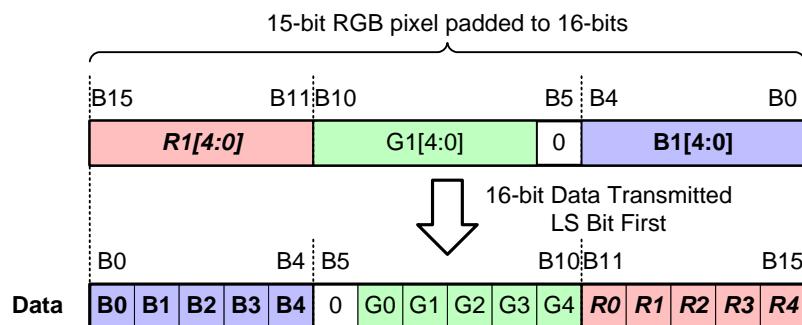


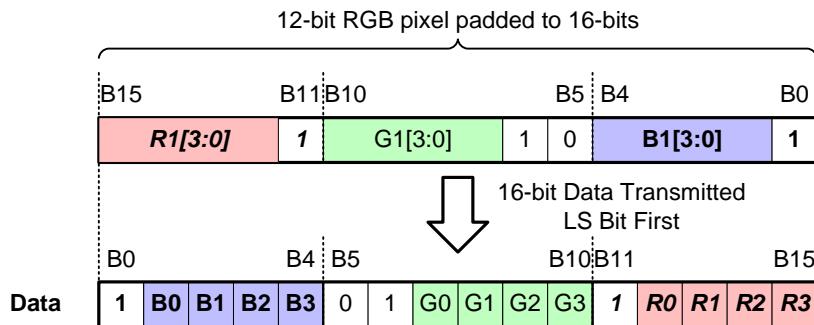
Figure 87 RGB555 Transmission on CSI-2 Bus Bitwise Illustration

1367 **11.3.5 RGB444**

1368 RGB444 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB444 data should  
1369 be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs of each  
1370 color component as illustrated in Figure 88.

1371 Both the frame format and the package size constraints are the same as the RGB565 case.

1372 Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB444 case the length of one data  
1373 word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words; i.e. instead of flipping each  
1374 byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 88.



1375 **Figure 88 RGB444 Transmission on CSI-2 Bus Bitwise Illustration**

1377 **11.4 RAW Image Data**

1378 The RAW 6/7/8/10/12/14 modes are used for transmitting Raw image data from the image sensor.

1379 The intent is that Raw image data is unprocessed image data (i.e. Raw Bayer data) or complementary color  
1380 data, but RAW image data is not limited to these data types.

1381 It is possible to transmit e.g. light shielded pixels in addition to effective pixels. This leads to a situation  
1382 where the line length is longer than sum of effective pixels per line. The line length, if not specified otherwise,  
1383 has to be a multiple of word (32 bits).

1384 Table 20 defines the data type codes for RAW data formats described in this section.

1385

**Table 20 RAW Image Data Types**

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

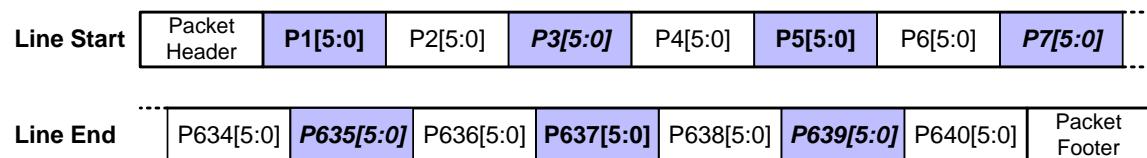
1386 **11.4.1 RAW6**

1387 The 6-bit Raw data transmission is done by transmitting the pixel data over CSI-2 bus. Each line is separated  
1388 by line start / end synchronization codes. This sequence is illustrated in Figure 89 (VGA case). Table 21  
1389 specifies the packet size constraints for RAW6 packets. The length of each packet must be a multiple of the  
1390 values in the table.

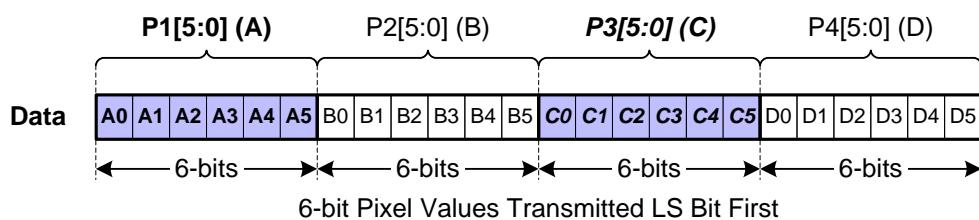
1391 **Table 21 RAW6 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	3	24

1392 Each 6-bit pixel is sent LSB first. This is an exception to general CSI-2 rule byte wise LSB first.



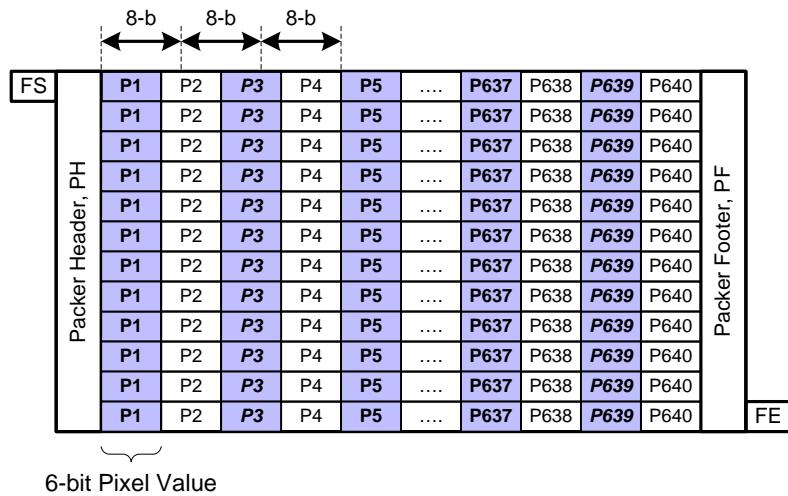
**Figure 89 RAW6 Transmission**



1395            6-bit Pixel Values Transmitted LS Bit First

1396

**Figure 90 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration**



1397

1398

**Figure 91 RAW6 Frame Format**

#### 1399 11.4.2 RAW7

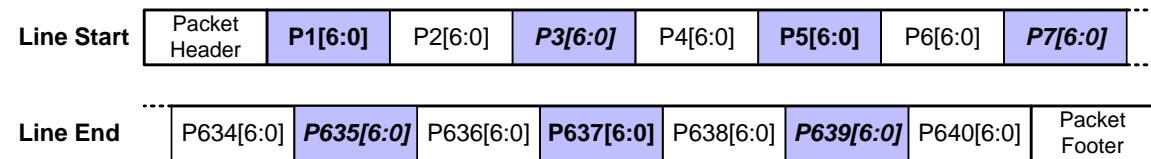
1400 The 7-bit Raw data transmission is done by transmitting the pixel data over CSI-2 bus. Each line is separated  
 1401 by line start / end synchronization codes. This sequence is illustrated in Figure 92 (VGA case). Table 22  
 1402 specifies the packet size constraints for RAW7 packets. The length of each packet must be a multiple of the  
 1403 values in the table.

1404

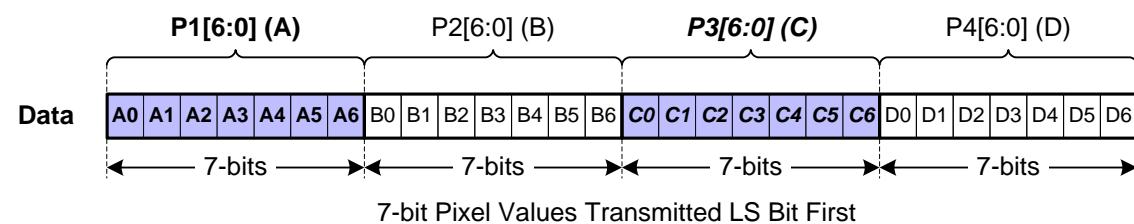
**Table 22 RAW7 Packet Data Size Constraints**

Pixels	Bytes	Bits
8	7	56

1405 Each 7-bit pixel is sent LSB first. This is an exception to general CSI-2 rule byte-wise LSB first.



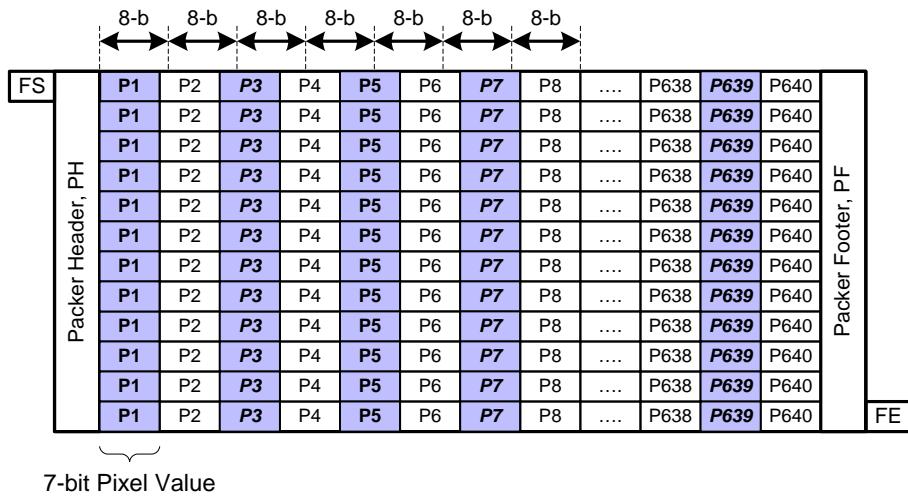
**Figure 92 RAW7 Transmission**



1408

1409

**Figure 93 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration**



1410

1411

**Figure 94 RAW7 Frame Format**

### 11.4.3 RAW8

1413 The 8-bit Raw data transmission is done by transmitting the pixel data over a CSI-2 bus. Table 23 specifies  
1414 the packet size constraints for RAW8 packets. The length of each packet must be a multiple of the values in  
1415 the table.

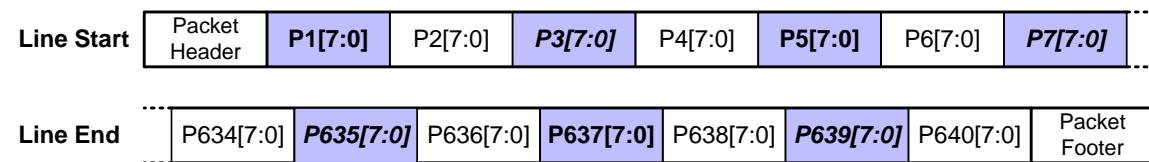
1416

**Table 23 RAW8 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	1	8

1417 This sequence is illustrated in Figure 95 (VGA case).

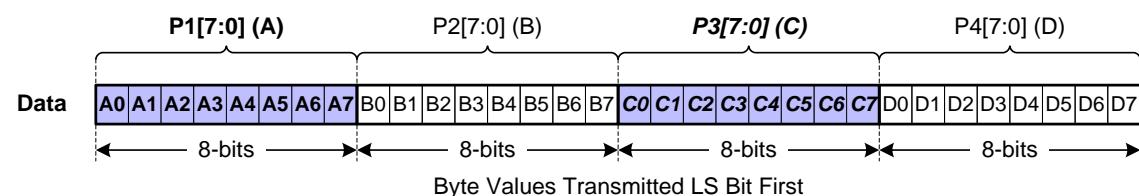
1418 Bit order in transmission follows the general CSI-2 rule, LSB first.



1419

1420

**Figure 95 RAW8 Transmission**



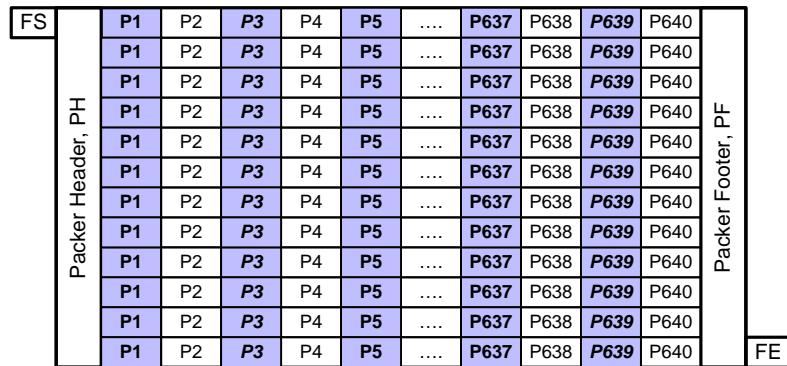
1421

1422

**Figure 96 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration**

1423

1424



**Figure 97 RAW8 Frame Format**

#### 1425 11.4.4 RAW10

1426 The transmission of 10-bit Raw data is done by packing the 10-bit pixel data to look like 8-bit data format.  
1427 Table 24 specifies the packet size constraints for RAW10 packets. The length of each packet must be a  
1428 multiple of the values in the table.

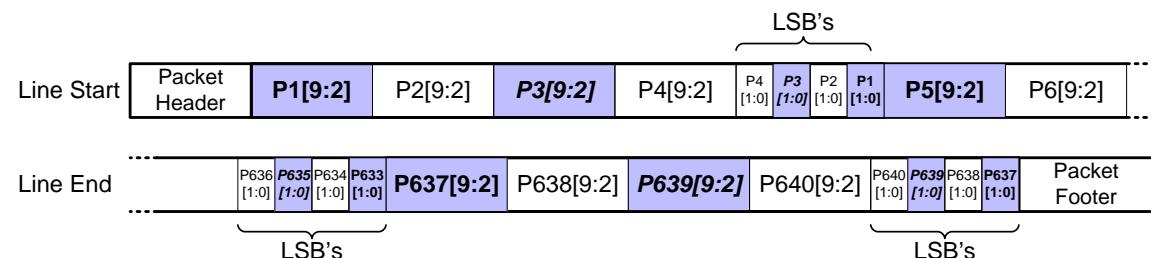
1429

**Table 24 RAW10 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	5	40

1430 This sequence is illustrated in Figure 98 (VGA case).

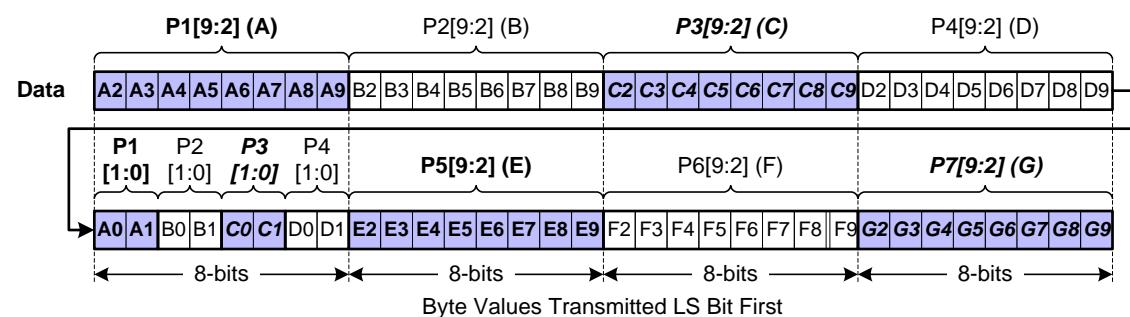
1431 Bit order in transmission follows the general CSI-2 rule, LSB first.



1432

1433

**Figure 98 RAW10 Transmission**



1434

1435

**Figure 99 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration**

1436

1437

#### 1438 11.4.5 RAW12

1439 The transmission of 12-bit Raw data is done by packing the 12-bit pixel data to look like 8-bit data format.  
1440 Table 25 specifies the packet size constraints for RAW12 packets. The length of each packet must be a  
1441 multiple of the values in the table.

1442

**Table 25 RAW12 Packet Data Size Constraints**

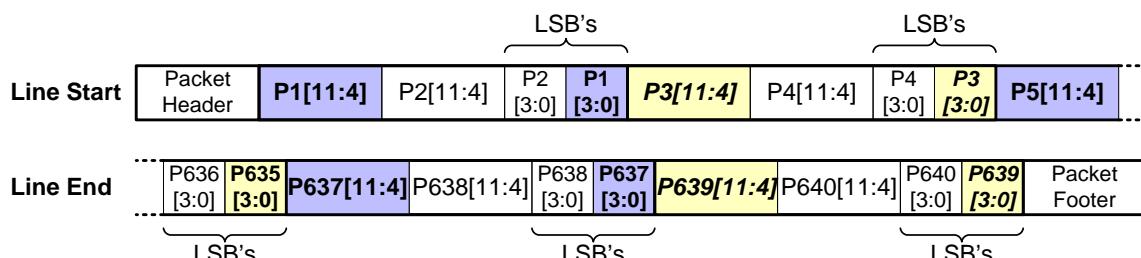
Pixels	Bytes	Bits
2	3	24

1443 This sequence is illustrated in Figure 101 (VGA case).

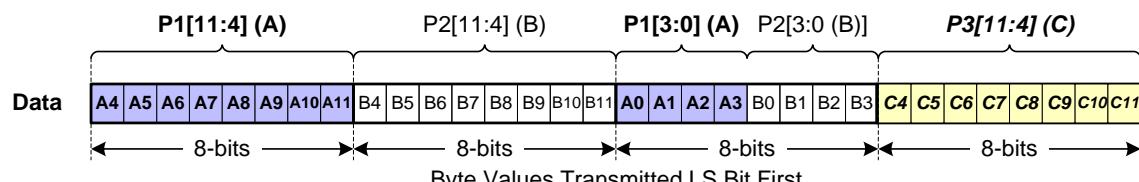
1444 Bit order in transmission follows the general CSI-2 rule, LSB first.

1445

1446



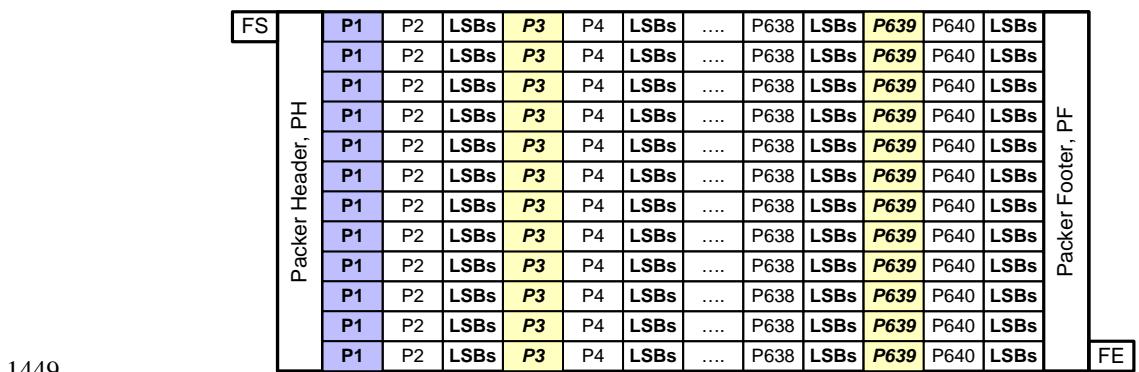
**Figure 101 RAW12 Transmission**



1447

1448

**Figure 102 RAW12 Transmission on CSI-2 Bus Bitwise Illustration**



**Figure 103 RAW12 Frame Format**

#### 11.4.6 RAW14

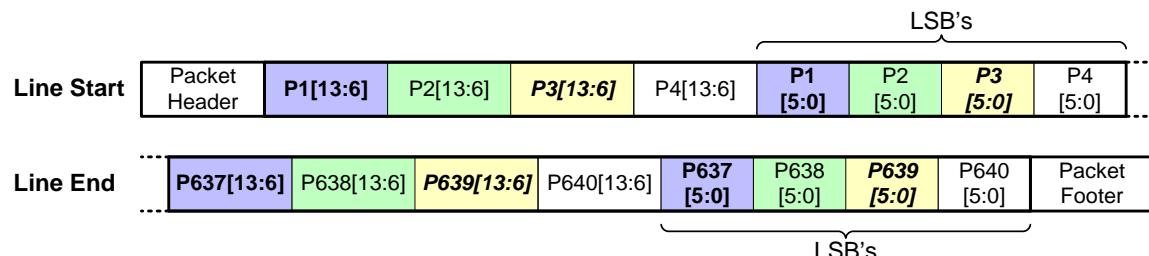
The transmission of 14-bit Raw data is done by packing the 14-bit pixel data in 8-bit slices. For every four pixels, seven bytes of data is generated. Table 26 specifies the packet size constraints for RAW14 packets. The length of each packet must be a multiple of the values in the table.

**Table 26 RAW14 Packet Data Size Constraints**

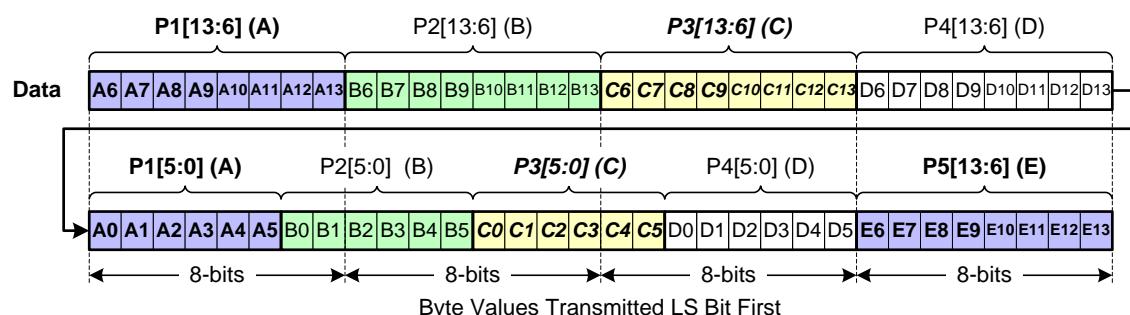
Pixels	Bytes	Bits
4	7	56

The sequence is illustrated in Figure 104 (VGA case).

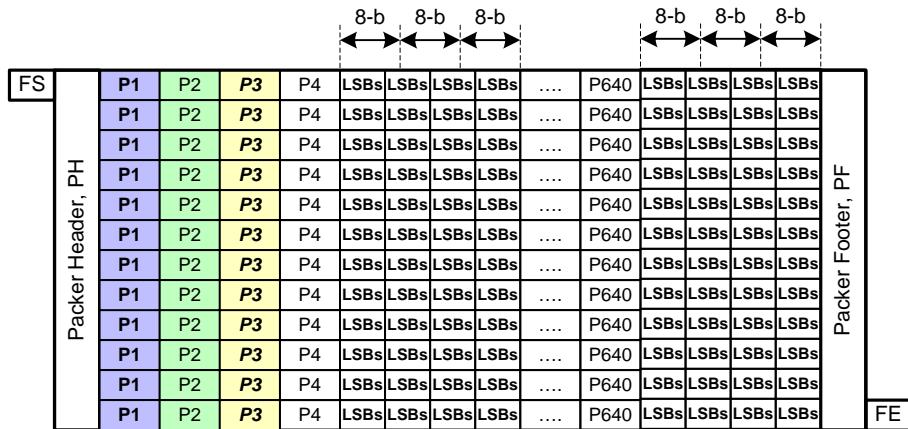
The LS bits for P1, P2, P3 and P4 are distributed in three bytes as shown in Figure 105. The same is true for the LS bits for P637, P638, P639 and P640. The bit order during transmission follows the general CSI-2 rule, i.e. LSB first.



**Figure 104 RAW14 Transmission**



**Figure 105 RAW14 Transmission on CSI-2 Bus Bitwise Illustration**



1464

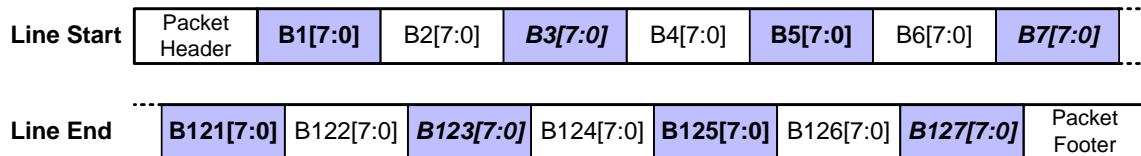
1465

**Figure 106 RAW14 Frame Format**

## 1466 11.5 User Defined Data Formats

1467 The User Defined Data Type values shall be used to transmit arbitrary data, such as JPEG and MPEG4 data,  
1468 over the CSI-2 bus. Data shall be packed so that the data length is divisible by eight bits. If data padding is  
1469 required, the padding shall be added before data is presented to the CSI-2 protocol interface.

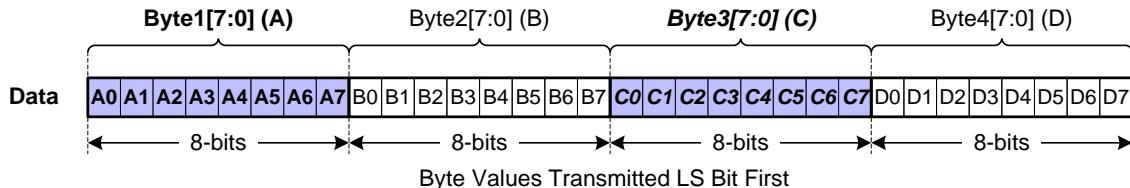
1470 Bit order in transmission follows the general CSI-2 rule, LSB first.



1471

1472

**Figure 107 User Defined 8-bit Data (128 Byte Packet)**



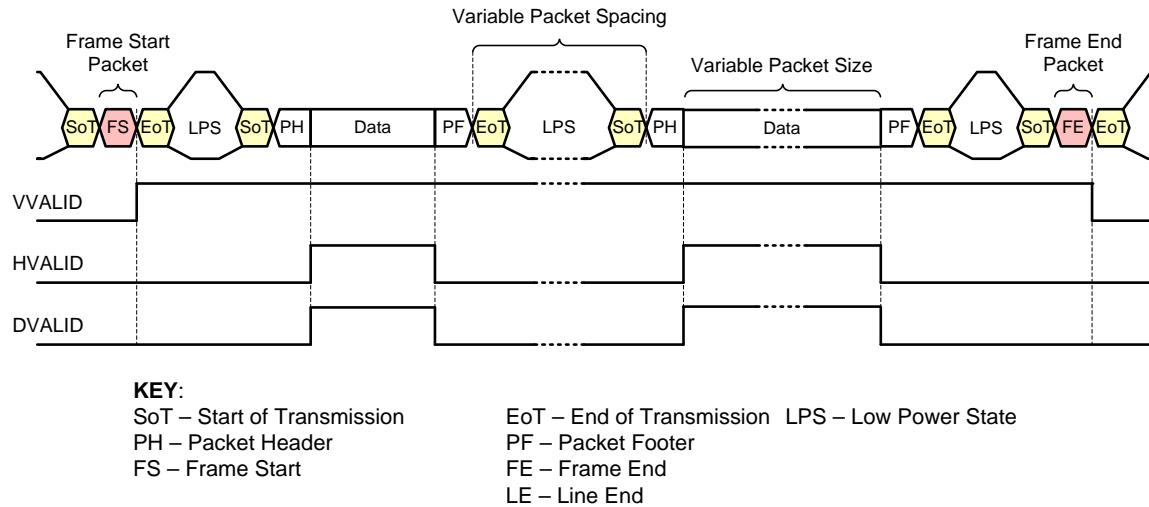
1473

1474 **Figure 108 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration**

1475 The packet data size in bits shall be divisible by eight, i.e. a whole number of bytes shall be transmitted.

1476 For User Defined data:

- 1477 • The frame is transmitted as a sequence of arbitrary sized packets.
- 1478 • The packet size may vary from packet to packet.
- 1479 • The packet spacing may vary between packets.



1480

1481

**Figure 109 Transmission of User Defined 8-bit Data**

1482 Eight different User Defined data type codes are available as shown in Table 27.

**Table 27 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

## 1484 12 Recommended Memory Storage

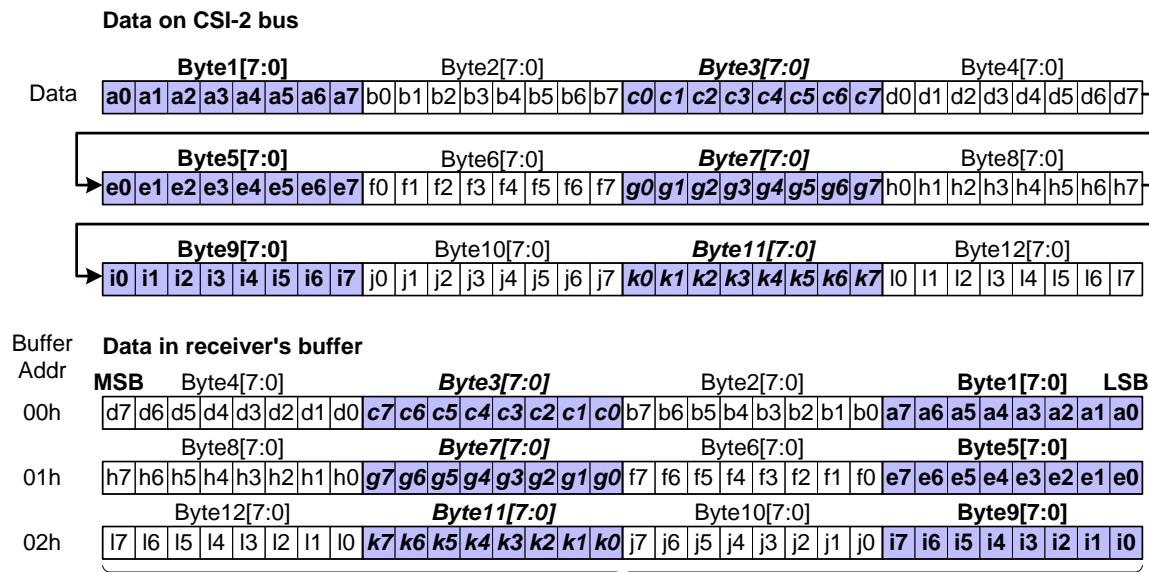
1485 This section is informative.

1486 The CSI-2 data protocol requires certain behavior from the receiver connected to the CSI transmitter. The  
1487 following sections describe how different data formats should be stored inside the receiver. While  
1488 informative, this section is provided to ease application software development by suggesting a common data  
1489 storage format among different receivers.

### 1490 12.1 General/Arbitrary Data Reception

1491 In the generic case and for arbitrary data the first byte of payload data transmitted maps the LS byte of the  
1492 32-bit memory word and the fourth byte of payload data transmitted maps to the MS byte of the 32-bit  
1493 memory word.

1494 Figure 110 shows the generic CSI-2 byte to 32-bit memory word mapping rule.

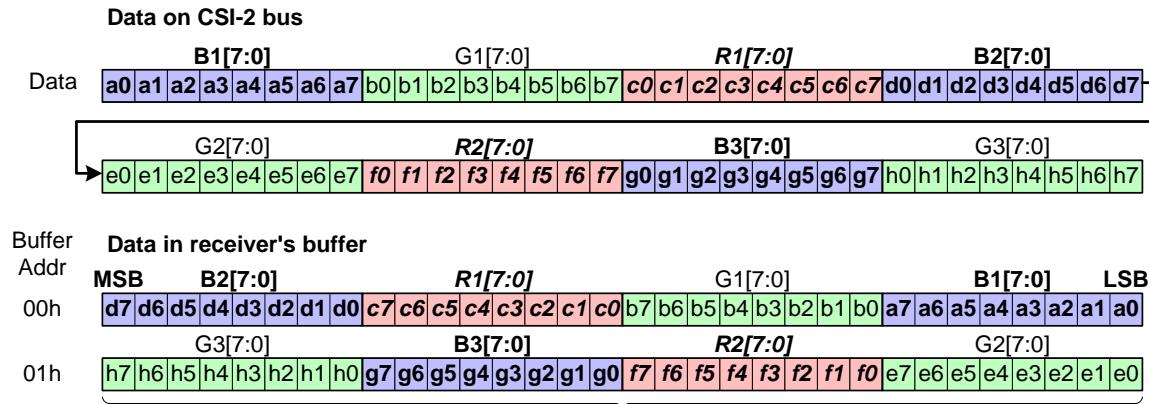


1495  
1496

1497 **Figure 110 General/Arbitrary Data Reception**

## 1498 12.2 RGB888 Data Reception

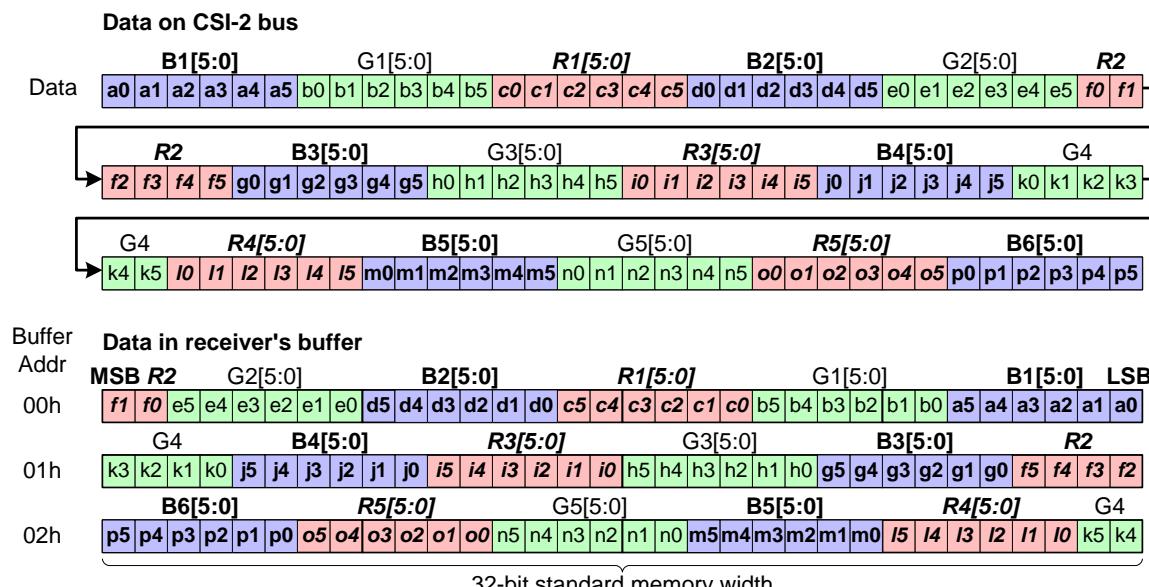
1499 The RGB888 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.



1500

Figure 111 RGB888 Data Format Reception

## 1501 12.3 RGB666 Data Reception

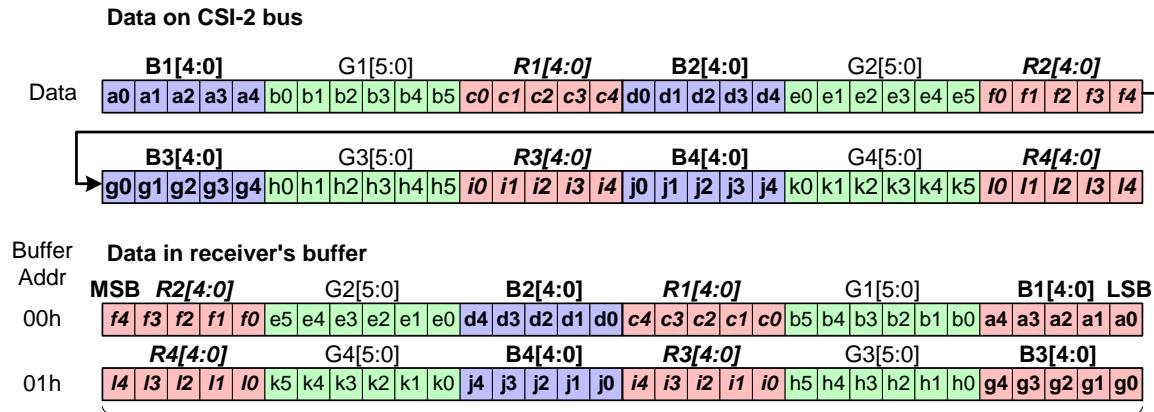


1503

Figure 112 RGB666 Data Format Reception

1505

## 12.4 RGB565 Data Reception

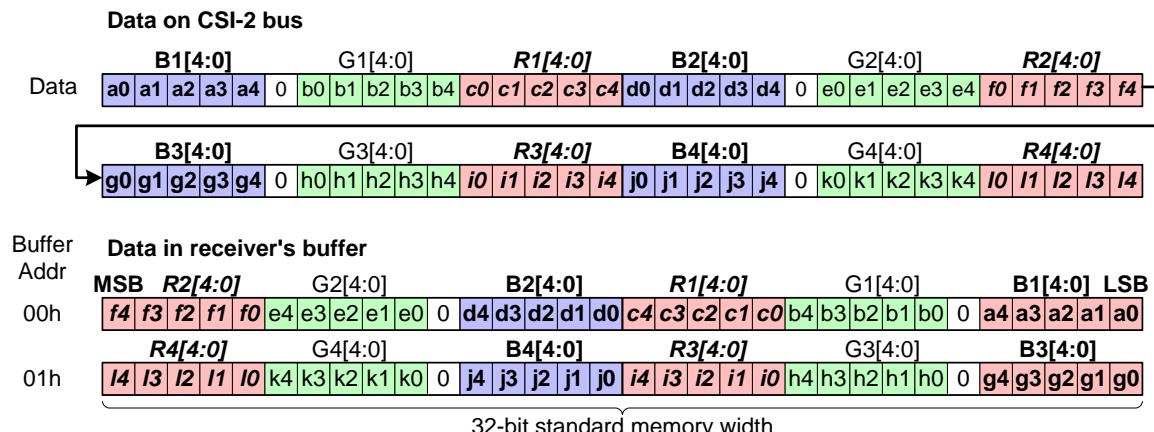


1506

Figure 113 RGB565 Data Format Reception

1508

## 12.5 RGB555 Data Reception

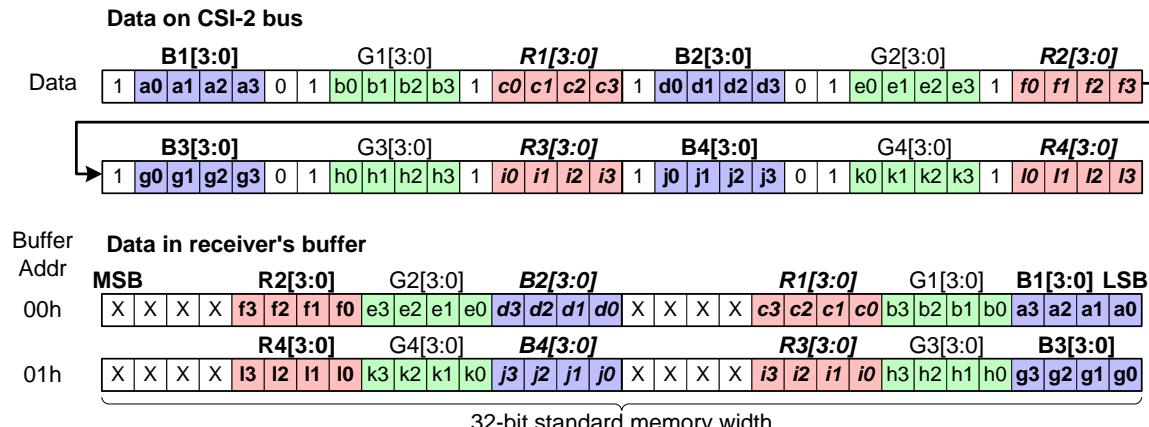


1509

Figure 114 RGB555 Data Format Reception

## 1511 12.6 RGB444 Data Reception

1512 The RGB444 data format byte to 32-bit memory word mapping has a special transform as shown in Figure  
1513 115.



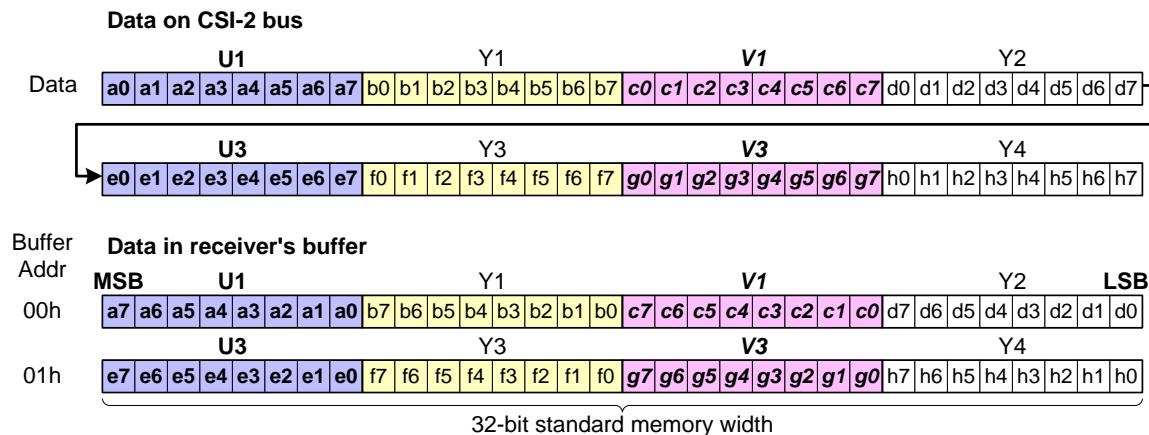
1514

Figure 115 RGB444 Data Format Reception

## 1516 12.7 YUV422 8-bit Data Reception

1517 The YUV422 8-bit data format the byte to 32-bit memory word mapping does not follow the generic CSI-2  
1518 rule.

1519 For YUV422 8-bit data format the first byte of payload data transmitted maps the MS byte of the 32-bit  
1520 memory word and the fourth byte of payload data transmitted maps to the LS byte of the 32-bit memory  
1521 word.

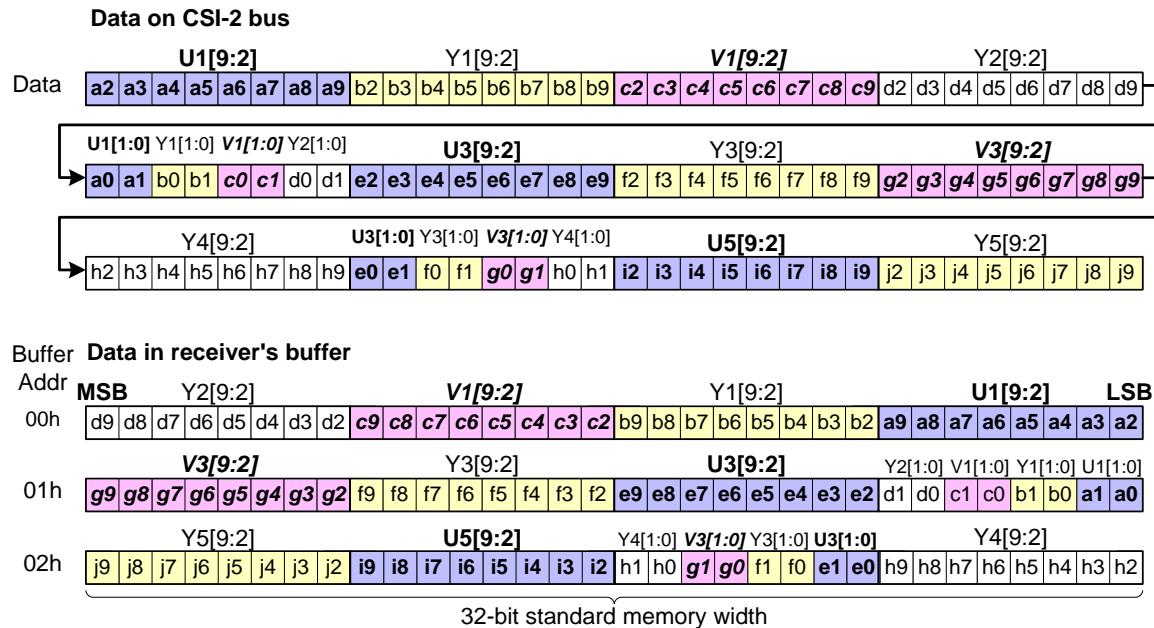


1522

Figure 116 YUV422 8-bit Data Format Reception

## 1524 12.8 YUV422 10-bit Data Reception

1525 The YUV422 10-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.



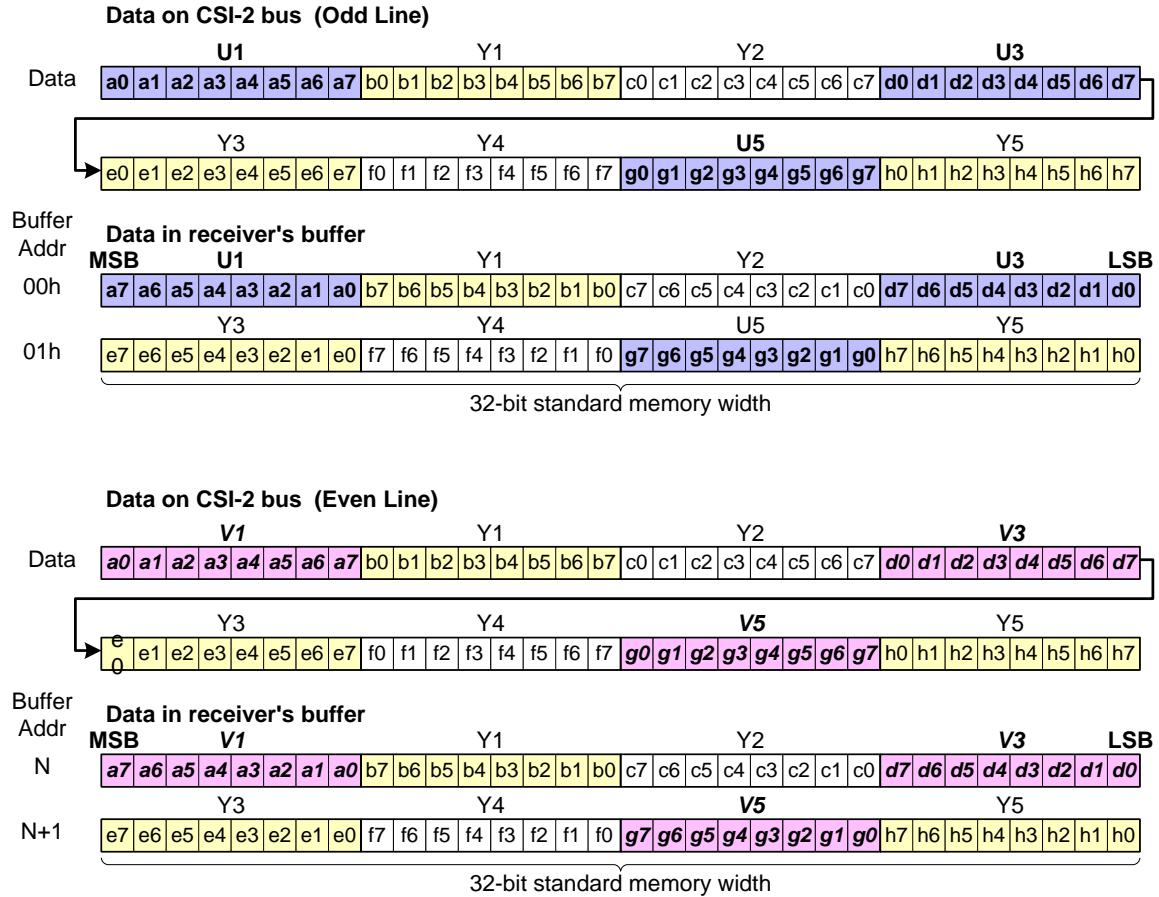
1526

**Figure 117 YUV422 10-bit Data Format Reception**

## 1528 12.9 YUV420 8-bit (Legacy) Data Reception

1529 The YUV420 8-bit (legacy) data format the byte to 32-bit memory word mapping does not follow the generic  
1530 CSI-2 rule.

1531 For YUV422 8-bit (legacy) data format the first byte of payload data transmitted maps the MS byte of the  
1532 32-bit memory word and the fourth byte of payload data transmitted maps to the LS byte of the 32-bit memory  
1533 word.

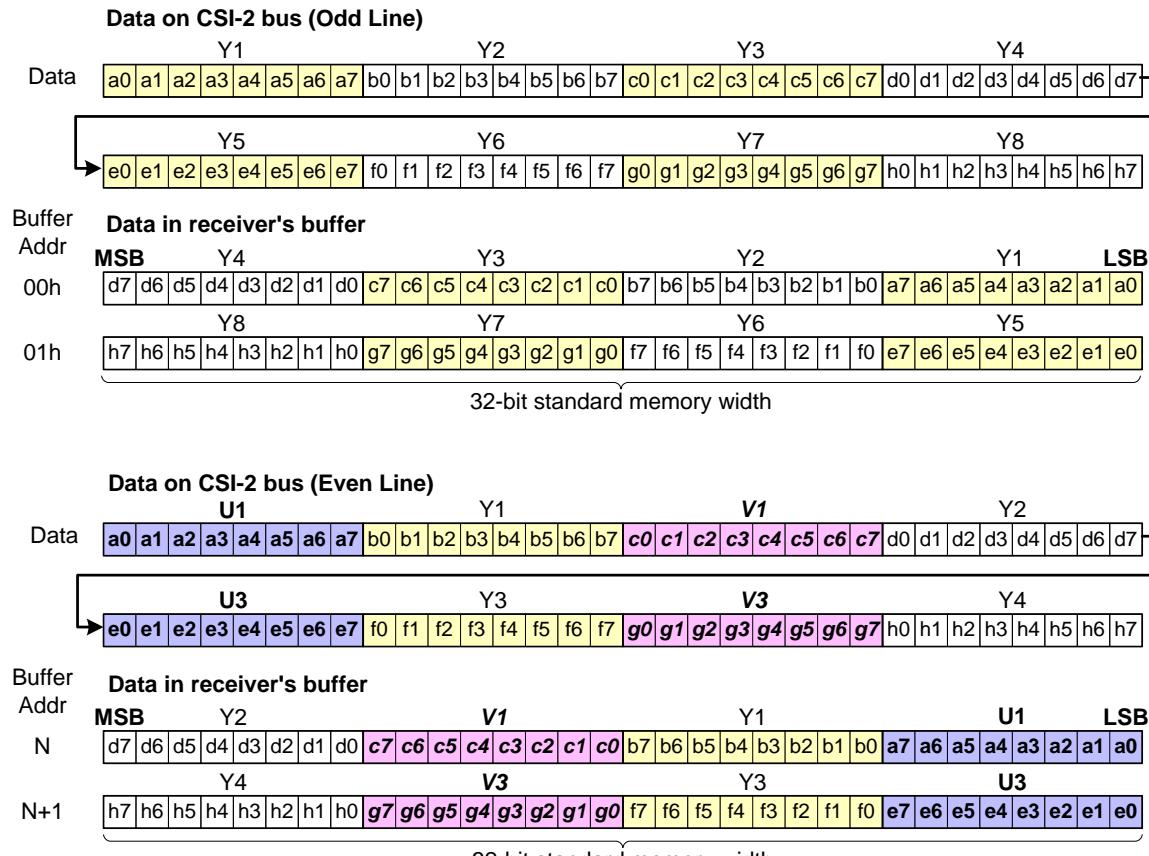


1534

**Figure 118 YUV420 8-bit Legacy Data Format Reception**

## 1536 12.10 YUV420 8-bit Data Reception

1537 The YUV420 8-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.



1538

**Figure 119 YUV420 8-bit Data Format Reception**

## 1540 12.11 YUV420 10-bit Data Reception

1541 The YUV420 10-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

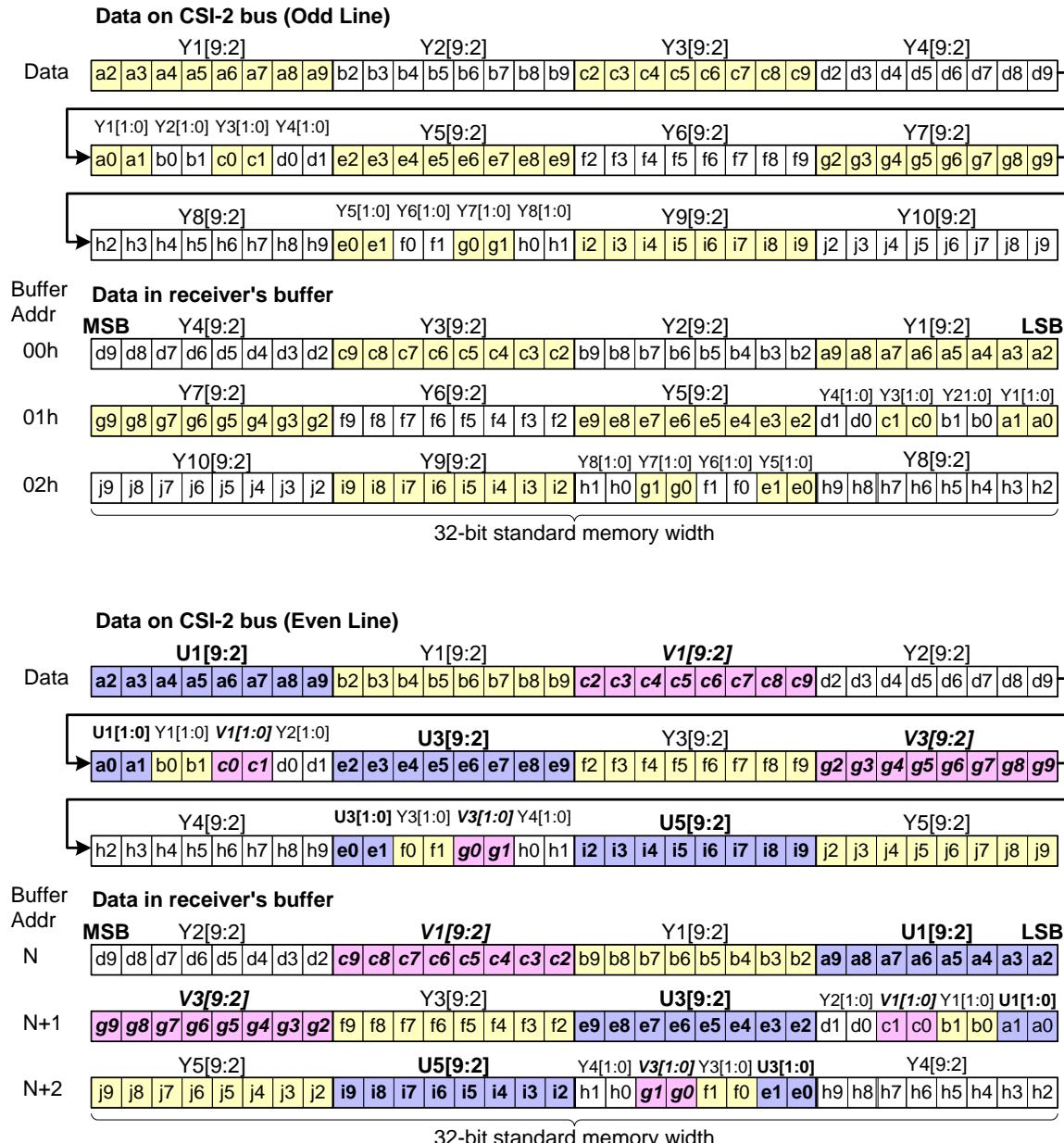
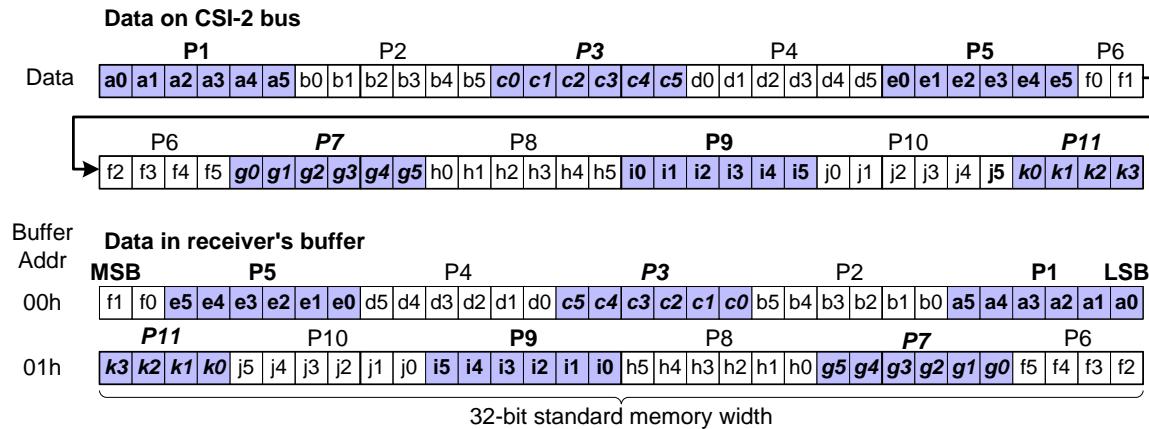


Figure 120 YUV420 10-bit Data Format Reception

1544

## 12.12 RAW6 Data Reception

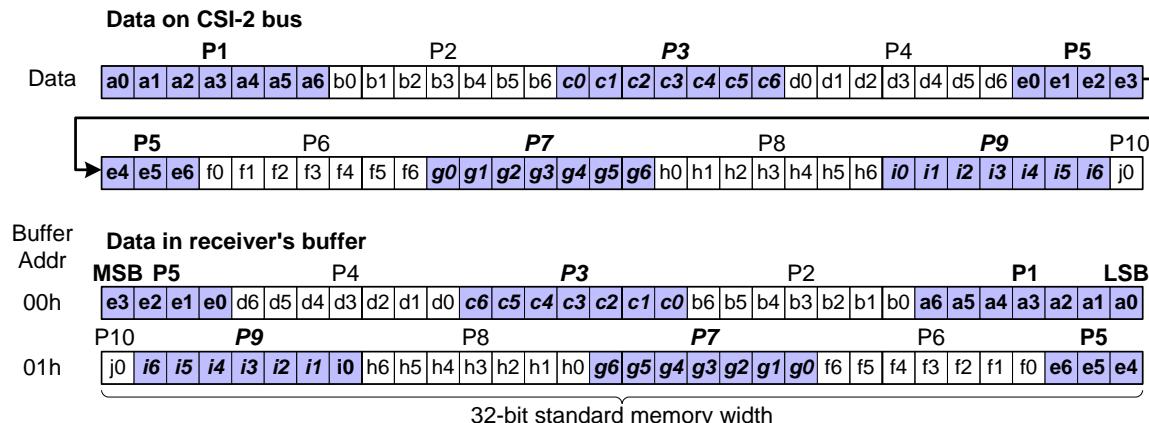


1545  
1546

Figure 121 RAW6 Data Format Reception

1547

## 12.13 RAW7 Data Reception

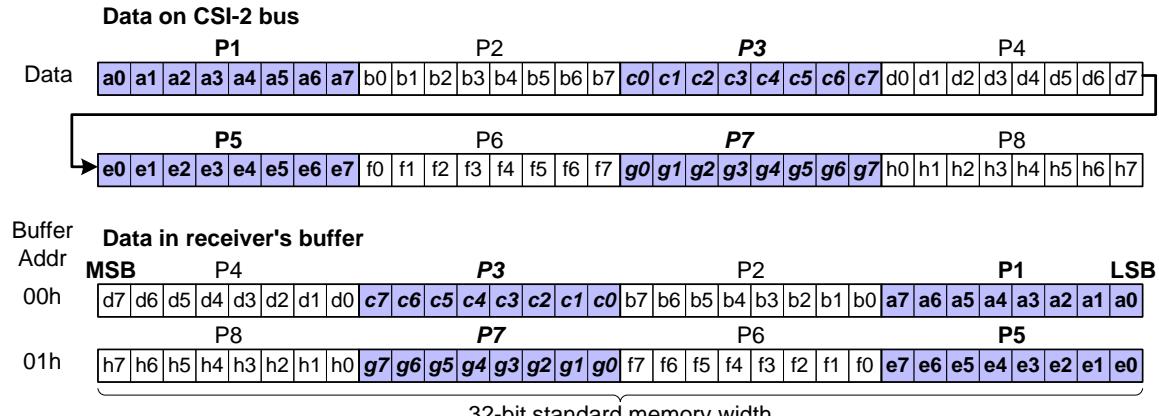


1549  
1550

Figure 122 RAW7 Data Format Reception

## 1551 12.14 RAW8 Data Reception

1552 The RAW8 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

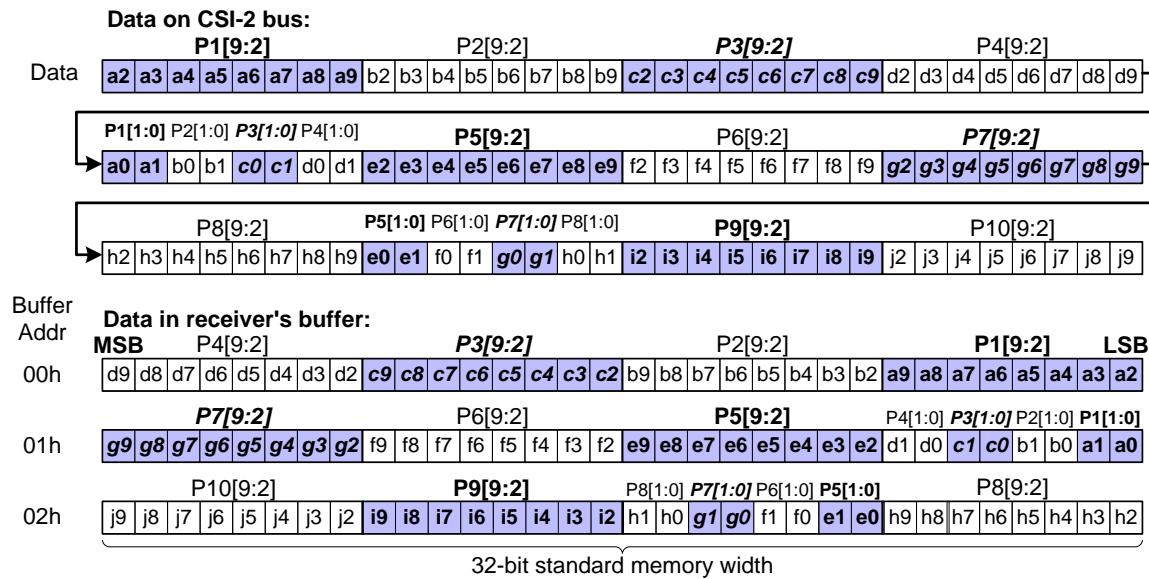


1553

**Figure 123 RAW8 Data Format Reception**

## 1555 12.15 RAW10 Data Reception

1556 The RAW10 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

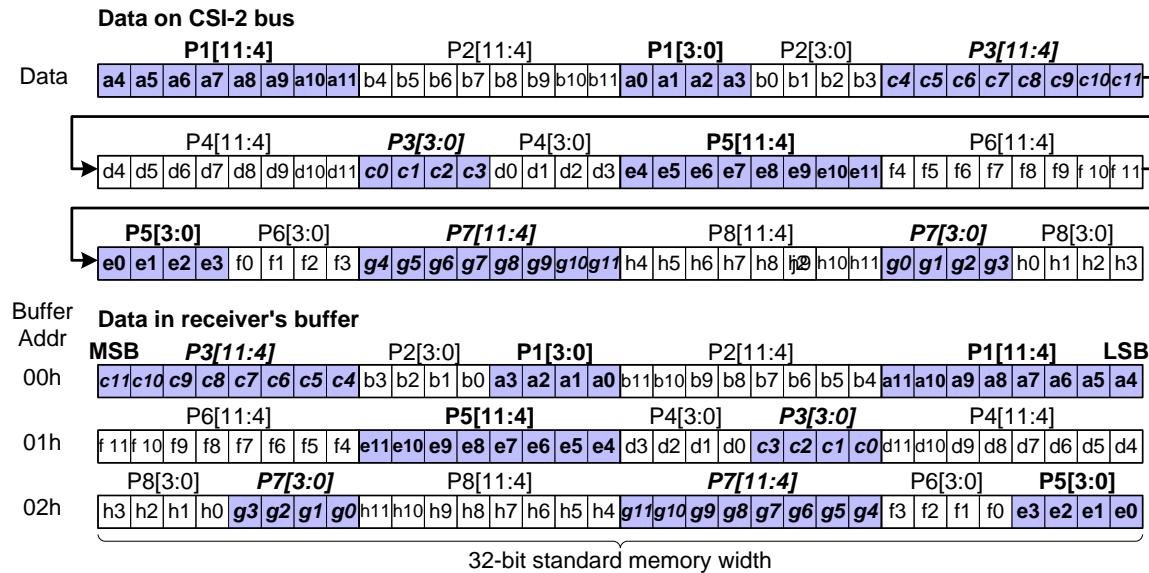


1557

**Figure 124 RAW10 Data Format Reception**

## 1559 12.16 RAW12 Data Reception

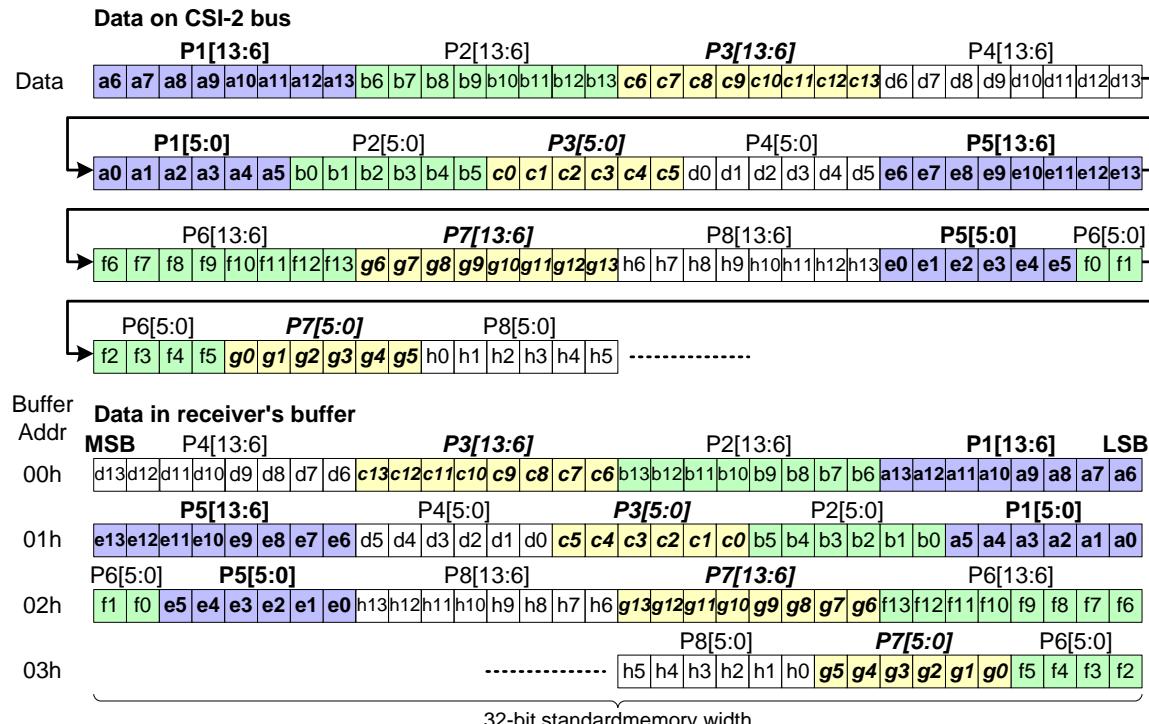
1560 The RAW12 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.



1561

Figure 125 RAW12 Data Format Reception

## 1563 12.17 RAW14 Data Reception



1564

Figure 126 RAW 14 Data Format Reception

1566 **Annex A JPEG8 Data Format (informative)**

1567 **A.1 Introduction**

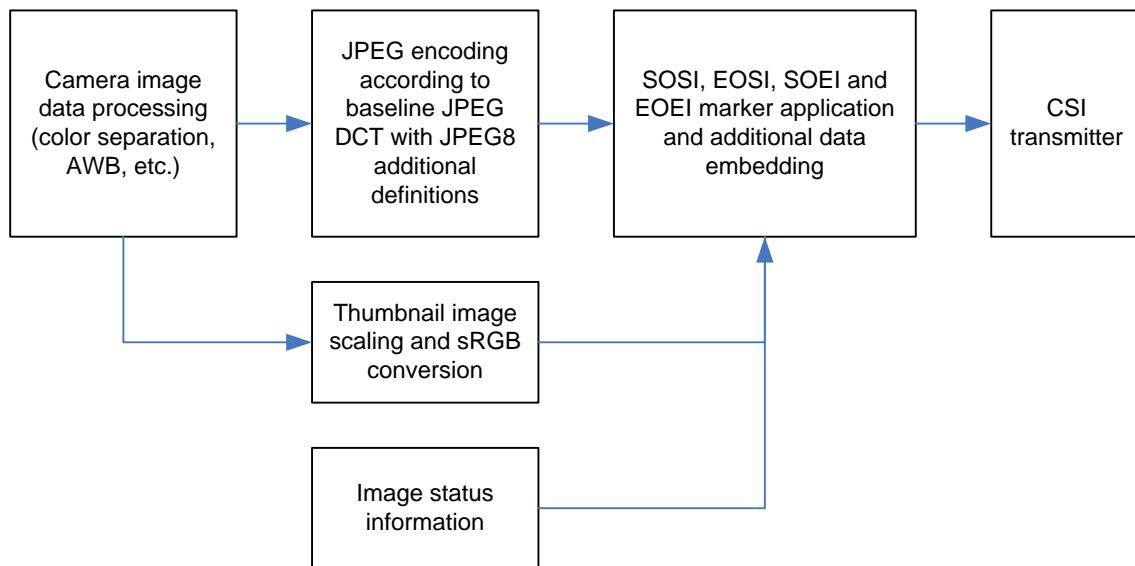
1568 This Annex contains an informative example of the transmission of compressed image data format using the  
1569 arbitrary Data Type values.

1570 JPEG8 has two non-standard extensions:

- 1571
  - Status information (mandatory)
  - Embedded Image information e.g. a thumbnail image (optional)

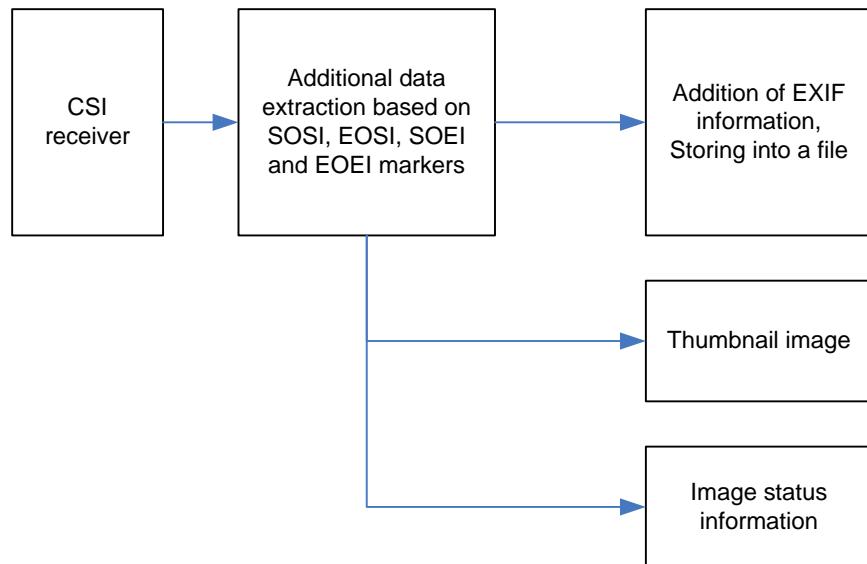
1573 Any non-standard or additional data inside the baseline JPEG data structure has to be removed from JPEG8  
1574 data before it is compliant with e.g. standard JPEG image viewers in e.g. a personal computer.

1575 The JPEG8 data flow is illustrated in the Figure 127 and Figure 128.



1576  
1577

**Figure 127 JPEG8 Data Flow in the Encoder**



1578

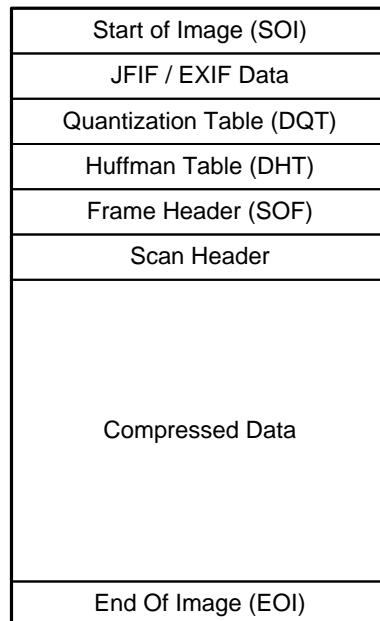
1579

**Figure 128 JPEG8 Data Flow in the Decoder**

1580 **A.2 JPEG Data Definition**

1581 The JPEG data generated in camera module is baseline JPEG DCT format defined in ISO/IEC 10918-1, with  
1582 following additional definitions or modifications:

- 1583 • sRGB color space shall be used. The JPEG is generated from YCbCr format after sRGB to YCbCr  
1584 conversion.
- 1585 • The JPEG metadata has to be EXIF compatible, i.e. metadata within application segments has to  
1586 be placed in beginning of file, in the order illustrated in Figure 129.
- 1587 • A status line is added in the end of JPEG data as defined in Section A.3.
- 1588 • If needed, an embedded image is interlaced in order which is free of choice as defined in Section  
1589 A.4.
- 1590 • Prior to storing into a file, the CSI-2 JPEG data is processed by the data separation process  
1591 described in Section A.1.



1592

1593

**Figure 129 EXIF Compatible Baseline JPEG DCT Format**

1594 **A.3 Image Status Information**

1595 Information of at least the following items has to be stored in the end of the JPEG sequence as illustrated in  
1596 Figure 130:

- 1597     • Image exposure time  
1598     • Analog & digital gains used  
1599     • White balancing gains for each color component  
1600     • Camera version number  
1601     • Camera register settings  
1602     • Image resolution and possible thumbnail resolution

1603 The camera register settings may include a subset of camera's registers. The essential information needed for  
1604 JPEG8 image is the information needed for converting the image back to linear space. This is necessary e.g.  
1605 for printing service. An example of register settings is following:

- 1606     • Sample frequency  
1607     • Exposure  
1608     • Analog and digital gain  
1609     • Gamma  
1610     • Color gamut conversion matrix  
1611     • Contrast  
1612     • Brightness  
1613     • Pre-gain

- 1614 The status information content has to be defined in the product specification of each camera module  
1615 containing the JPEG8 feature. The format and content is manufacturer specific.
- 1616 The image status data should be arranged so that each byte is split into two 4-bit nibbles and “1010” padding  
1617 sequence is added to MSB, as presented in the Table 28. This ensures that no JPEG escape sequences (0xFF  
1618 0x00) are present in the status data.
- 1619 The SOSI and EOSI markers are defined in Section A.5.

1620 **Table 28 Status Data Padding**

Data Word	After Padding
D7D6D5D4 D3D2D1D0	1010D7D6D5D4 1010D3D2D1D0

The diagram illustrates the structure of the Status Information Field in the End of Baseline JPEG Frame. It consists of a vertical stack of markers and data segments. The markers are: Start of Image (SOI), JFIF / EXIF Data, Quantization Table (DQT), Huffman Table (DHT), Frame Header (SOF), Scan Header, End Of Image (EOI), Start of Status Information (SOSI), Image Status Information, and End of Status Information (EOSI). Between the markers, there is a large central area labeled "Compressed Data".

1621  
1622 **Figure 130 Status Information Field in the End of Baseline JPEG Frame**

1623 **A.4 Embedded Images**

1624 An image may be embedded inside the JPEG data, if needed. The embedded image feature is not compulsory  
1625 for each camera module containing the JPEG8 feature. An example of embedded data is a 24-bit RGB  
1626 thumbnail image.

1627 The philosophy of embedded / interleaved thumbnail additions is to minimize the needed frame memory.  
1628 The EI (Embedded Image) data can be included in any part of the compressed image data segment and in as  
1629 many pieces as needed. See Figure 131.

1630 Embedded Image data is separated from compressed data by SOEI (Start Of Embedded Image) and EOEI  
1631 (End Of Embedded Image) non-standard markers, which are defined in Section A.5. The amount of fields  
1632 separated by SOEI and EOEI is not limited.

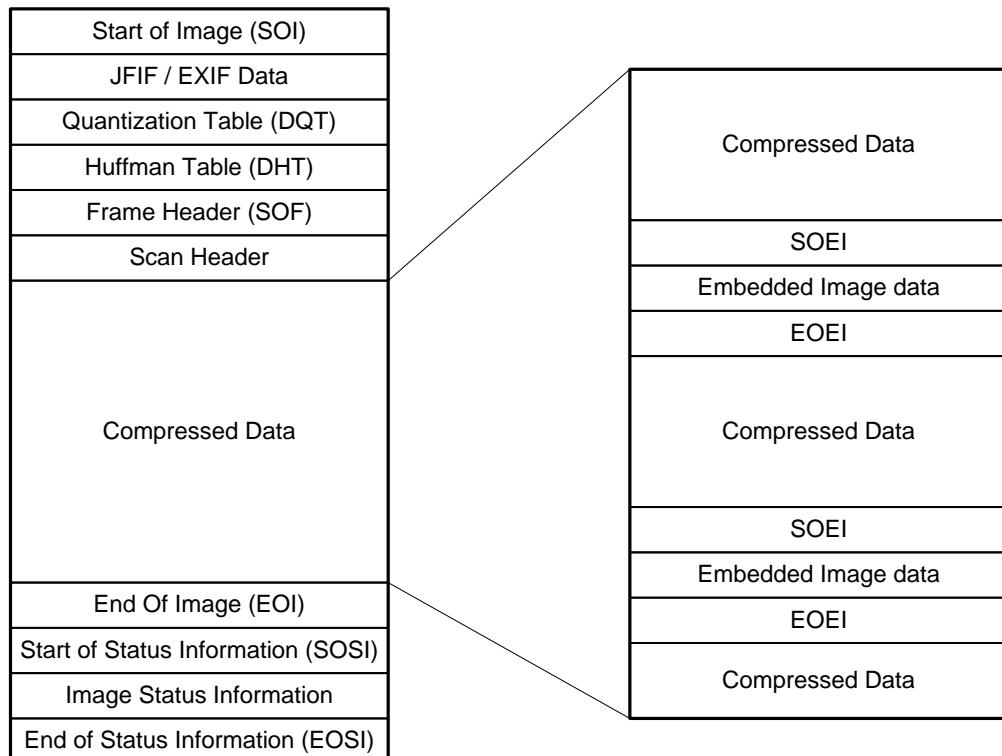
1633 The pixel to byte packing for image data within an EI data field should be as specified for the equivalent CSI-  
1634 2 data format. However there is an additional restriction; the embedded image data must not generate any  
1635 false JPEG marker sequences (0xFFXX).

1636 The suggested method of preventing false JPEG marker codes from occurring within the embedded image  
1637 data is to limit the data range for the pixel values. For example

- 1638 • For RGB888 data the suggested way to solve the false synchronization code issue is to constrain  
1639 the numerical range of R, G and B values from 1 to 254.
- 1640 • For RGB565 data the suggested way to solve the false synchronization code issue is to constrain  
1641 the numerical range of G component from 1-62 and R component from 1-30.

1642 Each EI data field is separated by the SOEI / EOEI markers, and has to contain an equal amount bytes and a  
1643 complete number of pixels. An EI data field may contain multiple lines or a full frame of image data.

1644 The embedded image data is decoded and removed apart from the JPEG compressed data prior to writing the  
1645 JPEG into a file. In the process, EI data fields are appended one after each other, in order of occurrence in  
1646 the received JPEG data.



1647 **Figure 131 Example of TN Image Embedding Inside the Compressed JPEG Data Block**

1649 **A.5 JPEG8 Non-standard Markers**

1650 JPEG8 uses the reserved JPEG data markers for special purposes, marking the additional segments inside the  
1651 data file. These segments are not part of the JPEG, JFIF [0], EXIF [0] or any other specifications; instead  
1652 their use is specified in this document in Section A.3 and Section A.4.

1653 The use of the non-standard markers is always internal to a product containing the JPEG8 camera module,  
1654 and these markers are always removed from the JPEG data before storing it into a file.

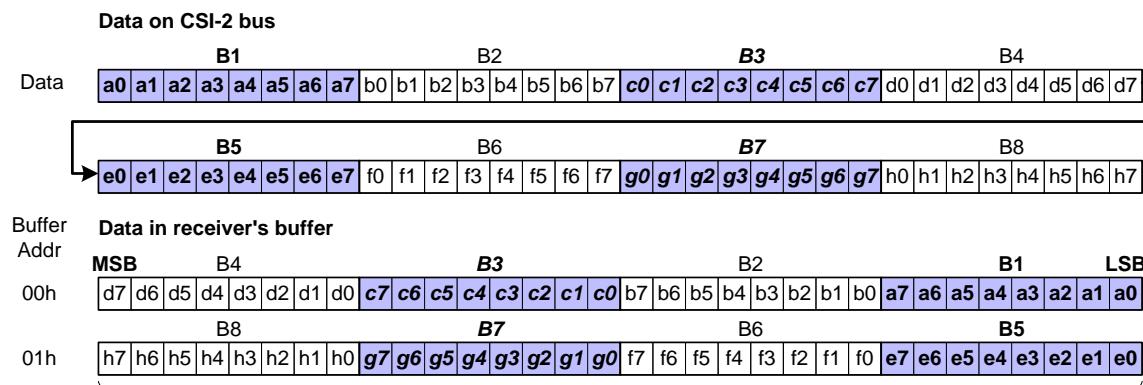
1655

**Table 29 JPEG8 Additional Marker Codes Listing**

Non-standard Marker Symbol	Marker Data Code
SOSI	0xFF 0xBC
EOSI	0xFF 0xBD
SOEI	0xFF 0xBE
EOEI	0xFF 0xBF

1656 **A.6 JPEG8 Data Reception**

1657 The compressed data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.



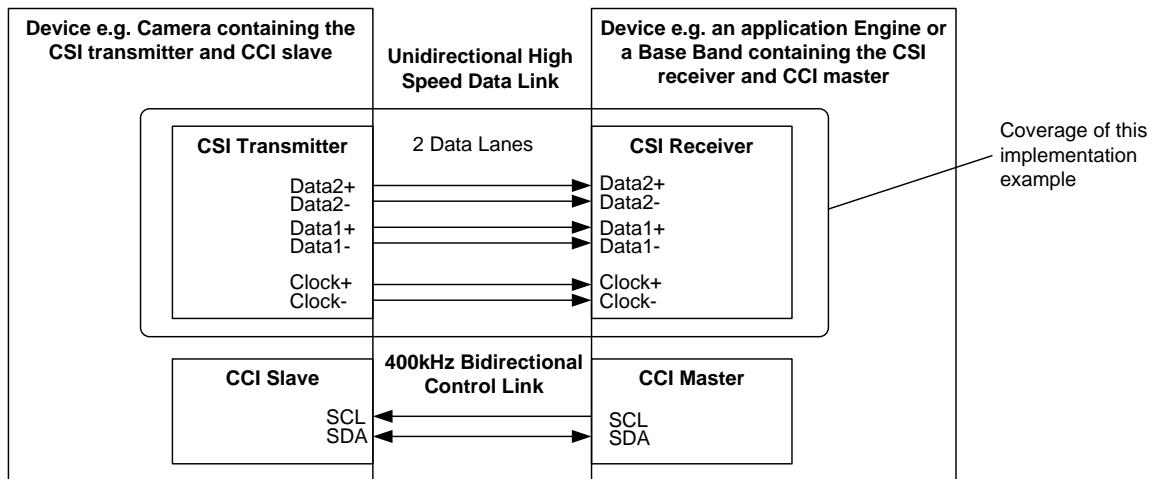
1658

**Figure 132 JPEG8 Data Format Reception**

## 1660 Annex B CSI-2 Implementation Example (informative)

### 1661 B.1 Overview

1662 The CSI-2 implementation example assumes that the interface comprises of D-PHY unidirectional Clock and  
1663 Data, with forward escape mode and optional deskew functionality. The scope in this implementation  
1664 example refers only to the unidirectional data link without any references to the CCI interface, as it can be  
1665 seen in Figure 133. This implementation example varies from the informative PPI example in [MIPI01].



1666  
1667 **Figure 133 Implementation Example Block Diagram and Coverage**

1668 For this implementation example a layered structure is described with the following parts:

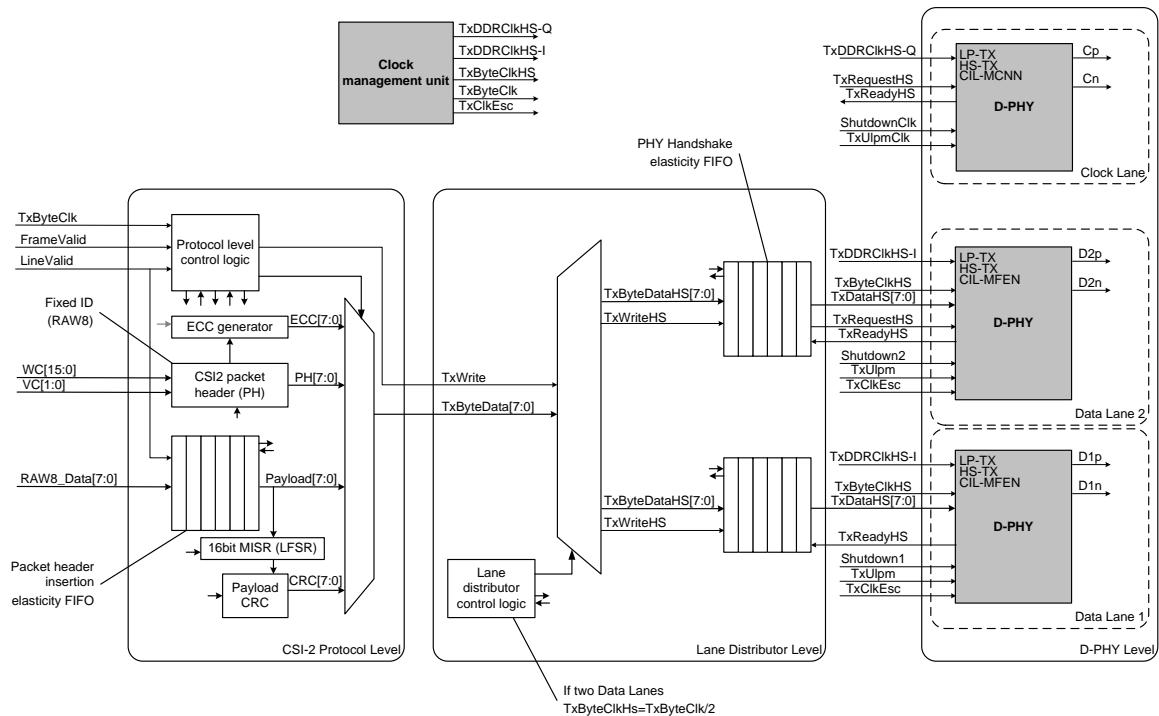
- 1669
  - D-PHY implementation details
  - Multi lane merger details
  - Protocol layer details
- 1670
- 1671

1672 This implementation example refers to a RAW8 data type only; hence no packing/unpacking or byte  
1673 clock/pixel clock timing will be referenced as for this type of implementation they are not needed.

1674 No error recovery mechanism or error processing details will be presented, as the intent of the document is  
1675 to present an implementation from the data flow perspective.

### 1676 B.2 CSI-2 Transmitter Detailed Block Diagram

1677 Using the layered structure described in the overview the CSI-2 transmitter could have the block diagram in  
1678 Figure 134.



1679

1680

**Figure 134 CSI-2 Transmitter Block Diagram**

1681 **B.3 CSI-2 Receiver Detailed Block Diagram**

1682 Using the layered structure described in the overview, the CSI-2 receiver could have the block diagram in  
1683 Figure 135.

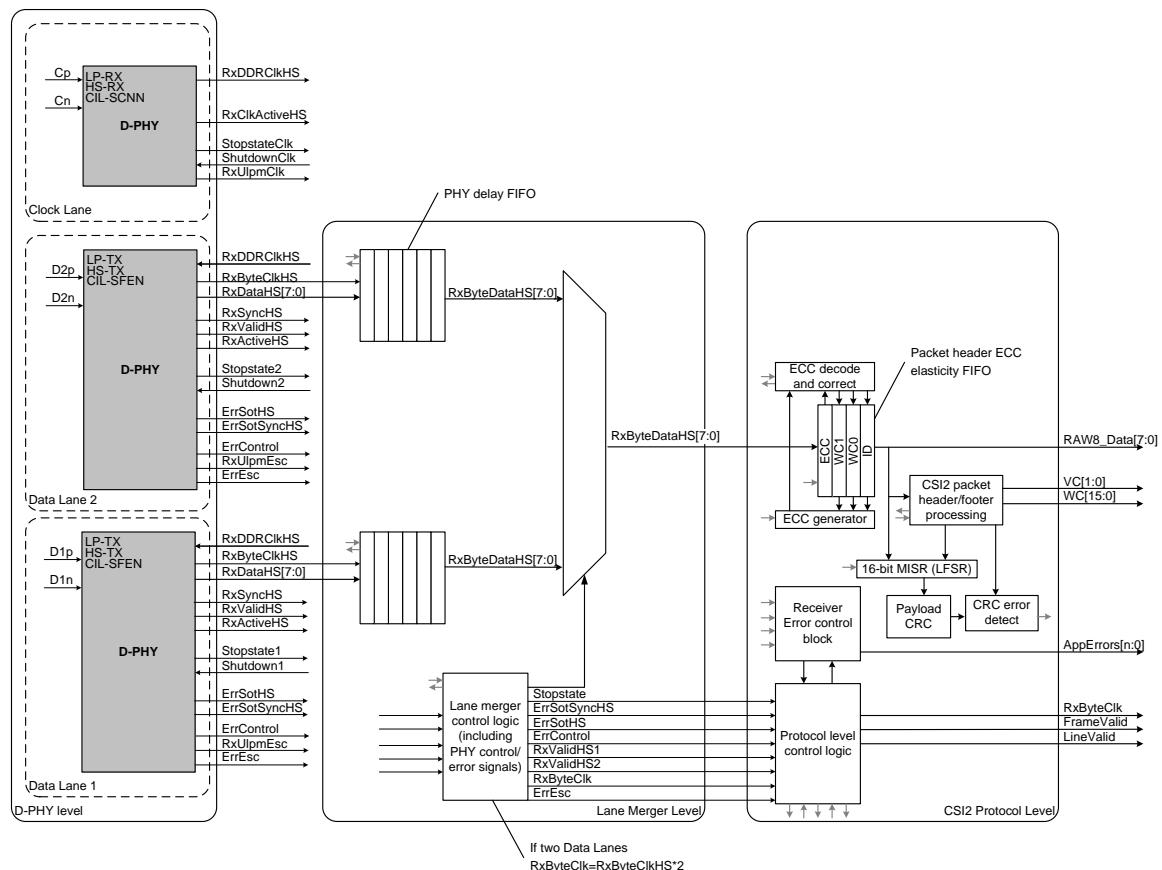
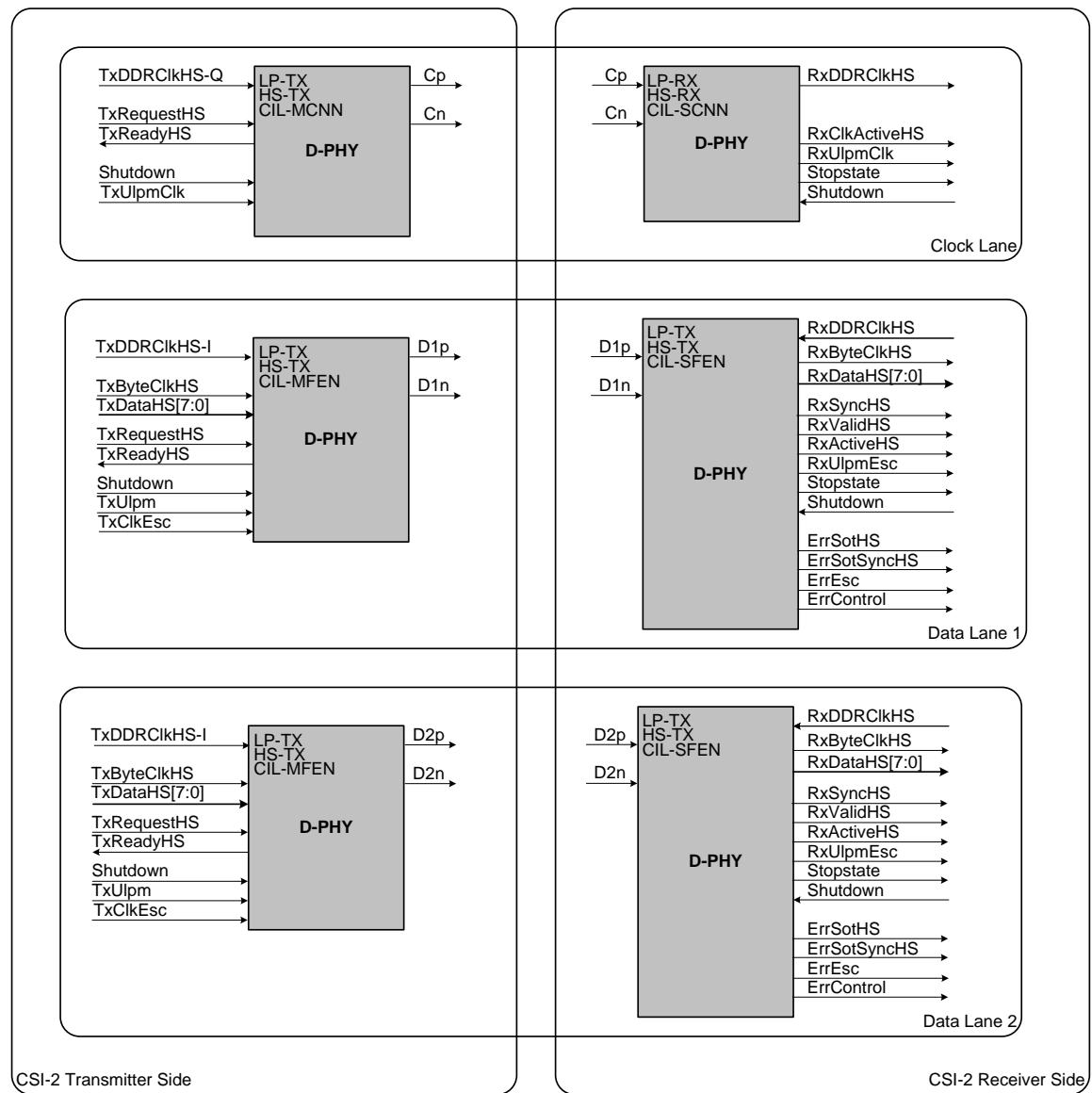


Figure 135 CSI-2 Receiver Block Diagram

1684      **B.4 Details on the D-PHY implementation**

1685      The PHY level of implementation has the top level structure as seen in Figure 136.



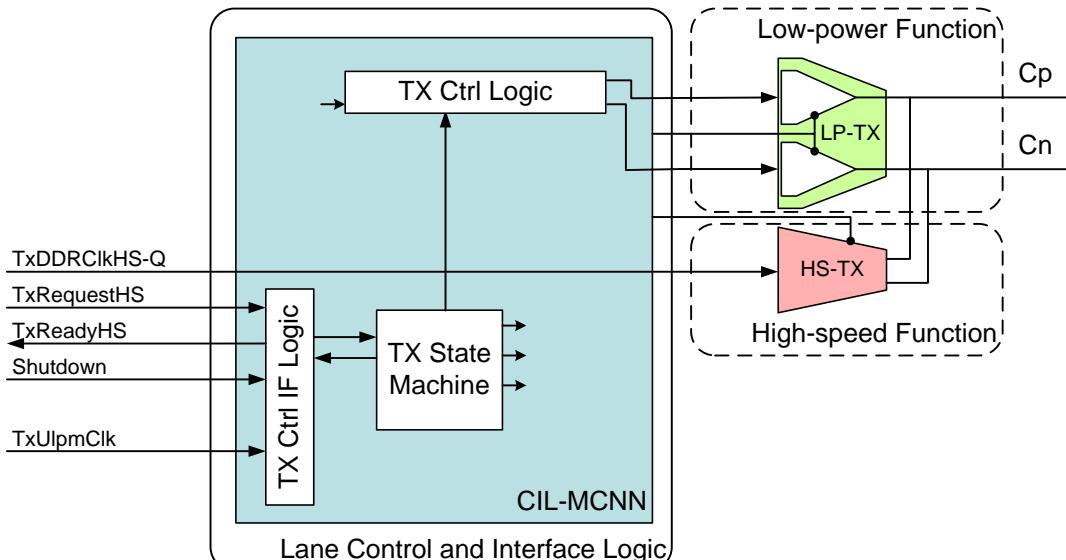
**Figure 136 D-PHY Level Block Diagram**

1690 The components can be categorized as:

- 1691 • CSI-2 Transmitter side:
  - 1692 • Clock lane (Transmitter)
  - 1693 • Data1 lane (Transmitter)
  - 1694 • Data2 lane (Transmitter)
- 1695 • CSI-2 Receiver side:
  - 1696 • Clock lane (Receiver)
  - 1697 • Data1 lane (Receiver)
  - 1698 • Data2 lane (Receiver)

1699 **B.4.1 CSI-2 Clock Lane Transmitter**

1700 The suggested implementation can be seen in Figure 137.



1701  
1702 **Figure 137 CSI-2 Clock Lane Transmitter**

1703 The modular D-PHY components used to build a CSI-2 clock lane transmitter are:

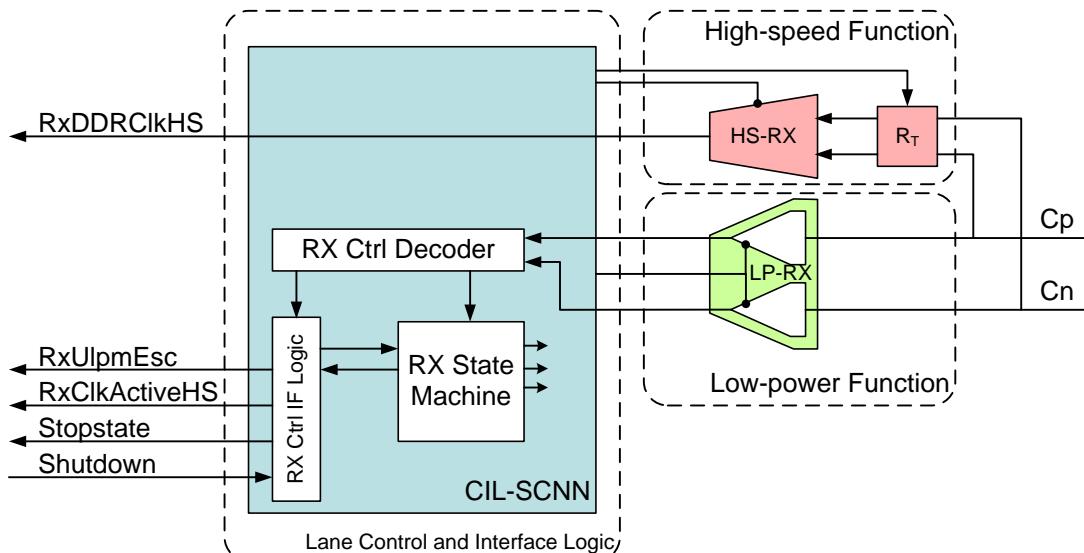
- **LP-TX** for the Low-power function
- **HS-TX** for the High-speed function
- **CIL-MCNN** for the Lane control and interface logic

1707 The PPI interface signals to the CSI-2 clock lane transmitter are:

- **TxDRClkHS-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.
- **TxUlpClk** (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 TxUlpClk is de-asserted.

1721 **B.4.2 CSI-2 Clock Lane Receiver**

1722 The suggested implementation can be seen in Figure 138.



1723

**Figure 138 CSI-2 Clock Lane Receiver**

1725 The modular D-PHY components used to build a CSI-2 clock lane receiver are:

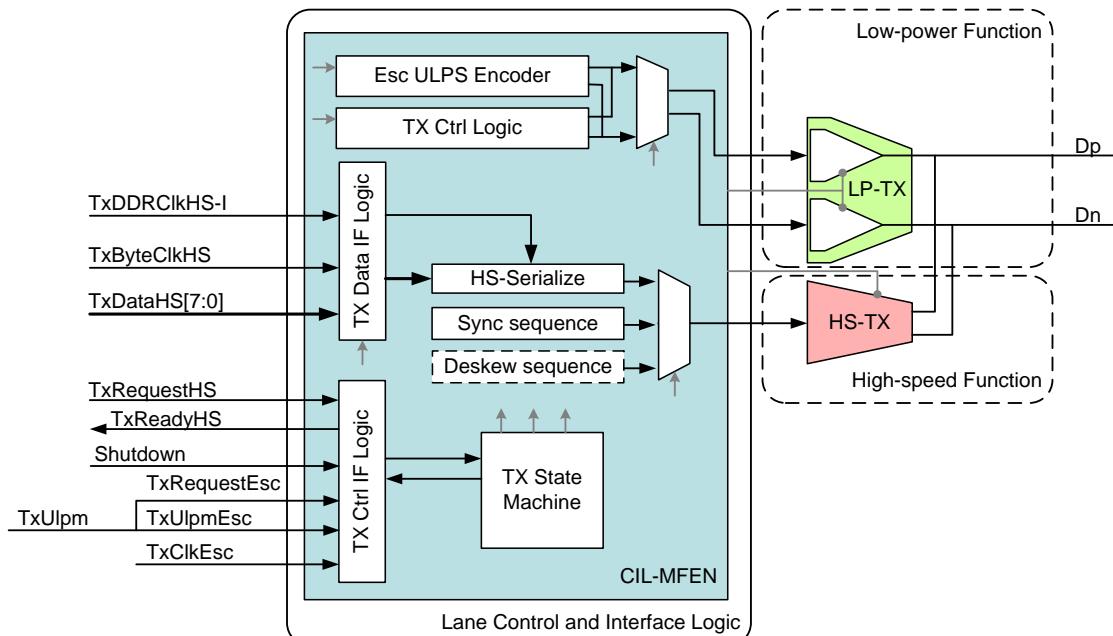
- 1726 • **LP-RX** for the Low-power function  
1727 • **HS-RX** for the High-speed function  
1728 • **CIL-SCNN** for the Lane control and interface logic

1729 The PPI interface signals to the CSI-2 clock lane receiver are:

- 1730 • **RxDDRClkHS** (Output): High-Speed Receive DDR Clock used to sample the data in all data  
1731 lanes.  
1732 • **RxClkActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that  
1733 the clock lane is receiving valid clock. This signal is asynchronous.  
1734 • **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is  
1735 currently in Stop state. This signal is asynchronous.  
1736 • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into  
1737 “shutdown”, disabling all activity. All line drivers, including terminators, are turned off when  
1738 Shutdown is asserted. When Shutdown is high, all PPI outputs are driven to the default inactive  
1739 state. Shutdown is a level sensitive signal and does not depend on any clock.  
1740 • **RxUlpEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is  
1741 asserted to indicate that the lane module has entered the ultra low power mode. The lane module  
1742 remains in this mode with RxUlpEsc asserted until a Stop state is detected on the lane  
1743 interconnect.

1744 **B.4.3 CSI-2 Data Lane Transmitter**

1745 The suggested implementation can be seen in Figure 139.



1746  
1747

1748 **Figure 139 CSI-2 Data Lane Transmitter**

1749 The modular D-PHY components used to build a CSI-2 data lane transmitter are:

- 1750 • **LP-TX** for the Low-power function
- 1751 • **HS-TX** for the High-speed function
- 1752 • **CIL-MFEN** for the Lane control and interface logic. For optional deskew calibration support, the
- 1753 data lane transmitter transmits a deskew sequence. The deskew sequence transmission is enabled
- 1754 by a mechanism out of the scope of this specification.

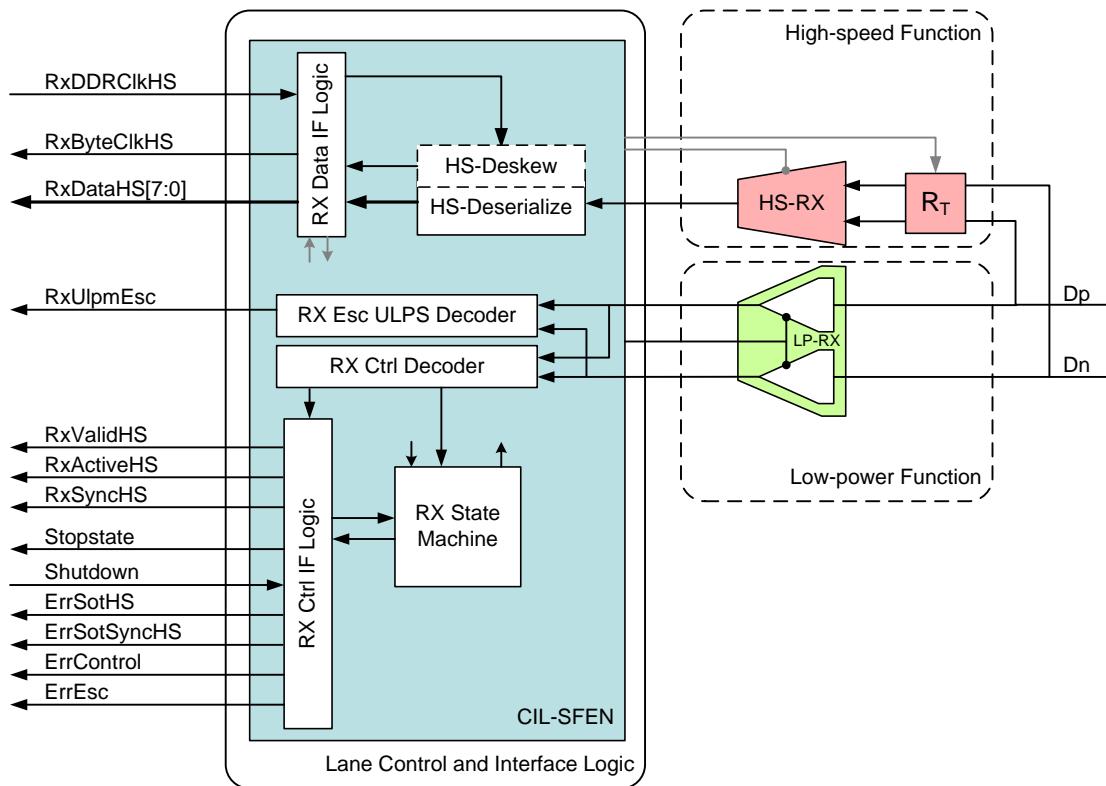
1755 The PPI interface signals to the CSI-2 data lane transmitter are:

- 1756 • **TxDRClkHS-I** (Input): High-Speed Transmit DDR Clock (in-phase).
- 1757 • **TxByteClkHS** (Input): High-Speed Transmit Byte Clock. This is used to synchronize PPI signals
- 1758 in the high-speed transmit clock domain. It is recommended that both transmitting data lane
- 1759 modules share one TxByteClkHS signal. The frequency of TxByteClkHS must be exactly 1/8 the
- 1760 high-speed bit rate.
- 1761 • **TxDataHS[7:0]** (Input): High-Speed Transmit Data. Eight bit high-speed data to be transmitted.
- 1762 The signal connected to TxDataHS[0] is transmitted first. Data is registered on rising edges of
- 1763 TxByteClkHS.
- 1764 • **TxRequestHS** (Input): High-Speed Transmit Request. A low-to-high transition on TxRequestHS
- 1765 causes the lane module to initiate a Start-of-Transmission sequence. A high-to-low transition on
- 1766 TxRequest causes the lane module to initiate an End-of-Transmission sequence. This active high
- 1767 signal also indicates that the protocol is driving valid data on TxByteDataHS to be transmitted.
- 1768 The lane module accepts the data when both TxRequestHS and TxReadyHS are active on the
- 1769 same rising TxByteClkHS clock edge. The protocol always provides valid transmit data when

- 1770      TxRequestHS is active. Once asserted, TxRequestHS should remain high until the all the data has  
1771      been accepted.
- 1772      • **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that  
1773      TxDataHS is accepted by the lane module to be serially transmitted. TxReadyHS is valid on rising  
1774      edges of TxByteClkHS. Valid data has to be provided for the whole duration of active  
1775      TxReadyHS.
- 1776      • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into  
1777      “shutdown”, disabling all activity. All line drivers, including terminators, are turned off when  
1778      Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs  
1779      are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on  
1780      any clock.
- 1781      • **TxUlpmEsc** (Input): Escape mode Transmit Ultra Low Power. This active high signal is asserted  
1782      with TxRequestEsc to cause the lane module to enter the ultra low power mode. The lane module  
1783      remains in this mode until TxRequestEsc is de-asserted.
- 1784      • **TxRequestEsc** (Input): This active high signal, asserted together with TxUlpmEsc is used to  
1785      request entry into escape mode. Once in escape mode, the lane stays in escape mode until  
1786      TxRequestEsc is de-asserted. TxRequestEsc is only asserted by the protocol while TxRequestHS  
1787      is low.
- 1788      • **TxClockEsc** (Input): Escape mode Transmit Clock. This clock is directly used to generate escape  
1789      sequences. The period of this clock determines the symbol time for low power signals. It is  
1790      therefore constrained by the normative part of the [MIPI01].

1791      **B.4.4      CSI-2 Data Lane Receiver**

1792      The suggested implementation can be seen in Figure 140.



1793

1794

**Figure 140 CSI-2 Data Lane Receiver**

1795 The modular D-PHY components used to build a CSI-2 data lane receiver are:

- 1796 • **LP-RX** for the Low-power function
- 1797 • **HS-RX** for the High-speed function
- 1798 • **CIL-SFEN** for the Lane control and interface logic. For optional deskew calibration support the
- 1799 data lane receiver detects a transmitted deskew calibration pattern and performs optimum deskew
- 1800 of the Data with respect to the RxDDRClkHS Clock.

1801 The PPI interface signals to the CSI-2 data lane receiver are:

- 1802 • **RxDDRClkHS** (Input): High-Speed Receive DDR Clock used to sample the date in all data lanes.  
1803 This signal is supplied by the CSI-2 clock lane receiver.
- 1804 • **RxByteClkHS** (Output): High-Speed Receive Byte Clock. This signal is used to synchronize  
1805 signals in the high-speed receive clock domain. The RxByteClkHS is generated by dividing the  
1806 received RxDDRClkHS.
- 1807 • **RXDataHS[7:0]** (Output): High-Speed Receive Data. Eight bit high-speed data received by the  
1808 lane module. The signal connected to RxDataHS[0] was received first. Data is transferred on  
1809 rising edges of RxByteClkHS.
- 1810 • **RxValidHS** (Output): High-Speed Receive Data Valid. This active high signal indicates that the  
1811 lane module is driving valid data to the protocol on the RxDataHS output. There is no  
1812 “RxReadyHS” signal, and the protocol is expected to capture RxDataHS on every rising edge of  
1813 RxByteClkHS where RxValidHS is asserted. There is no provision for the protocol to slow down  
1814 (“throttle”) the receive data.

- 1815 • **RxActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the  
1816 lane module is actively receiving a high-speed transmission from the lane interconnect.
- 1817 • **RxSyncHS** (Output): Receiver Synchronization Observed. This active high signal indicates that  
1818 the lane module has seen an appropriate synchronization event. In a typical high-speed  
1819 transmission, RxSyncHS is high for one cycle of RxByteClkHS at the beginning of a high-speed  
1820 transmission when RxActiveHS is first asserted. This signal missing is signaled using  
1821 ErrSotSyncHS.
- 1822 • **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is  
1823 asserted to indicate that the lane module has entered the ultra low power mode. The lane module  
1824 remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane  
1825 interconnect.
- 1826 • **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is  
1827 currently in Stop state. This signal is asynchronous.
- 1828 • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into  
1829 “shutdown”, disabling all activity. All line drivers, including terminators, are turned off when  
1830 Shutdown is asserted. When Shutdown is high, all PPI outputs are driven to the default inactive  
1831 state. Shutdown is a level sensitive signal and does not depend on any clock.
- 1832 • **ErrSotHS** (Output): Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is  
1833 corrupted, but in such a way that proper synchronization can still be achieved, this error signal is  
1834 asserted for one cycle of RxByteClkHS. This is considered to be a “soft error” in the leader  
1835 sequence and confidence in the payload data is reduced.
- 1836 • **ErrSotSyncHS** (Output): Start-of-Transmission Synchronization Error. If the high-speed SoT  
1837 leader sequence is corrupted in a way that proper synchronization cannot be expected, this error is  
1838 asserted for one cycle of RxByteClkHS.
- 1839 • **ErrControl** (Output): Control Error. This signal is asserted when an incorrect line state sequence  
1840 is detected.
- 1841 • **ErrEsc** (Output): Escape Entry Error. If an unrecognized escape entry command is received, this  
1842 signal is asserted and remains high until the next change in line state. The only escape entry  
1843 command supported by the receiver is the ULPS.

## 1844 Annex C CSI-2 Recommended Receiver Error Behavior (informative)

### 1845 C.1 Overview

1846 This section proposes one approach to handling error conditions at the receiving side of a CSI-2 Link.  
1847 Although the section is informative and therefore does not affect compliance for CSI-2, the approach is  
1848 offered by the MIPI Camera Working Group as a recommended approach. The CSI-2 receiver assumes the  
1849 case of a CSI-2 Link comprised of unidirectional Lanes for D-PHY Clock and Data Lanes with Escape Mode  
1850 functionality on the Data Lanes and a continuously running clock. This Annex does not discuss other cases,  
1851 including those that differ widely in implementation, where the implementer should consider other potential  
1852 error situations.

1853 Because of the layered structure of a compliant CSI-2 receiver implementation, the error behavior is  
1854 described in a similar way with several “levels” where errors could occur, each requiring some  
1855 implementation at the appropriate functional layer of the design:

- 1856 • *D-PHY Level errors*  
1857 Refers to any PHY related transmission error and is unrelated to the transmission’s contents:
  - 1858 • *Start of Transmission (SoT) errors*, which can be:
    - 1859 • Recoverable, if the PHY successfully identifies the Sync code but an error was detected.
    - 1860 • Unrecoverable, if the PHY does not successfully identify the sync code but does detect a  
1861 HS transmission.
  - 1862 • *Control Error*, which signals that the PHY has detected a control sequence that should not be  
1863 present in this implementation of the Link.
- 1864 • *Packet Level errors*  
1865 This type of error refers strictly to data integrity of the received Packet Header and payload data:
  - 1866 • *Packet Header errors*, signaled through the ECC code, that result in:
    - 1867 • A single bit-error, which can be detected and corrected by the ECC code
    - 1868 • Two bit-errors in the header, which can be detected but not corrected by the ECC code,  
1869 resulting in a corrupt header
  - 1870 • *Packet payload errors*, signaled through the CRC code
- 1871 • *Protocol Decoding Level errors*  
1872 This type of error refers to errors present in the decoded Packet Header or errors resulting from an  
1873 incomplete sequence of events:
  - 1874 • *Frame Sync Error*, caused when a FS could not be successfully paired with a FE on a given  
1875 virtual channel
  - 1876 • *Unrecognized ID*, caused by the presence of an unimplemented or unrecognized ID in the  
1877 header

1878 The proposed methodology for handling errors is signal based, since it offers an easy path to a viable CSI-2  
1879 implementation that handles all three error levels. Even so, error handling at the Protocol Decoding Level  
1880 should implement sequential behavior using a state machine for proper operation.

### 1881 C.2 D-PHY Level Error

1882 The recommended behavior for handling this error level covers only those errors generated by the Data  
1883 Lane(s), since an implementation can assume that the Clock Lane is running reliably as provided by the  
1884 expected BER of the Link, as discussed in [MIPI01]. Note that this error handling behavior assumes

1885 unidirectional Data Lanes without escape mode functionality. Considering this, and using the signal names  
1886 and descriptions from the [MIPI01], PPI Annex, signal errors at the PHY-Protocol Interface (PPI) level  
1887 consist of the following:

- 1888 • **ErrSotHS:** Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is  
1889 corrupted, but in such a way that proper synchronization can still be achieved, this error signal is  
1890 asserted for one cycle of RxByteClkHS. This is considered to be a “soft error” in the leader  
1891 sequence and confidence in the payload data is reduced.
- 1892 • **ErrSotSyncHS:** Start-of-Transmission Synchronization Error. If the high-speed SoT leader  
1893 sequence is corrupted in a way that proper synchronization cannot be expected, this error signal is  
1894 asserted for one cycle of RxByteClkHS.
- 1895 • **ErrControl:** Control Error. This signal is asserted when an incorrect line state sequence is  
1896 detected. For example, if a Turn-around request or Escape Mode request is immediately followed  
1897 by a Stop state instead of the required Bridge state, this signal is asserted and remains high until  
1898 the next change in line state.

1899 The recommended receiver error behavior for this level is:

- 1900 • **ErrSotHS** should be passed to the Application Layer. Even though the error was detected and  
1901 corrected and the Sync mechanism was unaffected, confidence in the data integrity is reduced and  
1902 the application should be informed. This signal should be referenced to the corresponding data  
1903 packet.
- 1904 • **ErrSotSyncHS** should be passed to the Protocol Decoding Level, since this is an unrecoverable  
1905 error. An unrecoverable type of error should also be signaled to the Application Layer, since the  
1906 whole transmission until the first D-PHY Stop state should be ignored if this type of error occurs.
- 1907 • **ErrControl** should be passed to the Application Layer, since this type of error doesn't normally  
1908 occur if the interface is configured to be unidirectional. Even so, the application needs to be aware  
1909 of the error and configure the interface accordingly through other, implementation specific means.

1910 Also, it is recommended that the PPI StopState signal for each implemented Lane should be propagated to  
1911 the Application Layer during configuration or initialization to indicate the Lane is ready.

### 1912 C.3 Packet Level Error

1913 The recommended behavior for this error level covers only errors recognized by decoding the Packet  
1914 Header's ECC byte and computing the CRC of the data payload.

1915 Decoding and applying the ECC byte of the Packet Header should signal the following errors:

- 1916 • **ErrEccDouble:** Asserted when an ECC syndrome was computed and two bit-errors are detected  
1917 in the received Packet Header.
- 1918 • **ErrEccCorrected:** Asserted when an ECC syndrome was computed and a single bit-error in the  
1919 Packet Header was detected and corrected.
- 1920 • **ErrEccNoError:** Asserted when an ECC syndrome was computed and the result is zero  
1921 indicating a Packet Header that is considered to be without errors or has more than two bit-errors.  
1922 CSI-2's ECC mechanism cannot detect this type of error.

1923 Also, computing the CRC code over the whole payload of the received packet could generate the following  
1924 errors:

- 1925 • **ErrCrc:** Asserted when the computed CRC code is different than the received CRC code.
- 1926 • **ErrID:** Asserted when a Packet Header is decoded with an unrecognized or unimplemented data  
1927 ID.

1928 The recommended receiver error behavior for this level is:

- 1929 • **ErrEccDouble** should be passed to the Application Layer since assertion of this signal proves that  
1930 the Packet Header information is corrupt, and therefore the WC is not usable, and thus the packet  
1931 end cannot be estimated. Commonly, this type of error will be accompanied with an ErrCrc. This  
1932 type of error should also be passed to the Protocol Decoding Level, since the whole transmission  
1933 until D-PHY Stop state should be ignored.
- 1934 • **ErrEccCorrected** should be passed to the Application Layer since the application should be  
1935 informed that an error had occurred but was corrected, so the received Packet Header was  
1936 unaffected, although the confidence in the data integrity is reduced.
- 1937 • **ErrEccNoError** can be passed to the Protocol Decoding Level to signal the validity of the current  
1938 Packet Header.
- 1939 • **ErrCrc** should be passed to the Protocol Decoding Level to indicate that the packet's payload data  
1940 might be corrupt.
- 1941 • **ErrID** should be passed to the Application Layer to indicate that the data packet is unidentified  
1942 and cannot be unpacked by the receiver. This signal should be asserted after the ID has been  
1943 identified and de-asserted on the first Frame End (FE) on same virtual channel.

1944 **C.4 Protocol Decoding Level Error**

1945 The recommended behavior for this error level covers errors caused by decoding the Packet Header  
1946 information and detecting a sequence that is not allowed by the CSI-2 protocol or a sequence of detected  
1947 errors by the previous layers. CSI-2 implementers will commonly choose to implement this level of error  
1948 handling using a state machine that should be paired with the corresponding virtual channel. The state  
1949 machine should generate at least the following error signals:

- 1950 • **ErrFrameSync:** Asserted when a Frame End (FE) is not paired with a Frame Start (FS) on the  
1951 same virtual channel. An ErrSotSyncHS should also generate this error signal.
- 1952 • **ErrFrameData:** Asserted after a FE when the data payload received between FS and FE contains  
1953 errors.

1954 The recommended receiver error behavior for this level is:

- 1955 • **ErrFrameSync** should be passed to the Application Layer with the corresponding virtual channel,  
1956 since the frame could not be successfully identified. Several error cases on the same virtual  
1957 channel can be identified for this type of error.
  - 1958 • If a FS is followed by a second FS on the same virtual channel, the frame corresponding to the  
1959 first FS is considered in error.
  - 1960 • If a Packet Level ErrEccDouble was signaled from the Protocol Layer, the whole transmission  
1961 until the first D-PHY Stop-state should be ignored since it contains no information that can be  
1962 safely decoded and cannot be qualified with a data valid signal.
  - 1963 • If a FE is followed by a second FE on the same virtual channel, the frame corresponding to  
1964 the second FE is considered in error.
  - 1965 • If an ErrSotSyncHS was signaled from the PHY Layer, the whole transmission until the first  
1966 D-PHY Stop state should be ignored since it contains no information that can be safely  
1967 decoded and cannot be qualified with a data valid signal.
- 1968 • **ErrFrameData:** should be passed to the Application Layer to indicate that the frame contains data  
1969 errors. This signal should be asserted on any ErrCrc and de-asserted on the first FE.

1970 **Annex D CSI-2 Sleep Mode (informative)**

1971 **D.1 Overview**

1972 Since a camera in a mobile terminal spends most of its time in an inactive state, implementers need a way to  
1973 put the CSI-2 Link into a low power mode that approaches, or may be as low as, the leakage level. This  
1974 section proposes one approach for putting a CSI-2 Link in a “Sleep Mode” (SLM). Although the section is  
1975 informative and therefore does not affect compliance for CSI-2, the approach is offered by the MIPI Camera  
1976 Working Group as a recommended approach.

1977 This approach relies on an aspect of a D-PHY transmitter’s behavior that permits regulators to be disabled  
1978 safely when LP-00 (Space state) is on the Link. Accordingly, this will be the output state for a CSI-2 camera  
1979 transmitter in SLM.

1980 SLM can be thought of as a three-phase process:

- 1981 1. SLM Command Phase. The ‘ENTER SLM’ command is issued to the TX side only, or to both sides  
1982 of the Link.
- 1983 2. SLM Entry Phase. The CSI-2 Link has entered, or is entering, the SLM in a controlled or  
1984 synchronized manner. This phase is also part of the power-down process.
- 1985 3. SLM Exit Phase. The CSI-2 Link has exited the SLM and the interface/device is operational. This  
1986 phase is also part of the power-up process.

1987 In general, when in SLM, both sides of the interface will be in ULPS, as defined in [MIPI01].

1988 **D.2 SLM Command Phase**

1989 For the first phase, initiation of SLM occurs by a mechanism outside the scope of CSI-2. Of the many  
1990 mechanisms available, two examples would be:

- 1991 1. An External SLEEP signal input to the CSI-2 transmitter and optionally also to the CSI-2 Receiver.  
1992 When at logic 0, the CSI-2 Transmitter and the CSI Receiver (if connected) will enter Sleep mode.  
1993 When at logic 1, normal operation will take place.
- 1994 2. A CCI control command, provided on the I2C control Link, is used to trigger ULPS.

1995 **D.3 SLM Entry Phase**

1996 For the second phase, consider one option:

1997 Only the TX side enters SLM and propagates the ULPS to the RX side by sending a D-PHY ‘ULPS’  
1998 command on Clock Lane and on Data Lane(s). In the following picture only Data Lane ‘ULPS’ command is  
1999 used as an example.

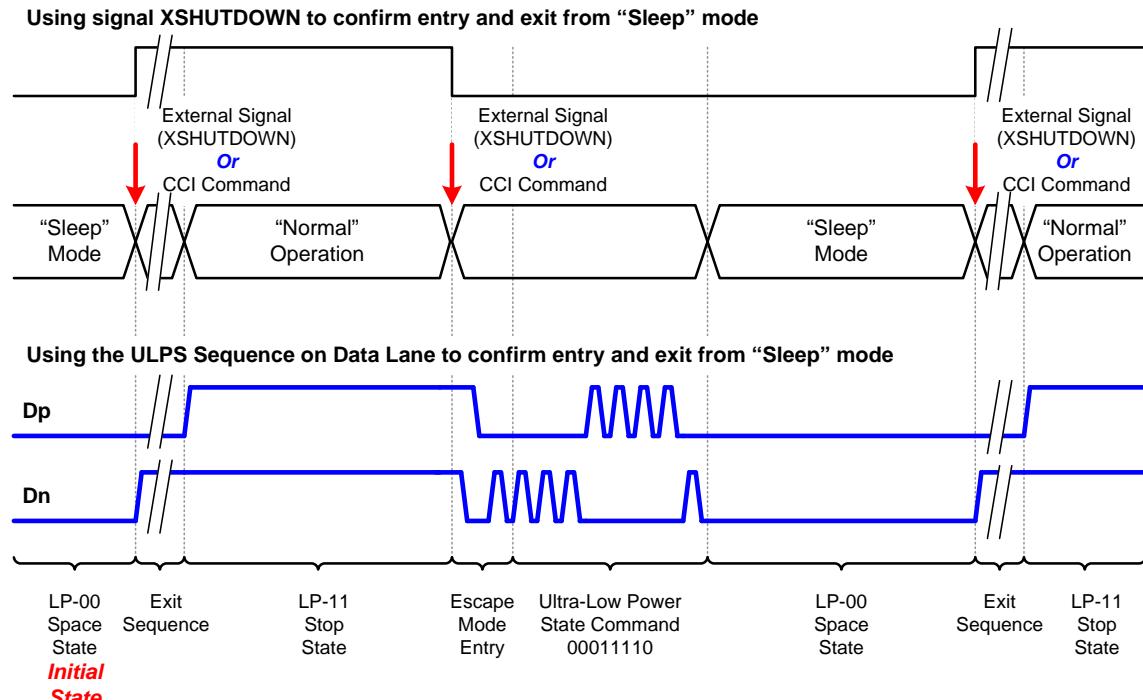


Figure 141 SLM Synchronization

#### 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 in the as described in [MIPI01] should be used for powering up the interface.

## 2015 Annex E Data Compression for RAW Data Types (normative)

2016 A CSI-2 implementation using RAW data types may support compression on the interface to reduce the data  
2017 bandwidth requirements between the host processor and a camera module. Data compression is not mandated  
2018 by this Specification. However, if data compression is used, it shall be implemented as described in this  
2019 annex.

2020 Data compression schemes use an X–Y–Z naming convention where X is the number of bits per pixel in the  
2021 original image, Y is the encoded (compressed) bits per pixel and Z is the decoded (uncompressed) bits per  
2022 pixel.

2023 The following data compression schemes are defined:

- 12–8–12

- 12–7–12

- 12–6–12

- 10–8–10

- 10–7–10

- 10–6–10

2030 To identify the type of data on the CSI-2 interface, packets with compressed data shall have a User Defined  
2031 Data Type value as indicated in Table 27. Note that User Defined data type codes are not reserved for  
2032 compressed data types. Therefore, a CSI-2 device shall be able to communicate over the CCI the data  
2033 compression scheme represented by a particular User Defined data type code for each scheme supported by  
2034 the device. Note that the method to communicate the data compression scheme to Data Type code mapping  
2035 is beyond the scope of this document.

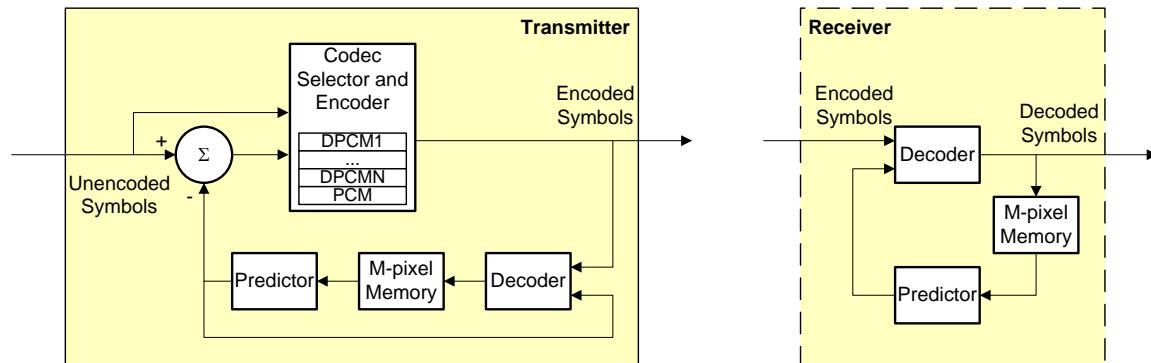
2036 The number of bits in a packet shall be a multiple of eight. Therefore, implementations with data compression  
2037 schemes that result in each pixel having less than eight encoded bits per pixel shall transfer the encoded data  
2038 in a packed pixel format. For example, the 12–7–12 data compression scheme uses a packed pixel format as  
2039 described in Section 11.4.2 except the Data Type value in the Packet Header is a User Defined data type  
2040 code.

2041 The data compression schemes in this annex are lossy and designed to encode each line independent of the  
2042 other lines in the image.

2043 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

2049 The data compression system consists of encoder, decoder and predictor blocks as shown in Figure 142.



**Figure 142 Data Compression System Block Diagram**

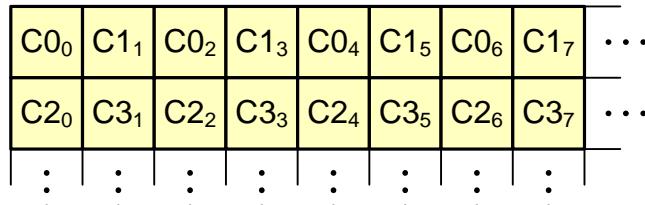
The encoder uses a simple algorithm to encode the pixel values. A fixed number of pixel values at the beginning of each line are encoded without using prediction. These first few values are used to initialize the predictor block. The remaining pixel values on the line are encoded using prediction.

If the predicted value of the pixel (**Xpred**) is close enough to the original value of the pixel (**Xorig**) ( $\text{abs}(\text{Xorig} - \text{Xpred}) <$  difference limit), its difference value (**Xdiff**) is quantized using a DPCM codec. Otherwise, **Xorig** is quantized using a PCM codec. The quantized value is combined with a code word describing the codec used to quantize the pixel and the sign bit, if applicable, to create the encoded value (**Xenco**).

## E.1 Predictors

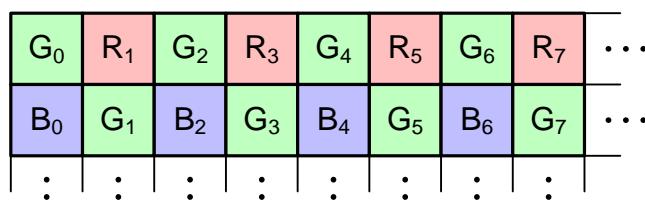
In order to have meaningful data transfer, both the transmitter and the receiver need to use the same predictor block.

The order of pixels in a raw image is shown in Figure 143.



**Figure 143 Pixel Order of the Original Image**

Figure 144 shows an example of the pixel order with RGB data.



**Figure 144 Example Pixel Order of the Original Image**

Two predictors are defined for use in the data compression schemes.

2070 Predictor1 uses a very simple algorithm and is intended to minimize processing power and memory size  
2071 requirements. Typically, this predictor is used when the compression requirements are modest and the  
2072 original image quality is high. Predictor1 should be used with 10–8–10, 10–7–10 and 12–8–12 data  
2073 compression schemes.

2074 The second predictor, Predictor2, is more complex than Predictor1. This predictor provides slightly better  
2075 prediction than Predictor1 and therefore the decoded image quality can be improved compared to Predictor1.  
2076 Predictor2 should be used with 10–6–10, 12–7–12, and 12–6–12 data compression schemes.

2077 Both receiver and transmitter shall support Predictor1 for all data compression schemes.

### 2078 **E.1.1 Predictor1**

2079 Predictor1 uses only the previous same color component value as the prediction value. Therefore, only a  
2080 two-pixel deep memory is required.

2081 The first two pixels ( $C_{00}$ ,  $C_{11}$  /  $C_{20}$ ,  $C_{31}$  or as in example  $G_0$ ,  $R_1$  /  $B_0$ ,  $G_1$ ) in a line are encoded without  
2082 prediction.

2083 The prediction values for the remaining pixels in the line are calculated using the previous same color  
2084 decoded value, **Xdeco**. Therefore, the predictor equation can be written as follows:

2085 
$$\text{Xpred}(n) = \text{Xdeco}(n-2)$$

### 2086 **E.1.2 Predictor2**

2087 Predictor2 uses the four previous pixel values, when the prediction value is evaluated. This means that also  
2088 the other color component values are used, when the prediction value has been defined. The predictor  
2089 equations can be written as shown in the following formulas.

2090 Predictor2 uses all color components of the four previous pixel values to create the prediction value.  
2091 Therefore, a four-pixel deep memory is required.

2092 The first pixel ( $C_{00}$  /  $C_{20}$ , or as in example  $G_0$  /  $B_0$ ) in a line is coded without prediction.

2093 The second pixel ( $C_{11}$  /  $C_{31}$  or as in example  $R_1$  /  $G_1$ ) in a line is predicted using the previous decoded different  
2094 color value as a prediction value. The second pixel is predicted with the following equation:

2095 
$$\text{Xpred}(n) = \text{Xdeco}(n-1)$$

2096 The third pixel ( $C_{02}$  /  $C_{22}$  or as in example  $G_2$  /  $B_2$ ) in a line is predicted using the previous decoded same  
2097 color value as a prediction value. The third pixel is predicted with the following equation:

2098 
$$\text{Xpred}(n) = \text{Xdeco}(n-2)$$

2099 The fourth pixel ( $C_{13}$  /  $C_{33}$  or as in example  $R_3$  /  $G_3$ ) in a line is predicted using the following equation:

```
2100     if ((Xdeco(n-1) <= Xdeco(n-2) AND Xdeco(n-2) <= Xdeco(n-3)) OR  
2101         (Xdeco(n-1) >= Xdeco(n-2) AND Xdeco(n-2) >= Xdeco(n-3))) then  
2102             Xpred(n) = Xdeco(n-1)  
2103         else  
2104             Xpred(n) = Xdeco(n-2)  
2105         endif
```

2106 Other pixels in all lines are predicted using the equation:

```
2107     if ((Xdeco( n-1 ) <= Xdeco( n-2 ) AND Xdeco( n-2 ) <= Xdeco( n-3 )) OR
2108         (Xdeco( n-1 ) >= Xdeco( n-2 ) AND Xdeco( n-2 ) >= Xdeco( n-3 ))) then
2109         Xpred( n ) = Xdeco( n-1 )
2110     else if ((Xdeco( n-1 ) <= Xdeco( n-3 ) AND Xdeco( n-2 ) <= Xdeco( n-4 )) OR
2111         (Xdeco( n-1 ) >= Xdeco( n-3 ) AND Xdeco( n-2 ) >= Xdeco( n-4 ))) then
2112         Xpred( n ) = Xdeco( n-2 )
2113     else
2114         Xpred( n ) = (Xdeco( n-2 ) + Xdeco( n-4 ) + 1) / 2
2115     endif
```

2116 **E.2 Encoders**

2117 There are six different encoders available, one for each data compression scheme.

2118 For all encoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
2119 for predicted pixels.

2120 **E.2.1 Coder for 10–8–10 Data Compression**

2121 The 10–8–10 coder offers a 20% bit rate reduction with very high image quality.

2122 Pixels without prediction are encoded using the following formula:

2123  $Xenco( n ) = Xorig( n ) / 4$

2124 To avoid a full-zero encoded value, the following check is performed:

```
2125     if (Xenco( n ) == 0) then
2126         Xenco( n ) = 1
2127     endif
```

2128 Pixels with prediction are encoded using the following formula:

```
2129     if (abs(Xdiff( n )) < 32) then
2130         use DPCM1
2131     else if (abs(Xdiff( n )) < 64) then
2132         use DPCM2
2133     else if (abs(Xdiff( n )) < 128) then
2134         use DPCM3
2135     else
2136         use PCM
2137     endif
```

2138 **E.2.1.1 DPCM1 for 10–8–10 Coder**

2139  $Xenco( n )$  has the following format:

2140  $Xenco( n ) = "00 s xxxxx"$

2141 where,

2142 “00” is the code word  
2143 “s” is the **sign** bit  
2144 “xxxx” is the five bit **value** field

2145 The coder equation is described as follows:

```
2146     if (Xdiff( n ) <= 0) then
2147         sign = 1
2148     else
2149         sign = 0
2150     endif
2151     value = abs(Xdiff( n ))
```

2152 *Note: Zero code has been avoided (0 is sent as -0).*

### 2153 E.2.1.2 DPCM2 for 10–8–10 Coder

2154 **Xenco( n )** has the following format:

2155 **Xenco( n )** = “010 s xxxx”

2156 where,

2157 “010” is the code word  
2158 “s” is the **sign** bit  
2159 “xxxx” is the four bit **value** field

2160 The coder equation is described as follows:

```
2161     if (Xdiff( n ) < 0) then
2162         sign = 1
2163     else
2164         sign = 0
2165     endif
2166     value = (abs(Xdiff( n )) - 32) / 2
2167
```

### 2168 E.2.1.3 DPCM3 for 10–8–10 Coder

2169 **Xenco( n )** has the following format:

2170 **Xenco( n )** = “011 s xxxx”

2171 where,

2172 “011” is the code word  
2173 “s” is the **sign** bit  
2174 “xxxx” is the four bit **value** field
2175

2176 The coder equation is described as follows:

```
2177     if (Xdiff( n ) < 0) then
2178         sign = 1
2179     else
2180         sign = 0
2181     endif
2182     value = (abs(Xdiff( n )) - 64) / 4
```

#### 2183 E.2.1.4 PCM for 10–8–10 Coder

2184 **Xenco( n )** has the following format:

```
2185     Xenco( n ) = "1 xxxxxxx"
```

2186 where,

```
2187     "1" is the code word
2188     the sign bit is not used
2189     "xxxxxx" is the seven bit value field
```

2190 The coder equation is described as follows:

```
2191     value = Xorig( n ) / 8
```

#### 2192 E.2.2 Coder for 10–7–10 Data Compression

2193 The 10–7–10 coder offers 30% bit rate reduction with high image quality.

2194 Pixels without prediction are encoded using the following formula:

```
2195     Xenco( n ) = Xorig( n ) / 8
```

2196 To avoid a full-zero encoded value, the following check is performed:

```
2197     if (Xenco( n ) == 0) then
2198         Xenco( n ) = 1
```

2199 Pixels with prediction are encoded using the following formula:

```
2200     if (abs(Xdiff( n )) < 8) then
2201         use DPCM1
2202     else if (abs(Xdiff( n )) < 16) then
2203         use DPCM2
2204     else if (abs(Xdiff( n )) < 32) then
2205         use DPCM3
2206     else if (abs(Xdiff( n )) < 160) then
2207         use DPCM4
2208     else
2209         use PCM
2210     endif
```

2211   **E.2.2.1     DPCM1 for 10–7–10 Coder**

2212   **Xenco( n )** has the following format:

2213           **Xenco( n )** = “000 s xxx”

2214           where,

2215           “000” is the code word

2216           “s” is the **sign** bit

2217           “xxx” is the three bit **value** field

2218   The coder equation is described as follows:

```
2219        if (Xdiff( n ) <= 0) then
2220            sign = 1
2221        else
2222            sign = 0
2223        endif
2224        value = abs(Xdiff( n ))
```

2225   *Note: Zero code has been avoided (0 is sent as -0).*

2226   **E.2.2.2     DPCM2 for 10–7–10 Coder**

2227   **Xenco( n )** has the following format:

2228           **Xenco( n )** = “0010 s xx”

2229           where,

2230           “0010” is the code word

2231           “s” is the **sign** bit

2232           “xx” is the two bit **value** field

2233   The coder equation is described as follows:

```
2234        if (Xdiff( n ) < 0) then
2235            sign = 1
2236        else
2237            sign = 0
2238        endif
2239        value = (abs(Xdiff( n )) - 8) / 2
```

2240   **E.2.2.3     DPCM3 for 10–7–10 Coder**

2241   **Xenco( n )** has the following format:

2242           **Xenco( n )** = “0011 s xx”

2243           where,

2244           “0011” is the code word

2245           “s” is the **sign** bit

2246           “xx” is the two bit **value** field

2247 The coder equation is described as follows:

```
2248     if (Xdiff( n ) < 0) then
2249         sign = 1
2250     else
2251         sign = 0
2252     endif
2253     value = (abs(Xdiff( n )) - 16) / 4
```

#### 2254 E.2.2.4 DPCM4 for 10-7-10 Coder

2255 **Xenco( n )** has the following format:

2256 **Xenco( n )** = “01 s xxxx”

2257 where,

```
2258     “01” is the code word
2259     “s” is the sign bit
2260     “xxxx” is the four bit value field
```

2261 The coder equation is described as follows:

```
2262     if (Xdiff( n ) < 0) then
2263         sign = 1
2264     else
2265         sign = 0
2266     endif
2267     value = (abs(Xdiff( n )) - 32) / 8
```

#### 2268 E.2.2.5 PCM for 10-7-10 Coder

2269 **Xenco( n )** has the following format:

2270 **Xenco( n )** = “1 xxxxxx”

2271 where,

```
2272     “1” is the code word
2273     the sign bit is not used
2274     “xxxxxx” is the six bit value field
```

2275 The coder equation is described as follows:

2276 **value** = **Xorig( n )** / 16

### 2277 E.2.3 Coder for 10-6-10 Data Compression

2278 The 10-6-10 coder offers 40% bit rate reduction with acceptable image quality.

2279 Pixels without prediction are encoded using the following formula:

2280 **Xenco( n )** = **Xorig( n )** / 16

2281 To avoid a full-zero encoded value, the following check is performed:

```
2282     if (Xenco( n ) == 0) then
2283         Xenco( n ) = 1
2284     endif
```

2285 Pixels with prediction are encoded using the following formula:

```
2286     if (abs(Xdiff( n )) < 1) then
2287         use DPCM1
2288     else if (abs(Xdiff( n )) < 3) then
2289         use DPCM2
2290     else if (abs(Xdiff( n )) < 11) then
2291         use DPCM3
2292     else if (abs(Xdiff( n )) < 43) then
2293         use DPCM4
2294     else if (abs(Xdiff( n )) < 171) then
2295         use DPCM5
2296     else
2297         use PCM
2298     endif
```

#### 2299 **E.2.3.1 DPCM1 for 10–6–10 Coder**

2300 **Xenco( n )** has the following format:

2301     **Xenco( n )** = “00000 s”

2302     where,

2303         “00000” is the code word
2304         “s” is the **sign** bit
2305         the **value** field is not used

2306 The coder equation is described as follows:

2307     **sign** = 1

2308 *Note: Zero code has been avoided (0 is sent as -0).*

#### 2309 **E.2.3.2 DPCM2 for 10–6–10 Coder**

2310 **Xenco( n )** has the following format:

2311     **Xenco( n )** = “00001 s”

2312     where,

2313         “00001” is the code word
2314         “s” is the **sign** bit
2315         the **value** field is not used
2316

2317 The coder equation is described as follows:

```
2318     if (Xdiff( n ) < 0) then
2319         sign = 1
2320     else
2321         sign = 0
2322     endif
```

#### 2323 E.2.3.3 DPCM3 for 10-6-10 Coder

2324 **Xenco( n )** has the following format:

```
2325     Xenco( n ) = "0001 s x"
```

2326 where,

```
2327     "0001" is the code word
2328     "s" is the sign bit
2329     "x" is the one bit value field
```

2330 The coder equation is described as follows:

```
2331     if (Xdiff( n ) < 0) then
2332         sign = 1
2333     else
2334         sign = 0
2335         value = (abs(Xdiff( n )) - 3) / 4
2336     endif
```

#### 2337 E.2.3.4 DPCM4 for 10-6-10 Coder

2338 **Xenco( n )** has the following format:

```
2339     Xenco( n ) = "001 s xx"
```

2340 where,

```
2341     "001" is the code word
2342     "s" is the sign bit
2343     "xx" is the two bit value field
```

2344 The coder equation is described as follows:

```
2345     if (Xdiff( n ) < 0) then
2346         sign = 1
2347     else
2348         sign = 0
2349     endif
2350     value = (abs(Xdiff( n )) - 11) / 8
```

#### 2351 E.2.3.5 DPCM5 for 10-6-10 Coder

2352 **Xenco( n )** has the following format:

```
2353     Xenco( n ) = "01 s xxx"
```

2354 where,

2355 “01” is the code word  
2356 “s” is the **sign** bit  
2357 “xxx” is the three bit **value** field

2358 The coder equation is described as follows:

```
2359     if (Xdiff( n ) < 0) then
2360         sign = 1
2361     else
2362         sign = 0
2363     endif
2364     value = (abs(Xdiff( n )) - 43) / 16
```

#### 2365 E.2.3.6 PCM for 10–6–10 Coder

2366 **Xenco( n )** has the following format:

2367 **Xenco( n )** = “1 xxxx”

2368 where,

2369 “1” is the code word  
2370 the **sign** bit is not used  
2371 “xxxx” is the five bit **value** field

2372 The coder equation is described as follows:

2373 **value** = **Xorig( n )** / 32

#### 2374 E.2.4 Coder for 12–8–12 Data Compression

2375 The 12–8–12 coder offers 33% bit rate reduction with very high image quality.

2376 Pixels without prediction are encoded using the following formula:

2377 **Xenco( n )** = **Xorig( n )** / 16

2378 To avoid a full-zero encoded value, the following check is performed:

```
2379     if (Xenco( n ) == 0) then
2380         Xenco( n ) = 1
2381     endif
```

2382 Pixels with prediction are encoded using the following formula:

```
2383     if (abs(Xdiff( n )) < 8) then
2384         use DPCM1
2385     else if (abs(Xdiff( n )) < 40) then
2386         use DPCM2
2387     else if (abs(Xdiff( n )) < 104) then
2388         use DPCM3
2389     else if (abs(Xdiff( n )) < 232) then
2390         use DPCM4
```

```
2391     else if (abs(Xdiff( n )) < 360) then
2392         use DPCM5
2393     else
2394         use PCM
```

#### 2395 E.2.4.1 DPCM1 for 12–8–12 Coder

2396 **Xenco( n )** has the following format:

```
2397     Xenco( n ) = "0000 s xxx"
```

2398 where,

```
2399     "0000" is the code word
2400     "s" is the sign bit
2401     "xxx" is the three bit value field
```

2402 The coder equation is described as follows:

```
2403     if (Xdiff( n ) <= 0) then
2404         sign = 1
2405     else
2406         sign = 0
2407     endif
2408     value = abs(Xdiff( n ))
```

2409 Note: Zero code has been avoided (0 is sent as -0).

#### 2410 E.2.4.2 DPCM2 for 12–8–12 Coder

2411 **Xenco( n )** has the following format:

```
2412     Xenco( n ) = "011 s xxxx"
```

2413 where,

```
2414     "011" is the code word
2415     "s" is the sign bit
2416     "xxxx" is the four bit value field
```

2417 The coder equation is described as follows:

```
2418     if (Xdiff( n ) < 0) then
2419         sign = 1
2420     else
2421         sign = 0
2422     endif
2423     value = (abs(Xdiff( n )) - 8) / 2
```

#### 2424 E.2.4.3 DPCM3 for 12–8–12 Coder

2425 **Xenco( n )** has the following format:

```
2426     Xenco( n ) = "010 s xxxx"
```

2427 where,

2428 “010” is the code word  
2429 “s” is the **sign** bit  
2430 “xxxx” is the four bit **value** field

2431 The coder equation is described as follows:

```
2432     if (Xdiff( n ) < 0) then
2433         sign = 1
2434     else
2435         sign = 0
2436     endif
2437     value = (abs(Xdiff( n )) - 40) / 4
```

#### 2438 E.2.4.4 DPCM4 for 12–8–12 Coder

2439 **Xenco( n )** has the following format:

2440 **Xenco( n )** = “001 s xxxx”

2441 where,

2442 “001” is the code word  
2443 “s” is the **sign** bit  
2444 “xxxx” is the four bit **value** field

2445 The coder equation is described as follows:

```
2446     if (Xdiff( n ) < 0) then
2447         sign = 1
2448     else
2449         sign = 0
2450     endif
2451     value = (abs(Xdiff( n )) - 104) / 8
```

#### 2452 E.2.4.5 DPCM5 for 12–8–12 Coder

2453 **Xenco( n )** has the following format:

2454 **Xenco( n )** = “0001 s xxx”

2455 where,

2456 “0001” is the code word  
2457 “s” is the **sign** bit  
2458 “xxx” is the three bit **value** field

2459 The coder equation is described as follows:

```
2460     if (Xdiff( n ) < 0) then
2461         sign = 1
2462     else
2463         sign = 0
2464     endif
```

2465       **value** = (abs(**Xdiff( n )**) - 232) / 16

2466     **E.2.4.6     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           “xxxxxxxx” is the seven bit **value** field

2473     The coder equation is described as follows:

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

2475     **E.2.5     Coder for 12–7–12 Data Compression**

2476     The 12–7–12 coder offers 42% bit rate reduction with high image quality.

2477     Pixels without prediction are encoded using the following formula:

2478       **Xenco( n )** = **Xorig( n )** / 32

2479     To avoid a full-zero encoded value, the following check is performed:

2480       if (**Xenco( n )** == 0) then  
2481           **Xenco( n )** = 1  
2482       endif

2483     Pixels with prediction are encoded using the following formula:

2484       if (abs(**Xdiff( n )**) < 4) then  
2485           use **DPCM1**  
2486       else if (abs(**Xdiff( n )**) < 12) then  
2487           use **DPCM2**  
2488       else if (abs(**Xdiff( n )**) < 28) then  
2489           use **DPCM3**  
2490       else if (abs(**Xdiff( n )**) < 92) then  
2491           use **DPCM4**  
2492       else if (abs(**Xdiff( n )**) < 220) then  
2493           use **DPCM5**  
2494       else if (abs(**Xdiff( n )**) < 348) then  
2495           use **DPCM6**  
2496       else  
2497           use **PCM**  
2498       endif

2499     **E.2.5.1     DPCM1 for 12–7–12 Coder**

2500     **Xenco( n )** has the following format:

2501       **Xenco( n )** = “0000 s xx”

2502 where,

2503 “0000” is the code word  
2504 “s” is the **sign** bit  
2505 “xx” is the two bit **value** field

2506 The coder equation is described as follows:

```
2507     if (Xdiff( n ) <= 0) then
2508         sign = 1
2509     else
2510         sign = 0
2511     endif
2512     value = abs(Xdiff( n ))
```

2513 *Note: Zero code has been avoided (0 is sent as -0).*

#### 2514 E.2.5.2 DPCM2 for 12–7–12 Coder

2515 **Xenco( n )** has the following format:

2516 **Xenco( n )** = “0001 s xx”

2517 where,

2518 “0001” is the code word  
2519 “s” is the **sign** bit  
2520 “xx” is the two bit **value** field

2521 The coder equation is described as follows:

```
2522     if (Xdiff( n ) < 0) then
2523         sign = 1
2524     else
2525         sign = 0
2526     endif
2527     value = (abs(Xdiff( n )) - 4) / 2
```

#### 2528 E.2.5.3 DPCM3 for 12–7–12 Coder

2529 **Xenco( n )** has the following format:

2530 **Xenco( n )** = “0010 s xx”

2531 where,

2532 “0010” is the code word  
2533 “s” is the **sign** bit  
2534 “xx” is the two bit **value** field

2535 The coder equation is described as follows:

```
2536     if (Xdiff( n ) < 0) then
2537         sign = 1
2538     else
```

2539           **sign** = 0  
2540           **endif**  
2541           **value** = (abs(**Xdiff( n )**) - 12) / 4

2542       **E.2.5.4 DPCM4 for 12–7–12 Coder**

2543       **Xenco( n )** has the following format:

2544           **Xenco( n )** = “010 s xxx”

2545           where,

2546           “010” is the code word  
2547           “s” is the **sign** bit  
2548           “xxx” is the three bit **value** field

2549       The coder equation is described as follows:

2550           **if (Xdiff( n ) < 0) then**  
2551              **sign** = 1  
2552           **else**  
2553              **sign** = 0  
2554           **endif**  
2555           **value** = (abs(**Xdiff( n )**) - 28) / 8

2556       **E.2.5.5 DPCM5 for 12–7–12 Coder**

2557       **Xenco( n )** has the following format:

2558           **Xenco( n )** = “011 s xxx”

2559           where,

2560           “011” is the code word  
2561           “s” is the **sign** bit  
2562           “xxx” is the three bit **value** field

2563       The coder equation is described as follows:

2564           **if (Xdiff( n ) < 0) then**  
2565              **sign** = 1  
2566           **else**  
2567              **sign** = 0  
2568           **endif**  
2569           **value** = (abs(**Xdiff( n )**) - 92) / 16

2570       **E.2.5.6 DPCM6 for 12–7–12 Coder**

2571       **Xenco( n )** has the following format:

2572           **Xenco( n )** = “0011 s xx”

2573 where,

2574 “0011” is the code word  
2575 “s” is the **sign** bit  
2576 “xx” is the two bit **value** field

2577 The coder equation is described as follows:

```
2578     if (Xdiff( n ) < 0) then
2579         sign = 1
2580     else
2581         sign = 0
2582     endif
2583     value = (abs(Xdiff( n )) - 220) / 32
```

#### 2584 E.2.5.7 PCM for 12–7–12 Coder

2585 **Xenco( n )** has the following format:

2586 **Xenco( n )** = “1 xxxxxx”

2587 where,

2588 “1” is the code word  
2589 the **sign** bit is not used  
2590 “xxxxxx” is the six bit **value** field

2591 The coder equation is described as follows:

2592 **value** = **Xorig( n )** / 64

#### 2593 E.2.6 Coder for 12–6–12 Data Compression

2594 The 12–6–12 coder offers 50% bit rate reduction with acceptable image quality.

2595 Pixels without prediction are encoded using the following formula:

2596 **Xenco( n )** = **Xorig( n )** / 64

2597 To avoid a full-zero encoded value, the following check is performed:

```
2598     if (Xenco( n ) == 0) then
2599         Xenco( n ) = 1
2600     endif
```

2601 Pixels with prediction are encoded using the following formula:

```
2602     if (abs(Xdiff( n )) < 2) then
2603         use DPCM1
2604     else if (abs(Xdiff( n )) < 10) then
2605         use DPCM3
2606     else if (abs(Xdiff( n )) < 42) then
2607         use DPCM4
2608     else if (abs(Xdiff( n )) < 74) then
```

```
2609      use DPCM5
2610      else if (abs(Xdiff( n )) < 202) then
2611          use DPCM6
2612          else if (abs(Xdiff( n )) < 330) then
2613              use DPCM7
2614          else
2615              use PCM
2616          endif
```

2617 *Note: DPCM2 is not used.*

#### 2618 E.2.6.1 DPCM1 for 12–6–12 Coder

2619 **Xenco( n )** has the following format:

2620     **Xenco( n )** = “0000 s x”

2621     where,

2622         “0000” is the code word
2623         “s” is the **sign** bit
2624         “x” is the one bit **value** field

2625     The coder equation is described as follows:

```
2626      if (Xdiff( n ) <= 0) then
2627          sign = 1
2628      else
2629          sign = 0
2630      endif
2631      value = abs(Xdiff( n ))
```

2632 *Note: Zero code has been avoided (0 is sent as -0).*

#### 2633 E.2.6.2 DPCM3 for 12–6–12 Coder

2634 **Xenco( n )** has the following format:

2635     **Xenco( n )** = “0001 s x”

2636     where,

2637         “0001” is the code word
2638         “s” is the **sign** bit
2639         “x” is the one bit **value** field

2640     The coder equation is described as follows:

```
2641      if (Xdiff( n ) < 0) then
2642          sign = 1
2643      else
2644          sign = 0
2645      endif
2646      value = (abs(Xdiff( n )) - 2) / 4
```

2647   **E.2.6.3     DPCM4 for 12–6–12 Coder**

2648   **Xenco( n )** has the following format:

2649           **Xenco( n )** = “010 s xx”

2650           where,

2651           “010” is the code word

2652           “s” is the **sign** bit

2653           “xx” is the two bit **value** field

2654   The coder equation is described as follows:

```
2655           if (Xdiff( n ) < 0) then
2656              sign = 1
2657           else
2658              sign = 0
2659           endif
2660           value = (abs(Xdiff( n )) - 10) / 8
```

2661   **E.2.6.4     DPCM5 for 12–6–12 Coder**

2662   **Xenco( n )** has the following format:

2663           **Xenco( n )** = “0010 s x”

2664           where,

2665           “0010” is the code word

2666           “s” is the **sign** bit

2667           “x” is the one bit **value** field

2668   The coder equation is described as follows:

```
2669           if (Xdiff( n ) < 0) then
2670              sign = 1
2671           else
2672              sign = 0
2673           endif
2674           value = (abs(Xdiff( n )) - 42) / 16
```

2675   **E.2.6.5     DPCM6 for 12–6–12 Coder**

2676   **Xenco( n )** has the following format:

2677           **Xenco( n )** = “011 s xx”

2678           where,

2679           “011” is the code word

2680           “s” is the **sign** bit

2681           “xx” is the two bit **value** field

2682 The coder equation is described as follows:

```
2683     if (Xdiff( n ) < 0) then
2684         sign = 1
2685     else
2686         sign = 0
2687     endif
2688     value = (abs(Xdiff( n )) - 74) / 32
```

#### 2689 E.2.6.6 DPCM7 for 12-6-12 Coder

2690 **Xenco( n )** has the following format:

2691       **Xenco( n )** = “0011 s x”

2692       where,

2693       “0011” is the code word
2694       “s” is the **sign** bit
2695       “x” is the one bit **value** field

2696 The coder equation is described as follows:

```
2697     if (Xdiff( n ) < 0) then
2698         sign = 1
2699     else
2700         sign = 0
2701     endif
2702     value = (abs(Xdiff( n )) - 202) / 64
```

#### 2703 E.2.6.7 PCM for 12-6-12 Coder

2704 **Xenco( n )** has the following format:

2705       **Xenco( n )** = “1 xxxx”

2706       where,

2707       “1” is the code word
2708       the **sign** bit is not used
2709       “xxxx” is the five bit **value** field

2710 The coder equation is described as follows:

2711       **value** = **Xorig( n )** / 128

### 2712 E.3 Decoders

2713 There are six different decoders available, one for each data compression scheme.

2714 For all decoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
2715 for predicted pixels.

2716   **E.3.1      Decoder for 10–8–10 Data Compression**

2717   Pixels without prediction are decoded using the following formula:

2718            $Xdeco(n) = 4 * Xenco(n) + 2$

2719   Pixels with prediction are decoded using the following formula:

```
2720       if (Xenco(n) & 0xc0 == 0x00) then
2721           use DPCM1
2722       else if (Xenco(n) & 0xe0 == 0x40) then
2723           use DPCM2
2724       else if (Xenco(n) & 0xe0 == 0x60) then
2725           use DPCM3
2726       else
2727           use PCM
2728       endif
```

2730   **E.3.1.1    DPCM1 for 10–8–10 Decoder**

2731   **Xenco( n )** has the following format:

2732           **Xenco( n )** = “00 s xxxx”

2733       where,

```
2734       “00” is the code word
2735       “s” is the sign bit
2736       “xxxx” is the five bit value field
```

2737   The decoder equation is described as follows:

```
2738       sign = Xenco( n ) & 0x20
2739       value = Xenco( n ) & 0x1f
2740       if (sign > 0) then
2741           Xdeco( n ) = Xpred( n ) - value
2742       else
2743           Xdeco( n ) = Xpred( n ) + value
2744       endif
```

2745   **E.3.1.2    DPCM2 for 10–8–10 Decoder**

2746   **Xenco( n )** has the following format:

2747           **Xenco( n )** = “010 s xxxx”

2748       where,

```
2749       “010” is the code word
2750       “s” is the sign bit
2751       “xxxx” is the four bit value field
```

2752 The decoder equation is described as follows:

```
2753     sign = Xenco( n ) & 0x10
2754     value = 2 * (Xenco( n ) & 0xf) + 32
2755     if (sign > 0) then
2756         Xdeco( n ) = Xpred( n ) - value
2757     else
2758         Xdeco( n ) = Xpred( n ) + value
2759     endif
```

2760 **E.3.1.3 DPCM3 for 10–8–10 Decoder**

2761 **Xenco( n )** has the following format:

2762 **Xenco( n )** = “011 s xxxx”

2763 where,

2764 “011” is the code word
2765 “s” is the **sign** bit
2766 “xxxx” is the four bit **value** field

2767 The decoder equation is described as follows:

```
2768     sign = Xenco( n ) & 0x10
2769     value = 4 * (Xenco( n ) & 0xf) + 64 + 1
2770     if (sign > 0) then
2771         Xdeco( n ) = Xpred( n ) - value
2772         if (Xdeco( n ) < 0) then
2773             Xdeco( n ) = 0
2774         endif
2775     else
2776         Xdeco( n ) = Xpred( n ) + value
2777         if (Xdeco( n ) > 1023) then
2778             Xdeco( n ) = 1023
2779         endif
2780     endif
```

2781 **E.3.1.4 PCM for 10–8–10 Decoder**

2782 **Xenco( n )** has the following format:

2783 **Xenco( n )** = “1 xxxxxxx”

2784 where,

2785 “1” is the code word
2786 the **sign** bit is not used
2787 “xxxxxxxx” is the seven bit **value** field

2788 The codec equation is described as follows:

```
2789     value = 8 * (Xenco( n ) & 0x7f)
2790     if (value > Xpred( n )) then
2791         Xdeco( n ) = value + 3
```

```
2792     endif
2793     else
2794         Xdeco( n ) = value + 4
2795     endif
```

### 2796 E.3.2 Decoder for 10–7–10 Data Compression

2797 Pixels without prediction are decoded using the following formula:

```
2798     Xdeco( n ) = 8 * Xenco( n ) + 4
```

2799 Pixels with prediction are decoded using the following formula:

```
2800     if (Xenco( n ) & 0x70 == 0x00) then
2801         use DPCM1
2802     else if (Xenco( n ) & 0x78 == 0x10) then
2803         use DPCM2
2804     else if (Xenco( n ) & 0x78 == 0x18) then
2805         use DPCM3
2806     else if (Xenco( n ) & 0x60 == 0x20) then
2807         use DPCM4
2808     else
2809         use PCM
2810     endif
```

#### 2811 E.3.2.1 DPCM1 for 10–7–10 Decoder

2812 **Xenco( n )** has the following format:

```
2813     Xenco( n ) = "000 s xxx"
```

2814 where,

```
2815     "000" is the code word
2816     "s" is the sign bit
2817     "xxx" is the three bit value field
```

2818 The codec equation is described as follows:

```
2819     sign = Xenco( n ) & 0x8
2820     value = Xenco( n ) & 0x7
2821     if (sign > 0) then
2822         Xdeco( n ) = Xpred( n ) - value
2823     else
2824         Xdeco( n ) = Xpred( n ) + value
2825     endif
```

#### 2826 E.3.2.2 DPCM2 for 10–7–10 Decoder

2827 **Xenco( n )** has the following format:

```
2828     Xenco( n ) = "0010 s xx"
```

2829 where,

2830 “0010” is the code word  
2831 “s” is the **sign** bit  
2832 “xx” is the two bit **value** field

2833 The codec equation is described as follows:

```
2834 sign = Xenco( n ) & 0x4
2835 value = 2 * (Xenco( n ) & 0x3) + 8
2836 if (sign > 0) then
2837     Xdeco( n ) = Xpred( n ) - value
2838 else
2839     Xdeco( n ) = Xpred( n ) + value
2840 endif
```

#### 2841 E.3.2.3 DPCM3 for 10-7-10 Decoder

2842 **Xenco( n )** has the following format:

2843 **Xenco( n )** = “0011 s xx”

2844 where,

2845 “0011” is the code word  
2846 “s” is the **sign** bit  
2847 “xx” is the two bit **value** field

2848 The codec equation is described as follows:

```
2849 sign = Xenco( n ) & 0x4
2850 value = 4 * (Xenco( n ) & 0x3) + 16 + 1
2851 if (sign > 0) then
2852     Xdeco( n ) = Xpred( n ) - value
2853     if (Xdeco( n ) < 0) then
2854         Xdeco( n ) = 0
2855     endif
2856 else
2857     Xdeco( n ) = Xpred( n ) + value
2858     if (Xdeco( n ) > 1023) then
2859         Xdeco( n ) = 1023
2860     endif
2861 endif
```

#### 2862 E.3.2.4 DPCM4 for 10-7-10 Decoder

2863 **Xenco( n )** has the following format:

2864 **Xenco( n )** = “01 s xxxx”

2865 where,

2866 “01” is the code word  
2867 “s” is the **sign** bit  
2868 “xxxx” is the four bit **value** field

2869 The codec equation is described as follows:

```
2870     sign = Xenco( n ) & 0x10
2871     value = 8 * (Xenco( n ) & 0xf) + 32 + 3
2872     if (sign > 0) then
2873         Xdeco( n ) = Xpred( n ) - value
2874         if (Xdeco( n ) < 0) then
2875             Xdeco( n ) = 0
2876         endif
2877     else
2878         Xdeco( n ) = Xpred( n ) + value
2879         if (Xdeco( n ) > 1023) then
2880             Xdeco( n ) = 1023
2881         endif
2882     endif
```

#### 2883 E.3.2.5 PCM for 10–7–10 Decoder

2884 Xenco( n ) has the following format:

2885 Xenco( n ) = “1 xxxxx”

2886 where,

```
2887     “1” is the code word
2888     the sign bit is not used
2889     “xxxxx” is the six bit value field
```

2890 The codec equation is described as follows:

```
2891     value = 16 * (Xenco( n ) & 0x3f)
2892     if (value > Xpred( n )) then
2893         Xdeco( n ) = value + 7
2894     else
2895         Xdeco( n ) = value + 8
2896     endif
```

#### 2897 E.3.3 Decoder for 10–6–10 Data Compression

2898 Pixels without prediction are decoded using the following formula:

2899 Xdeco( n ) = 16 \* Xenco( n ) + 8

2900 Pixels with prediction are decoded using the following formula:

```
2901     if (Xenco( n ) & 0x3e == 0x00) then
2902         use DPCM1
2903     else if (Xenco( n ) & 0x3e == 0x02) then
2904         use DPCM2
2905     else if (Xenco( n ) & 0x3c == 0x04) then
2906         use DPCM3
2907     else if (Xenco( n ) & 0x38 == 0x08) then
2908         use DPCM4
2909     else if (Xenco( n ) & 0x30 == 0x10) then
2910         use DPCM5
```

2911           else  
2912            use PCM  
2913          endif

2914      **E.3.3.1 DPCM1 for 10–6–10 Decoder**

2915      **Xenco( n )** has the following format:

2916           **Xenco( n )** = “00000 s”

2917           where,

2918           “00000” is the code word  
2919           “s” is the **sign** bit  
2920           the **value** field is not used

2921      The codec equation is described as follows:

2922           **Xdeco( n )** = **Xpred( n )**

2923      **E.3.3.2 DPCM2 for 10–6–10 Decoder**

2924      **Xenco( n )** has the following format:

2925           **Xenco( n )** = “00001 s”

2926           where,

2927           “00001” is the code word  
2928           “s” is the **sign** bit  
2929           the **value** field is not used

2930      The codec equation is described as follows:

2931           **sign** = **Xenco( n )** & 0x1  
2932           **value** = 1  
2933           if (**sign** > 0) then  
2934             **Xdeco( n )** = **Xpred( n )** - **value**  
2935           else  
2936             **Xdeco( n )** = **Xpred( n )** + **value**  
2937          endif

2938      **E.3.3.3 DPCM3 for 10–6–10 Decoder**

2939      **Xenco( n )** has the following format:

2940           **Xenco( n )** = “0001 s x”

2941           where,

2942           “0001” is the code word  
2943           “s” is the **sign** bit  
2944           “x” is the one bit **value** field

2945 The codec equation is described as follows:

```
2946     sign = Xenco( n ) & 0x2
2947     value = 4 * (Xenco( n ) & 0x1) + 3 + 1
2948     if (sign > 0) then
2949         Xdeco( n ) = Xpred( n ) - value
2950         if (Xdeco( n ) < 0) then
2951             Xdeco( n ) = 0
2952         endif
2953     else
2954         Xdeco( n ) = Xpred( n ) + value
2955         if (Xdeco( n ) > 1023) then
2956             Xdeco( n ) = 1023
2957         endif
2958     endif
```

#### 2959 E.3.3.4 DPCM4 for 10-6-10 Decoder

2960 Xenco( n ) has the following format:

2961 Xenco( n ) = "001 s xx"

2962 where,

2963 "001" is the code word
2964 "s" is the **sign** bit
2965 "xx" is the two bit **value** field

2966 The codec equation is described as follows:

```
2967     sign = Xenco( n ) & 0x4
2968     value = 8 * (Xenco( n ) & 0x3) + 11 + 3
2969     if (sign > 0) then
2970         Xdeco( n ) = Xpred( n ) - value
2971         if (Xdeco( n ) < 0) then
2972             Xdeco( n ) = 0
2973         endif
2974     else
2975         Xdeco( n ) = Xpred( n ) + value
2976         if (Xdeco( n ) > 1023) then
2977             Xdeco( n ) = 1023
2978         endif
2979     endif
```

#### 2980 E.3.3.5 DPCM5 for 10-6-10 Decoder

2981 Xenco( n ) has the following format:

2982 Xenco( n ) = "01 s xxx"

2983 where,

2984 "01" is the code word
2985 "s" is the **sign** bit
2986 "xxx" is the three bit **value** field

2987 The codec equation is described as follows:

```
2988     sign = Xenco( n ) & 0x8
2989     value = 16 * (Xenco( n ) & 0x7) + 43 + 7
2990     if (sign > 0) then
2991         Xdeco( n ) = Xpred( n ) - value
2992         if (Xdeco( n ) < 0) then
2993             Xdeco( n ) = 0
2994         endif
2995     else
2996         Xdeco( n ) = Xpred( n ) + value
2997         if (Xdeco( n ) > 1023) then
2998             Xdeco( n ) = 1023
2999         endif
3000     endif
```

### 3001 E.3.3.6 PCM for 10–6–10 Decoder

3002 Xenco( n ) has the following format:

3003 Xenco( n ) = “1 xxxx”

3004 where,

3005 “1” is the code word
3006 the sign bit is not used
3007 “xxxx” is the five bit value field

3008 The codec equation is described as follows:

```
3009     value = 32 * (Xenco( n ) & 0x1f)
3010     if (value > Xpred( n )) then
3011         Xdeco( n ) = value + 15
3012     else
3013         Xdeco( n ) = value + 16
3014     endif
```

### 3015 E.3.4 Decoder for 12–8–12 Data Compression

3016 Pixels without prediction are decoded using the following formula:

3017 Xdeco( n ) = 16 \* Xenco( n ) + 8

3018 Pixels with prediction are decoded using the following formula:

```
3019     if (Xenco( n ) & 0xf0 == 0x00) then
3020         use DPCM1
3021     else if (Xenco( n ) & 0xe0 == 0x60) then
3022         use DPCM2
3023     else if (Xenco( n ) & 0xe0 == 0x40) then
3024         use DPCM3
3025     else if (Xenco( n ) & 0xe0 == 0x20) then
3026         use DPCM4
3027     else if (Xenco( n ) & 0xf0 == 0x10) then
3028         use DPCM5
```

3029           else  
3030            use PCM  
3031          endif

3032      **E.3.4.1     DPCM1 for 12–8–12 Decoder**

3033    **Xenco( n )** has the following format:

3034       **Xenco( n )** = “0000 s xxx”

3035       where,

3036       “0000” is the code word  
3037       “s” is the **sign** bit  
3038       “xxx” is the three bit **value** field

3039    The codec equation is described as follows:

```
3040       sign = Xenco( n ) & 0x8  
3041       value = Xenco( n ) & 0x7  
3042       if (sign > 0) then  
3043           Xdeco( n ) = Xpred( n ) - value  
3044       else  
3045           Xdeco( n ) = Xpred( n ) + value  
3046       endif
```

3047      **E.3.4.2     DPCM2 for 12–8–12 Decoder**

3048    **Xenco( n )** has the following format:

3049       **Xenco( n )** = “011 s xxxx”

3050       where,

3051       “011” is the code word  
3052       “s” is the **sign** bit  
3053       “xxxx” is the four bit **value** field

3054    The codec equation is described as follows:

```
3055       sign = Xenco( n ) & 0x10  
3056       value = 2 * (Xenco( n ) & 0xf) + 8  
3057       if (sign > 0) then  
3058           Xdeco( n ) = Xpred( n ) - value  
3059       else  
3060           Xdeco( n ) = Xpred( n ) + value  
3061       endif
```

3062      **E.3.4.3     DPCM3 for 12–8–12 Decoder**

3063    **Xenco( n )** has the following format:

3064       **Xenco( n )** = “010 s xxxx”

3065 where,

3066 “010” is the code word  
3067 “s” is the **sign** bit  
3068 “xxxx” is the four bit **value** field

3069 The codec equation is described as follows:

```
3070     sign = Xenco( n ) & 0x10
3071     value = 4 * (Xenco( n ) & 0xf) + 40 + 1
3072     if (sign > 0) then
3073         Xdeco( n ) = Xpred( n ) - value
3074         if (Xdeco( n ) < 0) then
3075             Xdeco( n ) = 0
3076         endif
3077     else
3078         Xdeco( n ) = Xpred( n ) + value
3079         if (Xdeco( n ) > 4095) then
3080             Xdeco( n ) = 4095
3081         endif
3082     endif
```

3083 **E.3.4.4 DPCM4 for 12–8–12 Decoder**

3084 **Xenco( n )** has the following format:

3085 **Xenco( n )** = “001 s xxxx”

3086 where,

3087 “001” is the code word  
3088 “s” is the **sign** bit  
3089 “xxxx” is the four bit **value** field

3090 The codec equation is described as follows:

```
3091     sign = Xenco( n ) & 0x10
3092     value = 8 * (Xenco( n ) & 0xf) + 104 + 3
3093     if (sign > 0) then
3094         Xdeco( n ) = Xpred( n ) - value
3095         if (Xdeco( n ) < 0) then
3096             Xdeco( n ) = 0
3097         endif
3098     else
3099         Xdeco( n ) = Xpred( n ) + value
3100         if (Xdeco( n ) > 4095)
3101             Xdeco( n ) = 4095
3102         endif
3103     endif
```

3104 **E.3.4.5 DPCM5 for 12–8–12 Decoder**

3105 **Xenco( n )** has the following format:

3106 **Xenco( n )** = “0001 s xxx”

3107 where,

3108 “0001” is the code word  
3109 “s” is the **sign** bit  
3110 “xxx” is the three bit **value** field

3111 The codec equation is described as follows:

```
3112     sign = Xenco( n ) & 0x8
3113     value = 16 * (Xenco( n ) & 0x7) + 232 + 7
3114     if (sign > 0) then
3115         Xdeco( n ) = Xpred( n ) - value
3116         if (Xdeco( n ) < 0) then
3117             Xdeco( n ) = 0
3118         endif
3119     else
3120         Xdeco( n ) = Xpred( n ) + value
3121         if (Xdeco( n ) > 4095) then
3122             Xdeco( n ) = 4095
3123         endif
3124     endif
```

#### 3125 E.3.4.6 PCM for 12–8–12 Decoder

3126 **Xenco( n )** has the following format:

3127 **Xenco( n )** = “1 xxxxxxx”

3128 where,

3129 “1” is the code word  
3130 the **sign** bit is not used  
3131 “xxxxxxxx” is the seven bit **value** field

3132 The codec equation is described as follows:

```
3133     value = 32 * (Xenco( n ) & 0x7f)
3134     if (value > Xpred( n )) then
3135         Xdeco( n ) = value + 15
3136     else
3137         Xdeco( n ) = value + 16
3138     endif
```

#### 3139 E.3.5 Decoder for 12–7–12 Data Compression

3140 Pixels without prediction are decoded using the following formula:

3141 **Xdeco( n )** = 32 \* **Xenco( n )** + 16

3142 Pixels with prediction are decoded using the following formula:

```
3143     if (Xenco( n ) & 0x78 == 0x00) then
3144         use DPCM1
3145     else if (Xenco( n ) & 0x78 == 0x08) then
3146         use DPCM2
```

```
3147     else if (Xenco( n ) & 0x78 == 0x10) then
3148         use DPCM3
3149     else if (Xenco( n ) & 0x70 == 0x20) then
3150         use DPCM4
3151     else if (Xenco( n ) & 0x70 == 0x30) then
3152         use DPCM5
3153     else if (Xenco( n ) & 0x78 == 0x18) then
3154         use DPCM6
3155     else
3156         use PCM
3157     endif
```

3158 DPCM1 for 12-7-12 Decoder

3159 **Xenco( n )** has the following format:

3160 **Xenco( n )** = “0000 s xx”

3161 where,

```
3162     “0000” is the code word
3163     “s” is the sign bit
3164     “xx” is the two bit value field
```

3165 The codec equation is described as follows:

```
3166     sign = Xenco( n ) & 0x4
3167     value = Xenco( n ) & 0x3
3168     if (sign > 0) then
3169         Xdeco( n ) = Xpred( n ) - value
3170     else
3171         Xdeco( n ) = Xpred( n ) + value
3172     endif
```

### 3173 E.3.5.1 DPCM2 for 12-7-12 Decoder

3174 **Xenco( n )** has the following format:

3175 **Xenco( n )** = “0001 s xx”

3176 where,

```
3177     “0001” is the code word
3178     “s” is the sign bit
3179     “xx” is the two bit value field
```

3180 The codec equation is described as follows:

```
3181     sign = Xenco( n ) & 0x4
3182     value = 2 * (Xenco( n ) & 0x3) + 4
3183     if (sign > 0) then
3184         Xdeco( n ) = Xpred( n ) - value
3185     else
3186         Xdeco( n ) = Xpred( n ) + value
3187     endif
```

3188   **E.3.5.2     DPCM3 for 12–7–12 Decoder**

3189   **Xenco( n )** has the following format:

3190           **Xenco( n )** = “0010 s xx”

3191           where,

3192           “0010” is the code word

3193           “s” is the **sign** bit

3194           “xx” is the two bit **value** field

3195   The codec equation is described as follows:

```
3196       sign = Xenco( n ) & 0x4
3197       value = 4 * (Xenco( n ) & 0x3) + 12 + 1
3198       if (sign > 0) then
3199           Xdeco( n ) = Xpred( n ) - value
3200           if (Xdeco( n ) < 0) then
3201               Xdeco( n ) = 0
3202           endif
3203       else
3204           Xdeco( n ) = Xpred( n ) + value
3205           if (Xdeco( n ) > 4095) then
3206               Xdeco( n ) = 4095
3207           endif
3208       endif
```

3209   **E.3.5.3     DPCM4 for 12–7–12 Decoder**

3210   **Xenco( n )** has the following format:

3211           **Xenco( n )** = “010 s xxx”

3212           where,

3213           “010” is the code word

3214           “s” is the **sign** bit

3215           “xxx” is the three bit **value** field

3216   The codec equation is described as follows:

```
3217       sign = Xenco( n ) & 0x8
3218       value = 8 * (Xenco( n ) & 0x7) + 28 + 3
3219       if (sign > 0) then
3220           Xdeco( n ) = Xpred( n ) - value
3221           if (Xdeco( n ) < 0) then
3222               Xdeco( n ) = 0
3223           endif
3224       else
3225           Xdeco( n ) = Xpred( n ) + value
3226           if (Xdeco( n ) > 4095) then
3227               Xdeco( n ) = 4095
3228           endif
3229       endif
```

3230   **E.3.5.4     DPCM5 for 12–7–12 Decoder**

3231   **Xenco( n )** has the following format:

3232           **Xenco( n )** = “011 s xxx”

3233           where,

3234           “011” is the code word

3235           “s” is the **sign** bit

3236           “xxx” is the three bit **value** field

3237   The codec equation is described as follows:

```
3238       sign = Xenco( n ) & 0x8
3239       value = 16 * (Xenco( n ) & 0x7) + 92 + 7
3240       if (sign > 0) then
3241           Xdeco( n ) = Xpred( n ) - value
3242           if (Xdeco( n ) < 0) then
3243               Xdeco( n ) = 0
3244           endif
3245       else
3246           Xdeco( n ) = Xpred( n ) + value
3247           if (Xdeco( n ) > 4095) then
3248               Xdeco( n ) = 4095
3249           endif
3250       endif
```

3251   **E.3.5.5     DPCM6 for 12–7–12 Decoder**

3252   **Xenco( n )** has the following format:

3253           **Xenco( n )** = “0011 s xx”

3254           where,

3255           “0011” is the code word

3256           “s” is the **sign** bit

3257           “xx” is the two bit **value** field

3258   The codec equation is described as follows:

```
3259       sign = Xenco( n ) & 0x4
3260       value = 32 * (Xenco( n ) & 0x3) + 220 + 15
3261       if (sign > 0) then
3262           Xdeco( n ) = Xpred( n ) - value
3263           if (Xdeco( n ) < 0) then
3264               Xdeco( n ) = 0
3265           endif
3266       else
3267           Xdeco( n ) = Xpred( n ) + value
3268           if (Xdeco( n ) > 4095) then
3269               Xdeco( n ) = 4095
3270           endif
3271       endif
```

3272   **E.3.5.6   PCM for 12–7–12 Decoder**

3273   **Xenco( n )** has the following format:

3274           **Xenco( n )** = “1 xxxxxx”

3275           where,

3276           “1” is the code word  
3277           the **sign** bit is not used  
3278           “xxxxxx” is the six bit **value** field

3279   The codec equation is described as follows:

3280           **value** = 64 \* (**Xenco( n )** & 0x3f)  
3281           if (**value** > **Xpred( n )**) then  
3282              **Xdeco( n )** = **value** + 31  
3283           else  
3284              **Xdeco( n )** = **value** + 32  
3285           endif

3286   **E.3.6   Decoder for 12–6–12 Data Compression**

3287   Pixels without prediction are decoded using the following formula:

3288           **Xdeco( n )** = 64 \* **Xenco( n )** + 32

3289   Pixels with prediction are decoded using the following formula:

3290           if (**Xenco( n )** & 0x3c == 0x00) then  
3291              use **DPCM1**  
3292           else if (**Xenco( n )** & 0x3c == 0x04) then  
3293              use **DPCM3**  
3294           else if (**Xenco( n )** & 0x38 == 0x10) then  
3295              use **DPCM4**  
3296           else if (**Xenco( n )** & 0x3c == 0x08) then  
3297              use **DPCM5**  
3298           else if (**Xenco( n )** & 0x38 == 0x18) then  
3299              use **DPCM6**  
3300           else if (**Xenco( n )** & 0x3c == 0x0c) then  
3301              use **DPCM7**  
3302           else  
3303              use **PCM**  
3304           endif

3305   Note: **DPCM2** is not used.

3306   **E.3.6.1   DPCM1 for 12–6–12 Decoder**

3307   **Xenco( n )** has the following format:

3308           **Xenco( n )** = “0000 s x”

3309 where,

3310 “0000” is the code word  
3311 “s” is the **sign** bit  
3312 “x” is the one bit **value** field

3313 The codec equation is described as follows:

```
3314     sign = Xenco( n ) & 0x2
3315     value = Xenco( n ) & 0x1
3316     if (sign > 0) then
3317         Xdeco( n ) = Xpred( n ) - value
3318     else
3319         Xdeco( n ) = Xpred( n ) + value
3320     endif
```

### 3321 E.3.6.2 DPCM3 for 12–6–12 Decoder

3322 **Xenco( n )** has the following format:

3323 **Xenco( n )** = “0001 s x”

3324 where,

3325 “0001” is the code word  
3326 “s” is the **sign** bit  
3327 “x” is the one bit **value** field

3328 The codec equation is described as follows:

```
3329     sign = Xenco( n ) & 0x2
3330     value = 4 * (Xenco( n ) & 0x1) + 2 + 1
3331     if (sign > 0) then
3332         Xdeco( n ) = Xpred( n ) - value
3333         if (Xdeco( n ) < 0) then
3334             Xdeco( n ) = 0
3335         endif
3336     else
3337         Xdeco( n ) = Xpred( n ) + value
3338         if (Xdeco( n ) > 4095) then
3339             Xdeco( n ) = 4095
3340         endif
3341     endif
```

### 3342 E.3.6.3 DPCM4 for 12–6–12 Decoder

3343 **Xenco( n )** has the following format:

3344 **Xenco( n )** = “010 s xx”

3345 where,

3346 “010” is the code word  
3347 “s” is the **sign** bit  
3348 “xx” is the two bit **value** field

3349 The codec equation is described as follows:

```
3350     sign = Xenco( n ) & 0x4
3351     value = 8 * (Xenco( n ) & 0x3) + 10 + 3
3352     if (sign > 0) then
3353         Xdeco( n ) = Xpred( n ) - value
3354         if (Xdeco( n ) < 0) then
3355             Xdeco( n ) = 0
3356         endif
3357     else
3358         Xdeco( n ) = Xpred( n ) + value
3359         if (Xdeco( n ) > 4095) then
3360             Xdeco( n ) = 4095
3361         endif
3362     endif
```

#### 3363 E.3.6.4 DPCM5 for 12–6–12 Decoder

3364 Xenco( n ) has the following format:

3365 Xenco( n ) = “0010 s x”

3366 where,

3367 “0010” is the code word
3368 “s” is the **sign** bit
3369 “x” is the one bit **value** field

3370 The codec equation is described as follows:

```
3371     sign = Xenco( n ) & 0x2
3372     value = 16 * (Xenco( n ) & 0x1) + 42 + 7
3373     if (sign > 0) then
3374         Xdeco( n ) = Xpred( n ) - value
3375         if (Xdeco( n ) < 0) then
3376             Xdeco( n ) = 0
3377         endif
3378     else
3379         Xdeco( n ) = Xpred( n ) + value
3380         if (Xdeco( n ) > 4095) then
3381             Xdeco( n ) = 4095
3382         endif
3383     endif
```

#### 3384 E.3.6.5 DPCM6 for 12–6–12 Decoder

3385 Xenco( n ) has the following format:

3386 Xenco( n ) = “011 s xx”

3387 where,

3388 “011” is the code word
3389 “s” is the **sign** bit
3390 “xx” is the two bit **value** field

3391 The codec equation is described as follows:

```
3392     sign = Xenco( n ) & 0x4
3393     value = 32 * (Xenco( n ) & 0x3) + 74 + 15
3394     if (sign > 0) then
3395         Xdeco( n ) = Xpred( n ) - value
3396         if (Xdeco( n ) < 0) then
3397             Xdeco( n ) = 0
3398         endif
3399     else
3400         Xdeco( n ) = Xpred( n ) + value
3401         if (Xdeco( n ) > 4095) then
3402             Xdeco( n ) = 4095
3403         endif
3404     endif
```

#### 3405 E.3.6.6 DPCM7 for 12–6–12 Decoder

3406 Xenco( n ) has the following format:

3407 Xenco( n ) = “0011 s x”

3408 where,

3409 “0011” is the code word
3410 “s” is the **sign** bit
3411 “x” is the one bit **value** field

3412 The codec equation is described as follows:

```
3413     sign = Xenco( n ) & 0x2
3414     value = 64 * (Xenco( n ) & 0x1) + 202 + 31
3415     if (sign > 0) then
3416         Xdeco( n ) = Xpred( n ) - value
3417         if (Xdeco( n ) < 0) then
3418             Xdeco( n ) = 0
3419         endif
3420     else
3421         Xdeco( n ) = Xpred( n ) + value
3422         if (Xdeco( n ) > 4095) then
3423             Xdeco( n ) = 4095
3424         endif
3425     endif
```

#### 3426 E.3.6.7 PCM for 12–6–12 Decoder

3427 Xenco( n ) has the following format:

3428 Xenco( n ) = “1 xxxx”

3429 where,

3430 “1” is the code word
3431 the **sign** bit is not used
3432 “xxxx” is the five bit **value** field

3433     The codec equation is described as follows:

```
3434       value = 128 * (Xenco( n ) & 0x1f)
3435       if (value > Xpred( n )) then
3436           Xdeco( n ) = value + 63
3437       else
3438           Xdeco( n ) = value + 64
3439       endif
```

## 3440 Annex F JPEG Interleaving (informative)

3441 This annex illustrates how the standard features of the CSI-2 protocol should be used to interleave (multiplex)  
3442 JPEG image data with other types of image data, e.g. RGB565 or YUV422, without requiring a custom JPEG  
3443 format such as JPEG8.

3444 The Virtual Channel Identifier and Data Type value in the CSI-2 Packet Header provide simple methods of  
3445 interleaving multiple data streams or image data types at the packet level. Interleaving at the packet level  
3446 minimizes the amount of buffering required in the system.

3447 The Data Type value in the CSI-2 Packet Header should be used to multiplex different image data types at  
3448 the CSI-2 transmitter and de-multiplex the data types at the CSI-2 receiver.

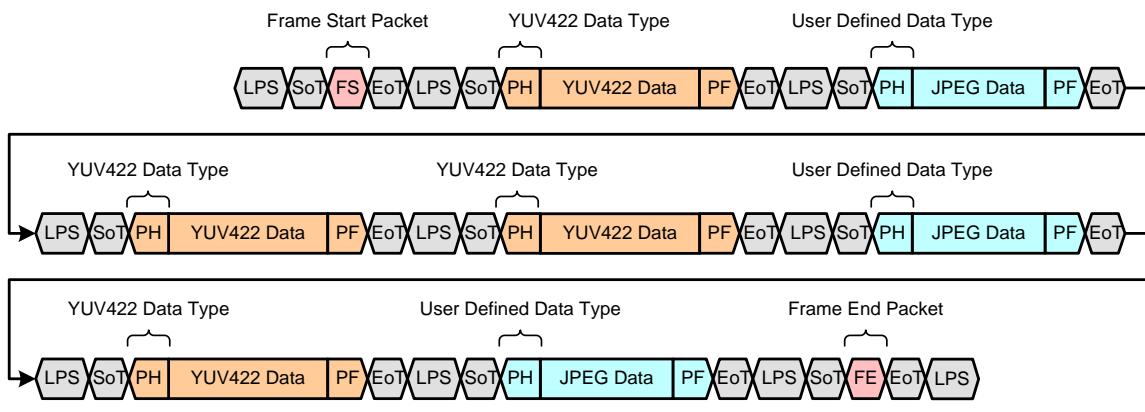
3449 The Virtual Channel Identifier in the CSI-2 Packet Header should be used to multiplex different data streams  
3450 (channels) at the CSI-2 transmitter and de-multiplex the streams at the CSI-2 receiver.

3451 The main difference between the two interleaving methods is that images with different Data Type values  
3452 within the same Virtual Channel use the same frame and line synchronization information, whereas multiple  
3453 Virtual Channels (data streams) each have their own independent frame and line synchronization information  
3454 and thus potentially each channel may have different frame rates.

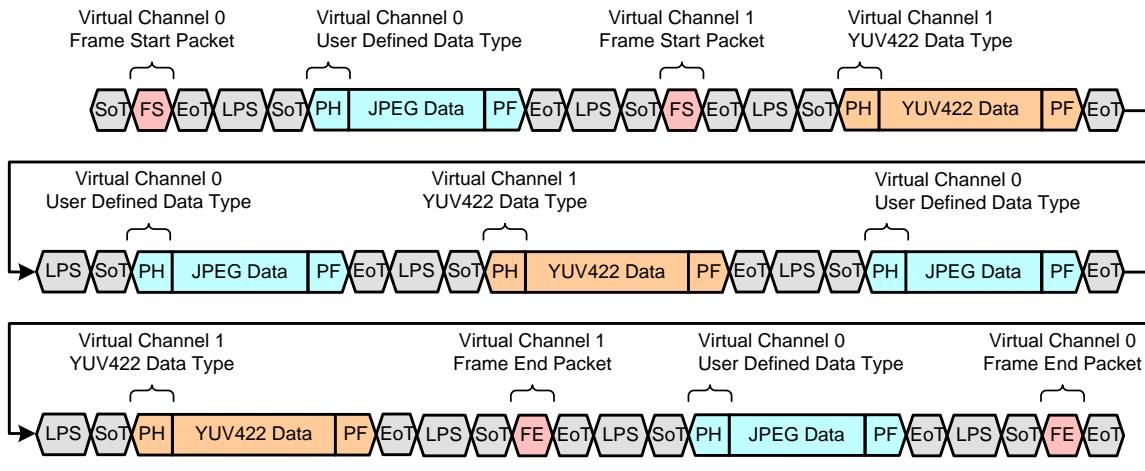
3455 Since the predefined Data Type values represent only YUV, RGB and RAW data types, one of the User  
3456 Defined Data Type values should be used to represent JPEG image data.

3457 Figure 145 illustrates interleaving JPEG image data with YUV422 image data using Data Type values.

3458 Figure 146 illustrates interleaving JPEG image data with YUV422 image data using both Data Type values  
3459 and Virtual Channel Identifiers.



3460  
3461 **Figure 145 Data Type Interleaving: Concurrent JPEG and YUV Image Data**



**KEY:**

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

PH – Packet Header  
PF – Packet Footer

FS – Frame Start Packet  
FE – Frame End Packet

3462

**Figure 146 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data**

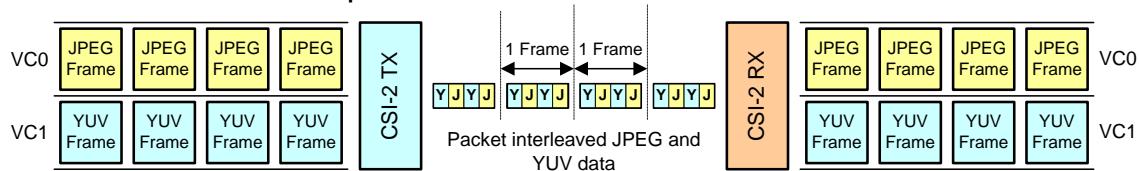
3464 Both Figure 145 and Figure 146 can be similarly extended to the interleaving of JPEG image data with any  
3465 other type of image data, e.g. RGB565.

3466 Figure 147 illustrates the use of Virtual Channels to support three different JPEG interleaving usage cases:

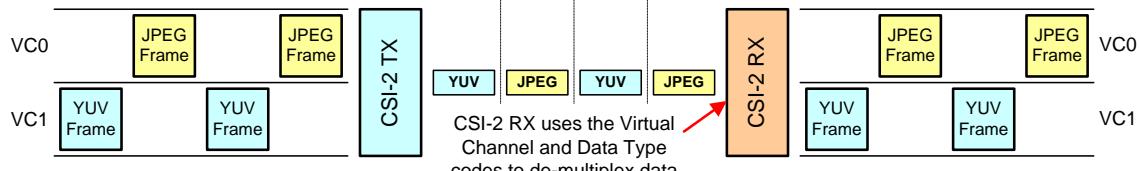
- 3467
- Concurrent JPEG and YUV422 image data.
  - Alternating JPEG and YUV422 output - one frame JPEG, then one frame YUV
  - Streaming YUV22 with occasional JPEG for still capture
- 3469

3470 Again, these examples could also represent interleaving JPEG data with any other image data type.

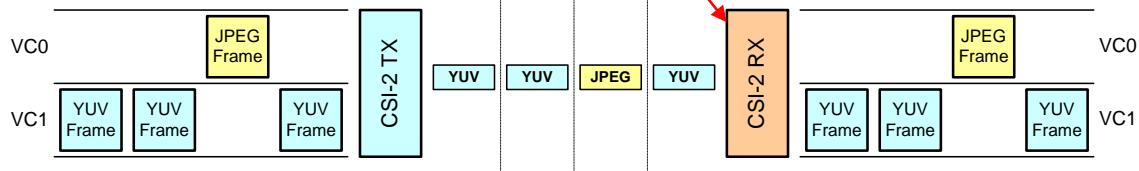
**Use Case 1: Concurrent JPEG output with YUV data**



**Use Case 2: Alternating JPEG and YUV output – one frame JPEG, then one frame YUV**



**Use Case 3: Streaming YUV with occasional JPEG still capture**



3471

3472

**Figure 147 Example JPEG and YUV Interleaving Use Cases**