



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

**Version 1.01.00 – 9 November 2010**

MIPI Board Approved 14-Feb-2011

Further technical changes to this document are expected as work continues in the Camera Working Group

**1 NOTICE OF DISCLAIMER**

2 The material contained herein is not a license, either expressly or impliedly, to any IPR owned or controlled  
3 by any of the authors or developers of this material or MIPI®. The material contained herein is provided on  
4 an "AS IS" basis and to the maximum extent permitted by applicable law, this material is provided AS IS  
5 AND WITH ALL FAULTS, and the authors and developers of this material and MIPI hereby disclaim all  
6 other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if  
7 any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of  
8 accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of  
9 negligence.

10 All materials contained herein are protected by copyright laws, and may not be reproduced, republished,  
11 distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express  
12 prior written permission of MIPI Alliance. MIPI, MIPI Alliance and the dotted rainbow arch and all related  
13 trademarks, tradenames, and other intellectual property are the exclusive property of MIPI Alliance and  
14 cannot be used without its express prior written permission.

15 ALSO, THERE IS NO WARRANTY OF CONDITION OF TITLE, QUIET ENJOYMENT, QUIET  
16 POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD  
17 TO THIS MATERIAL OR THE CONTENTS OF THIS DOCUMENT. IN NO EVENT WILL ANY  
18 AUTHOR OR DEVELOPER OF THIS MATERIAL OR THE CONTENTS OF THIS DOCUMENT OR  
19 MIPI BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE  
20 GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL,  
21 CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER  
22 CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR  
23 ANY OTHER AGREEMENT, SPECIFICATION OR DOCUMENT RELATING TO THIS MATERIAL,  
24 WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH  
25 DAMAGES.

26 Without limiting the generality of this Disclaimer stated above, the user of the contents of this Document is  
27 further notified that MIPI: (a) does not evaluate, test or verify the accuracy, soundness or credibility of the  
28 contents of this Document; (b) does not monitor or enforce compliance with the contents of this Document;  
29 and (c) does not certify, test, or in any manner investigate products or services or any claims of compliance  
30 with the contents of this Document. The use or implementation of the contents of this Document may  
31 involve or require the use of intellectual property rights ("IPR") including (but not limited to) patents,  
32 patent applications, or copyrights owned by one or more parties, whether or not Members of MIPI. MIPI  
33 does not make any search or investigation for IPR, nor does MIPI require or request the disclosure of any  
34 IPR or claims of IPR as respects the contents of this Document or otherwise.

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

36 MIPI Alliance, Inc.  
37 c/o IEEE-ISTO  
38 445 Hoes Lane  
39 Piscataway, NJ 08854  
40 Attn: Board Secretary

## 42 Contents

43	Version 1.01.00 – 9 November 2010.....	i
44	1 Overview .....	16
45	1.1 Scope .....	16
46	1.2 Purpose .....	16
47	2 Terminology .....	17
48	2.1 Definitions .....	17
49	2.2 Abbreviations .....	18
50	2.3 Acronyms .....	18
51	3 References .....	20
52	4 Overview of CSI-2 .....	21
53	5 CSI-2 Layer Definitions .....	22
54	6 Camera Control Interface (CCI).....	24
55	6.1 Data Transfer Protocol.....	24
56	6.1.1 Message Type.....	24
57	6.1.2 Read/Write Operations .....	25
58	6.2 CCI Slave Addresses .....	28
59	6.3 CCI Multi-Byte Registers.....	28
60	6.3.1 Overview .....	28
61	6.3.2 The Transmission Byte Order for Multi-byte Register Values.....	30
62	6.3.3 Multi-Byte Register Protocol.....	31
63	6.4 Electrical Specifications and Timing for I/O Stages.....	35
64	7 Physical Layer .....	38
65	8 Multi-Lane Distribution and Merging .....	39
66	8.1 Multi-Lane Interoperability .....	43
67	9 Low Level Protocol .....	46
68	9.1 Low Level Protocol Packet Format .....	46

69	9.1.1	Low Level Protocol Long Packet Format.....	46
70	9.1.2	Low Level Protocol Short Packet Format.....	48
71	9.2	Data Identifier (DI).....	48
72	9.3	Virtual Channel Identifier.....	48
73	9.4	Data Type (DT) .....	49
74	9.5	Packet Header Error Correction Code .....	50
75	9.5.1	General Hamming Code Applied to Packet Header .....	51
76	9.5.2	Hamming-modified Code .....	51
77	9.5.3	ECC Generation on TX Side .....	54
78	9.5.4	Applying ECC on RX Side.....	55
79	9.6	Checksum Generation.....	56
80	9.7	Packet Spacing.....	58
81	9.8	Synchronization Short Packet Data Type Codes .....	59
82	9.8.1	Frame Synchronization Packets.....	59
83	9.8.2	Line Synchronization Packets.....	60
84	9.9	Generic Short Packet Data Type Codes.....	60
85	9.10	Packet Spacing Examples .....	61
86	9.11	Packet Data Payload Size Rules .....	63
87	9.12	Frame Format Examples.....	64
88	9.13	Data Interleaving .....	66
89	9.13.1	Data Type Interleaving .....	66
90	9.13.2	Virtual Channel Identifier Interleaving.....	69
91	10	Color Spaces.....	71
92	10.1	RGB Color Space Definition .....	71
93	10.2	YUV Color Space Definition.....	71
94	11	Data Formats .....	72
95	11.1	Generic 8-bit Long Packet Data Types.....	73
96	11.1.1	Null and Blanking Data .....	73

97	11.1.2	Embedded Information .....	73
98	11.2	YUV Image Data .....	73
99	11.2.1	Legacy YUV420 8-bit .....	74
100	11.2.2	YUV420 8-bit .....	76
101	11.2.3	YUV420 10-bit .....	79
102	11.2.4	YUV422 8-bit .....	81
103	11.2.5	YUV422 10-bit .....	82
104	11.3	RGB Image Data .....	83
105	11.3.1	RGB888 .....	84
106	11.3.2	RGB666 .....	85
107	11.3.3	RGB565 .....	86
108	11.3.4	RGB555 .....	87
109	11.3.5	RGB444 .....	88
110	11.4	RAW Image Data .....	88
111	11.4.1	RAW6 .....	89
112	11.4.2	RAW7 .....	89
113	11.4.3	RAW8 .....	90
114	11.4.4	RAW10 .....	91
115	11.4.5	RAW12 .....	92
116	11.4.6	RAW14 .....	93
117	11.5	User Defined Data Formats .....	94
118	12	Recommended Memory Storage .....	97
119	12.1	General/Arbitrary Data Reception .....	97
120	12.2	RGB888 Data Reception .....	97
121	12.3	RGB666 Data Reception .....	98
122	12.4	RGB565 Data Reception .....	99
123	12.5	RGB555 Data Reception .....	99
124	12.6	RGB444 Data Reception .....	99

125	12.7	YUV422 8-bit Data Reception .....	100
126	12.8	YUV422 10-bit Data Reception .....	100
127	12.9	YUV420 8-bit (Legacy) Data Reception .....	101
128	12.10	YUV420 8-bit Data Reception .....	102
129	12.11	YUV420 10-bit Data Reception .....	103
130	12.12	RAW6 Data Reception .....	105
131	12.13	RAW7 Data Reception .....	105
132	12.14	RAW8 Data Reception .....	105
133	12.15	RAW10 Data Reception .....	106
134	12.16	RAW12 Data Reception .....	106
135	12.17	RAW14 Data Reception .....	107
136		Annex A JPEG8 Data Format (informative) .....	108
137	A.1	Introduction .....	108
138	A.2	JPEG Data Definition .....	109
139	A.3	Image Status Information .....	109
140	A.4	Embedded Images .....	111
141	A.5	JPEG8 Non-standard Markers .....	112
142	A.6	JPEG8 Data Reception .....	112
143		Annex B CSI-2 Implementation Example (informative) .....	113
144	B.1	Overview .....	113
145	B.2	CSI-2 Transmitter Detailed Block Diagram .....	113
146	B.3	CSI-2 Receiver Detailed Block Diagram .....	114
147	B.4	Details on the D-PHY implementation .....	115
148	B.4.1	CSI-2 Clock Lane Transmitter .....	117
149	B.4.2	CSI-2 Clock Lane Receiver .....	117
150	B.4.3	CSI-2 Data Lane Transmitter .....	118
151	B.4.4	CSI-2 Data Lane Receiver .....	120
152		Annex C CSI-2 Recommended Receiver Error Behavior (informative) .....	122

153	C.1	Overview .....	122
154	C.2	D-PHY Level Error .....	123
155	C.3	Packet Level Error .....	123
156	C.4	Protocol Decoding Level Error.....	124
157	Annex D CSI-2 Sleep Mode (informative).....		126
158	D.1	Overview .....	126
159	D.2	SLM Command Phase .....	126
160	D.3	SLM Entry Phase.....	126
161	D.4	SLM Exit Phase.....	127
162	Annex E Data Compression for RAW Data Types (normative).....		128
163	E.1	Predictors.....	129
164	E.1.1	Predictor1 .....	130
165	E.1.2	Predictor2 .....	130
166	E.2	Encoders .....	131
167	E.2.1	Coder for 10–8–10 Data Compression .....	131
168	E.2.2	Coder for 10–7–10 Data Compression .....	133
169	E.2.3	Coder for 10–6–10 Data Compression .....	136
170	E.2.4	Coder for 12–8–12 Data Compression .....	138
171	E.2.5	Coder for 12–7–12 Data Compression .....	141
172	E.2.6	Coder for 12–6–12 Data Compression .....	145
173	E.3	Decoders .....	148
174	E.3.1	Decoder for 10–8–10 Data Compression.....	148
175	E.3.2	Decoder for 10–7–10 Data Compression.....	150
176	E.3.3	Decoder for 10–6–10 Data Compression.....	153
177	E.3.4	Decoder for 12–8–12 Data Compression.....	156
178	E.3.5	Decoder for 12–7–12 Data Compression.....	159
179	E.3.6	Decoder for 12–6–12 Data Compression.....	163
180	Annex F JPEG Interleaving (informative).....		168

## Figures

182	<b>Figures</b>	
183	Figure 1 CSI-2 and CCI Transmitter and Receiver Interface .....	21
184	Figure 2 CSI-2 Layer Definitions .....	22
185	Figure 3 CCI Message Types .....	25
186	Figure 4 CCI Single Read from Random Location .....	25
187	Figure 5 CCI Single Read from Current Location .....	26
188	Figure 6 CCI Sequential Read Starting from a Random Location .....	26
189	Figure 7 CCI Sequential Read Starting from the Current Location .....	27
190	Figure 8 CCI Single Write to a Random Location .....	27
191	Figure 9 CCI Sequential Write Starting from a Random Location .....	28
192	Figure 10 Corruption of a 32-bit Wide Register during a Read Message .....	29
193	Figure 11 Corruption of a 32-bit Wide Register during a Write Message .....	30
194	Figure 12 Example 16-bit Register Write .....	30
195	Figure 13 Example 32-bit Register Write (address not shown) .....	31
196	Figure 14 Example 64-bit Register Write (address not shown) .....	31
197	Figure 15 Example 16-bit Register Read .....	32
198	Figure 16 Example 32-bit Register Read .....	33
199	Figure 17 Example 16-bit Register Write .....	34
200	Figure 18 Example 32-bit Register Write .....	35
201	Figure 19 CCI Timing .....	37
202	Figure 20 Conceptual Overview of the Lane Distributor Function .....	39
203	Figure 21 Conceptual Overview of the Lane Merging Function .....	40
204	Figure 22 Two Lane Multi-Lane Example .....	41
205	Figure 23 Three Lane Multi-Lane Example .....	42
206	Figure 24 Four Lane Multi-Lane Example .....	43
207	Figure 25 One Lane Transmitter and Four Lane Receiver Example .....	44
208	Figure 26 Two Lane Transmitter and Four Lane Receiver Example .....	44



209	Figure 27 Four Lane Transmitter and One Lane Receiver Example .....	45
210	Figure 28 Four Lane Transmitter and Two Lane Receiver Example .....	45
211	Figure 29 Low Level Protocol Packet Overview.....	46
212	Figure 30 Long Packet Structure .....	47
213	Figure 31 Short Packet Structure.....	48
214	Figure 32 Data Identifier Byte.....	48
215	Figure 33 Logical Channel Block Diagram (Receiver) .....	49
216	Figure 34 Interleaved Video Data Streams Examples .....	49
217	Figure 35 24-bit ECC Generation Example.....	50
218	Figure 36 64-bit ECC Generation on TX Side .....	54
219	Figure 37 24-bit ECC Generation on TX Side .....	55
220	Figure 38 64-bit ECC on RX Side Including Error Correction .....	55
221	Figure 39 24-bit ECC on RX side Including Error Correction.....	56
222	Figure 40 Checksum Transmission .....	56
223	Figure 41 Checksum Generation for Packet Data .....	57
224	Figure 42 Definition of 16-bit CRC Shift Register.....	57
225	Figure 43 16-bit CRC Software Implementation Example.....	58
226	Figure 44 Packet Spacing .....	59
227	Figure 45 Multiple Packet Example .....	61
228	Figure 46 Single Packet Example.....	61
229	Figure 47 Line and Frame Blanking Definitions .....	62
230	Figure 48 Vertical Sync Example.....	63
231	Figure 49 Horizontal Sync Example .....	63
232	Figure 50 General Frame Format Example .....	64
233	Figure 51 Digital Interlaced Video Example.....	65
234	Figure 52 Digital Interlaced Video with Accurate Synchronization Timing Information .....	66
235	Figure 53 Interleaved Data Transmission using Data Type Value .....	67
236	Figure 54 Packet Level Interleaved Data Transmission .....	68

237	Figure 55 Frame Level Interleaved Data Transmission.....	69
238	Figure 56 Interleaved Data Transmission using Virtual Channels .....	70
239	Figure 57 Frame Structure with Embedded Data at the Beginning and End of the Frame.....	74
240	Figure 58 Legacy YUV420 8-bit Transmission .....	75
241	Figure 59 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration.....	75
242	Figure 60 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1 .....	76
243	Figure 61 Legacy YUV420 8-bit Frame Format .....	76
244	Figure 62 YUV420 8-bit Data Transmission Sequence .....	77
245	Figure 63 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration .....	77
246	Figure 64 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1 .....	78
247	Figure 65 YUV420 Spatial Sampling for MPEG 2 and MPEG 4 .....	78
248	Figure 66 YUV420 8-bit Frame Format.....	79
249	Figure 67 YUV420 10-bit Transmission .....	80
250	Figure 68 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration .....	80
251	Figure 69 YUV420 10-bit Frame Format.....	80
252	Figure 70 YUV422 8-bit Transmission .....	81
253	Figure 71 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration .....	81
254	Figure 72 YUV422 Co-sited Spatial Sampling .....	82
255	Figure 73 YUV422 8-bit Frame Format.....	82
256	Figure 74 YUV422 10-bit Transmitted Bytes .....	83
257	Figure 75 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration .....	83
258	Figure 76 YUV422 10-bit Frame Format.....	83
259	Figure 77 RGB888 Transmission .....	84
260	Figure 78 RGB888 Transmission in CSI-2 Bus Bitwise Illustration.....	84
261	Figure 79 RGB888 Frame Format.....	85
262	Figure 80 RGB666 Transmission with 18-bit BGR Words.....	85
263	Figure 81 RGB666 Transmission on CSI-2 Bus Bitwise Illustration.....	85
264	Figure 82 RGB666 Frame Format.....	86

265	Figure 83 RGB565 Transmission with 16-bit BGR Words.....	86
266	Figure 84 RGB565 Transmission on CSI-2 Bus Bitwise Illustration.....	87
267	Figure 85 RGB565 Frame Format.....	87
268	Figure 86 RGB555 Transmission on CSI-2 Bus Bitwise Illustration.....	87
269	Figure 87 RGB444 Transmission on CSI-2 Bus Bitwise Illustration.....	88
270	Figure 88 RAW6 Transmission.....	89
271	Figure 89 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration .....	89
272	Figure 90 RAW6 Frame Format.....	89
273	Figure 91 RAW7 Transmission.....	90
274	Figure 92 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration .....	90
275	Figure 93 RAW7 Frame Format.....	90
276	Figure 94 RAW8 Transmission.....	91
277	Figure 95 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration .....	91
278	Figure 96 RAW8 Frame Format.....	91
279	Figure 97 RAW10 Transmission.....	92
280	Figure 98 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration .....	92
281	Figure 99 RAW10 Frame Format.....	92
282	Figure 100 RAW12 Transmission.....	93
283	Figure 101 RAW12 Transmission on CSI-2 Bus Bitwise Illustration.....	93
284	Figure 102 RAW12 Frame Format.....	93
285	Figure 103 RAW14 Transmission.....	94
286	Figure 104 RAW14 Transmission on CSI-2 Bus Bitwise Illustration.....	94
287	Figure 105 RAW14 Frame Format.....	94
288	Figure 106 User Defined 8-bit Data (128 Byte Packet).....	95
289	Figure 107 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration.....	95
290	Figure 108 Transmission of User Defined 8-bit Data.....	95
291	Figure 109 General/Arbitrary Data Reception.....	97
292	Figure 110 RGB888 Data Format Reception .....	98

293	Figure 111 RGB666 Data Format Reception .....	98
294	Figure 112 RGB565 Data Format Reception .....	99
295	Figure 113 RGB555 Data Format Reception .....	99
296	Figure 114 RGB444 Data Format Reception .....	100
297	Figure 115 YUV422 8-bit Data Format Reception .....	100
298	Figure 116 YUV422 10-bit Data Format Reception .....	101
299	Figure 117 YUV420 8-bit Legacy Data Format Reception.....	102
300	Figure 118 YUV420 8-bit Data Format Reception .....	103
301	Figure 119 YUV420 10-bit Data Format Reception .....	104
302	Figure 120 RAW6 Data Format Reception .....	105
303	Figure 121 RAW7 Data Format Reception .....	105
304	Figure 122 RAW8 Data Format Reception .....	106
305	Figure 123 RAW10 Data Format Reception .....	106
306	Figure 124 RAW12 Data Format Reception .....	107
307	Figure 125 RAW 14 Data Format Reception .....	107
308	Figure 126 JPEG8 Data Flow in the Encoder.....	108
309	Figure 127 JPEG8 Data Flow in the Decoder .....	108
310	Figure 128 EXIF Compatible Baseline JPEG DCT Format .....	109
311	Figure 129 Status Information Field in the End of Baseline JPEG Frame .....	110
312	Figure 130 Example of TN Image Embedding Inside the Compressed JPEG Data Block .....	111
313	Figure 131 JPEG8 Data Format Reception .....	112
314	Figure 132 Implementation Example Block Diagram and Coverage.....	113
315	Figure 133 CSI-2 Transmitter Block Diagram .....	114
316	Figure 134 CSI-2 Receiver Block Diagram.....	115
317	Figure 135 D-PHY Level Block Diagram.....	116
318	Figure 136 CSI-2 Clock Lane Transmitter .....	117
319	Figure 137 CSI-2 Clock Lane Receiver.....	118
320	Figure 138 CSI-2 Data Lane Transmitter .....	119

321	Figure 139 CSI-2 Data Lane Receiver.....	120
322	Figure 140 SLM Synchronization .....	127
323	Figure 141 Data Compression System Block Diagram .....	129
324	Figure 142 Pixel Order of the Original Image.....	129
325	Figure 143 Example Pixel Order of the Original Image.....	129
326	Figure 144 Data Type Interleaving: Concurrent JPEG and YUV Image Data .....	168
327	Figure 145 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data .....	169
328	Figure 146 Example JPEG and YUV Interleaving Use Cases .....	170
329		
330		

## 331 **Tables**

332	Table 1 CCI I/O Characteristics .....	35
333	Table 2 CCI Timing Specification.....	36
334	Table 3 Data Type Classes .....	50
335	Table 4 ECC Syndrome Association Matrix .....	51
336	Table 5 ECC Parity Generation Rules .....	52
337	Table 6 Synchronization Short Packet Data Type Codes .....	59
338	Table 7 Generic Short Packet Data Type Codes.....	60
339	Table 8 Primary and Secondary Data Formats Definitions .....	72
340	Table 9 Generic 8-bit Long Packet Data Types.....	73
341	Table 10 YUV Image Data Types .....	74
342	Table 11 Legacy YUV420 8-bit Packet Data Size Constraints .....	75
343	Table 12 YUV420 8-bit Packet Data Size Constraints.....	77
344	Table 13 YUV420 10-bit Packet Data Size Constraints.....	79
345	Table 14 YUV422 8-bit Packet Data Size Constraints.....	81
346	Table 15 YUV422 10-bit Packet Data Size Constraints.....	82
347	Table 16 RGB Image Data Types.....	84
348	Table 17 RGB888 Packet Data Size Constraints.....	84
349	Table 18 RGB666 Packet Data Size Constraints.....	85
350	Table 19 RGB565 Packet Data Size Constraints.....	86
351	Table 20 RAW Image Data Types .....	88
352	Table 21 RAW6 Packet Data Size Constraints.....	89
353	Table 22 RAW7 Packet Data Size Constraints.....	90
354	Table 23 RAW8 Packet Data Size Constraints.....	91
355	Table 24 RAW10 Packet Data Size Constraints.....	91
356	Table 25 RAW12 Packet Data Size Constraints.....	92
357	Table 26 RAW14 Packet Data Size Constraints.....	93

358	Table 27 User Defined 8-bit Data Types .....	95
359	Table 28 Status Data Padding .....	110
360	Table 29 JPEG8 Additional Marker Codes Listing .....	112
361		
362		

# MIPI Alliance Specification for Camera Serial Interface 2 (CSI-2)

## 1 Overview

### 1.1 Scope

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

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

### 1.2 Purpose

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

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



## 2 Terminology

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

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

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

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

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

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

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

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

### 2.1 Definitions

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

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

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

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

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

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

## 420 **2.2 Abbreviations**

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

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

## 423 **2.3 Acronyms**

424 BER Bit Error Rate

425 CCI Camera Control Interface

426 CIL Control and Interface Logic

427 CRC Cyclic Redundancy Check

428 CSI Camera Serial Interface

429 CSPS Chroma Sample Pixel Shifted

430 DDR Dual Data Rate

431 DI Data Identifier

432 DT Data Type

433 ECC Error Correction Code

434 EoT End of Transmission

435 EXIF Exchangeable Image File Format

436 FE Frame End

437 FS Frame Start

438 HS High Speed; identifier for operation mode

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

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

441 I2C Inter-Integrated Circuit

442 JFIF JPEG File Interchange Format

443 JPEG Joint Photographic Expert Group

444 LE Line End

445 LLP Low Level Protocol

446 LS Line Start

447	LSB	Least Significant Bit
448	LP	Low-Power; identifier for operation mode
449	LP-RX	Low-Power Receiver (Large-Swing Single Ended)
450	LP-TX	Low-Power Transmitter (Large-Swing Single Ended)
451	MIPI	Mobile Industry Processor Interface
452	MSB	Most Significant Bit
453	PF	Packet Footer
454	PH	Packet Header
455	PI	Packet Identifier
456	PT	Packet Type
457	PHY	Physical Layer
458	PPI	PHY Protocol Interface
459	RGB	Color representation (Red, Green, Blue)
460	RX	Receiver
461	SCL	Serial Clock (for CCI)
462	SDA	Serial Data (for CCI)
463	SLM	Sleep Mode
464	SoT	Start of Transmission
465	TX	Transmitter
466	ULPS	Ultra-low Power State
467	VGA	Video Graphics Array
468	YUV	Color representation (Y for luminance, U & V for chrominance)
469		

470   **3   References**

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

472   [MIP101]      *MIPI Alliance Specification for D-PHY*, version 1.00.00, MIPI Alliance, Inc., 14 May  
473                2009

474

4 Overview of CSI-2

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.

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

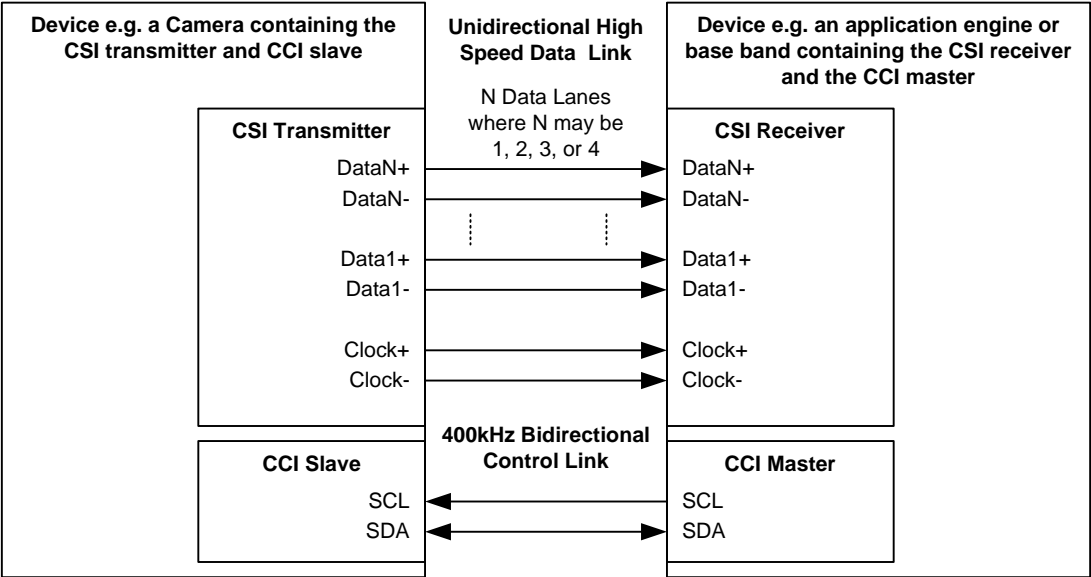
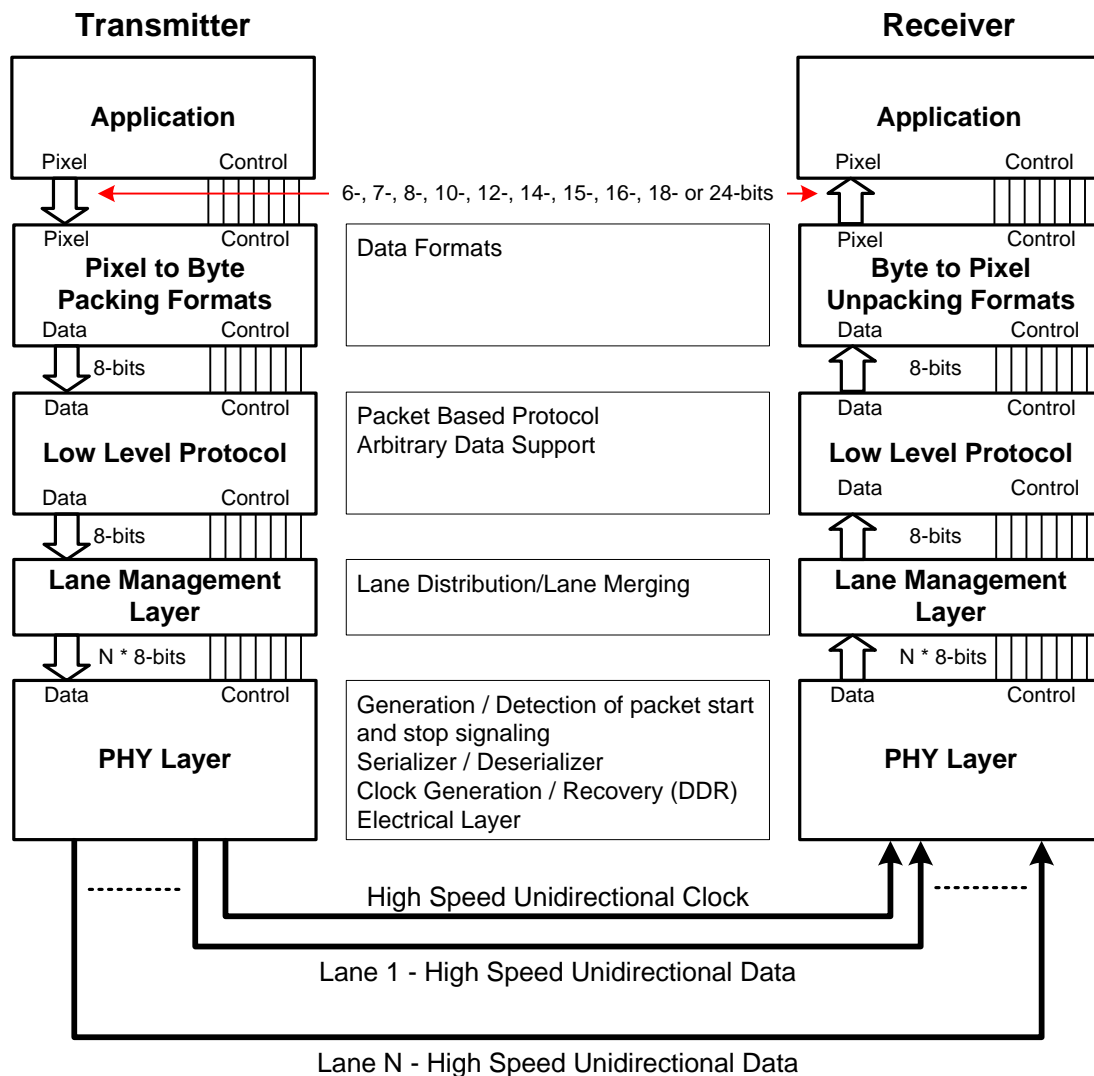


Figure 1 CSI-2 and CCI Transmitter and Receiver Interface

## 485 5 CSI-2 Layer Definitions



486  
487 **Figure 2 CSI-2 Layer Definitions**

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

- 489 • **PHY Layer.** The PHY Layer specifies the transmission medium (electrical conductors), the  
490 input/output circuitry and the clocking mechanism that captures “ones” and “zeroes” from the  
491 serial bit stream. This part of the specification documents the characteristics of the transmission  
492 medium, electrical parameters for signaling and the timing relationship between clock and data  
493 Lanes.

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

498 The PHY layer is described in [MIPI01].

- **Protocol Layer.** The Protocol layer is composed of several layers, each with distinct responsibilities. The CSI-2 protocol enables multiple data streams using a single interface on the host processor. The Protocol layer specifies how multiple data streams may be tagged and interleaved so each data stream can be properly reconstructed.
  - **Pixel/Byte Packing/Unpacking Layer.** The CSI-2 supports image applications with varying pixel formats from six to twenty-four bits per pixels. In the transmitter this layer packs pixels from the Application layer into bytes before sending the data to the Low Level Protocol layer. In the receiver this layer unpacks bytes from the Low Level Protocol layer into pixels before sending the data to the Application layer. Eight bits per pixel data is transferred unchanged by this layer.
  - **Low Level Protocol.** The Low Level Protocol (LLP) includes the means of establishing bit-level and byte-level synchronization for serial data transferred between SoT (Start of Transmission) and EoT (End of Transmission) events and for passing data to the next layer. The minimum data granularity of the LLP is one byte. The LLP also includes assignment of bit-value interpretation within the byte, i.e. the “Endian” assignment.
  - **Lane Management.** CSI-2 is Lane-scalable for increased performance. The number of data Lanes may be one, two, three or four depending on the bandwidth requirements of the application. The transmitting side of the interface distributes (“distributor” function) the outgoing data stream to one or more Lanes. On the receiving side, the interface collects bytes from the Lanes and merges (“merger” function) them together into a recombined data stream that restores the original stream sequence.
- Data within the Protocol layer is organized as packets. The transmitting side of the interface appends header and optional error-checking information on to data to be transmitted at the Low Level Protocol layer. On the receiving side, the header is stripped off at the Low Level Protocol layer and interpreted by corresponding logic in the receiver. Error-checking information may be used to test the integrity of incoming data.
- **Application Layer.** This layer describes higher-level encoding and interpretation of data contained in the data stream. The CSI-2 specification describes the mapping of pixel values to bytes.

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

## 6 Camera Control Interface (CCI)

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

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

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

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

Each CCI transmitter may have additional features to support I2C, but that is dependent on implementation. Further details can be found on a particular device's data sheet.

This specification does not attempt to define the contents of control messages sent by the CCI master. As such, it is the responsibility of the CSI-2 implementer to define a set of control messages and corresponding frame timing and I2C latency requirements, if any, that must be met by the CCI master when sending such control messages to the CCI slave.

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

### 6.1 Data Transfer Protocol

The data transfer protocol is according to I2C standard. The START, REPEATED START and STOP conditions as well as data transfer protocol are specified in *The I<sup>2</sup>C Specification* [NXP01].

#### 6.1.1 Message Type

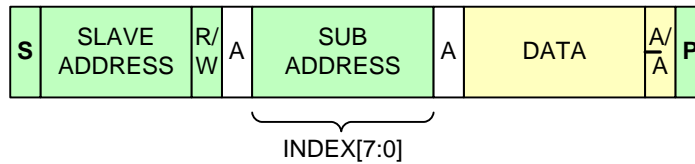
A basic CCI message consists of START condition, slave address with read/write bit, acknowledge from slave, sub address (index) for pointing at a register inside the slave device, acknowledge signal from slave, in write operation data byte from master, acknowledge/negative acknowledge from slave and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

The slave address in the CCI is 7-bit.

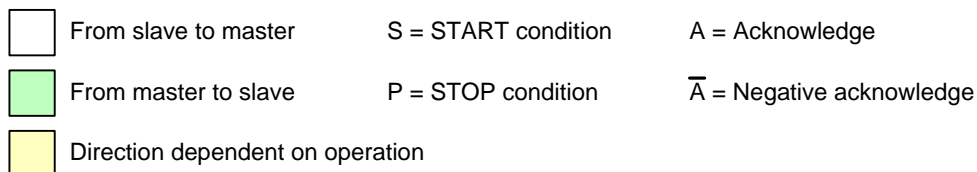
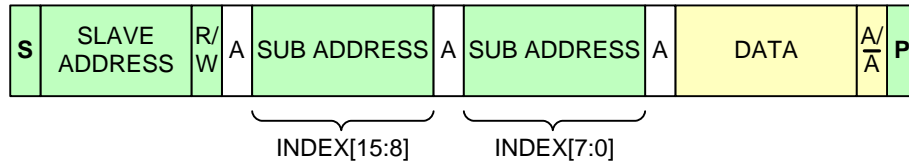
The CCI supports 8-bit index with 8-bit data or 16-bit index with 8-bit data. The slave device in question defines what message type is used.



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)



**Figure 3 CCI Message Types**

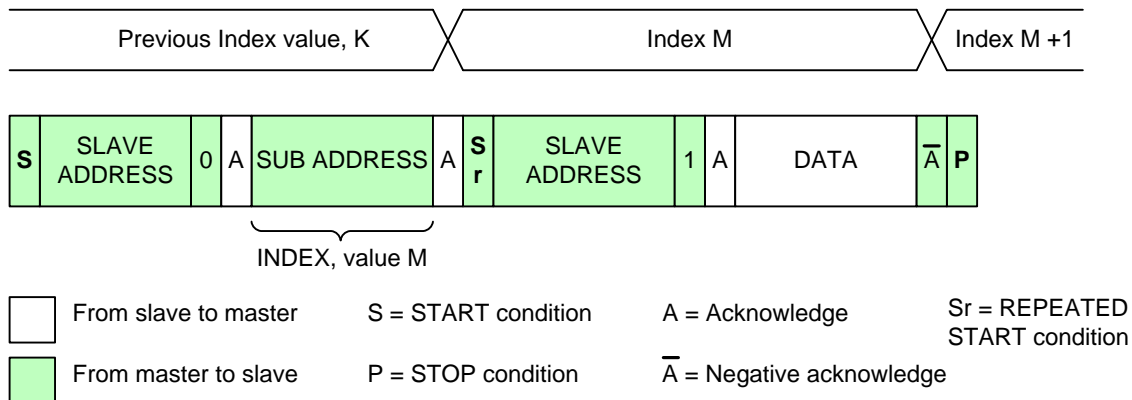
### 6.1.2 Read/Write Operations

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

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

#### 6.1.2.1 Single Read from Random Location

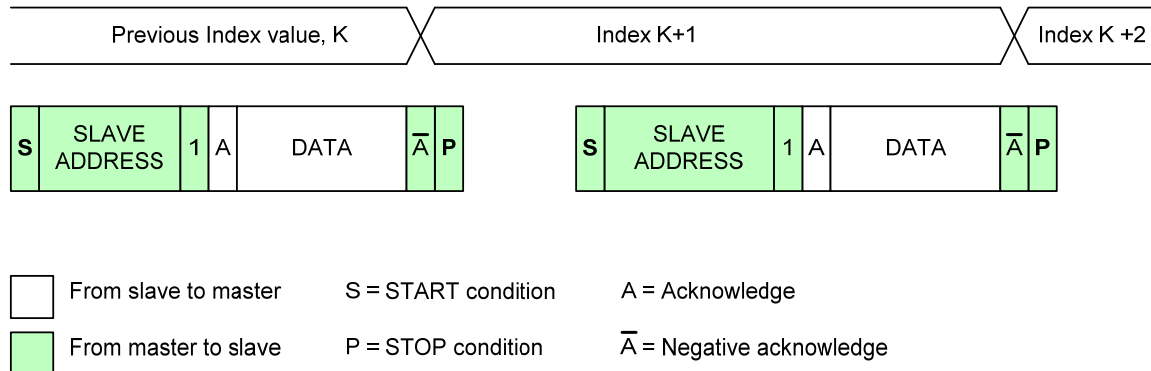
In single read from random location the master does a dummy write operation to desired index, issues a repeated start condition and then addresses the slave again with read operation. After acknowledging its slave address, the slave starts to output data onto SDA line. This is illustrated in Figure 4.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



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

### 6.1.2.2 Single Read from the Current Location

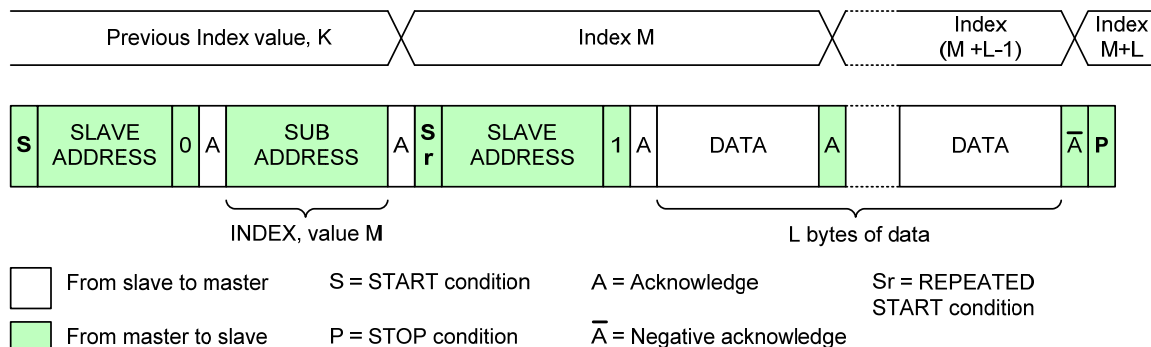
It is also possible to read from last used index by addressing the slave with read operation. The slave responds by setting the data from last used index to SDA line. This is illustrated in Figure 5. The master terminates the read operation by setting a negative acknowledge and stop condition.



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

### 6.1.2.3 Sequential Read Starting from a Random Location

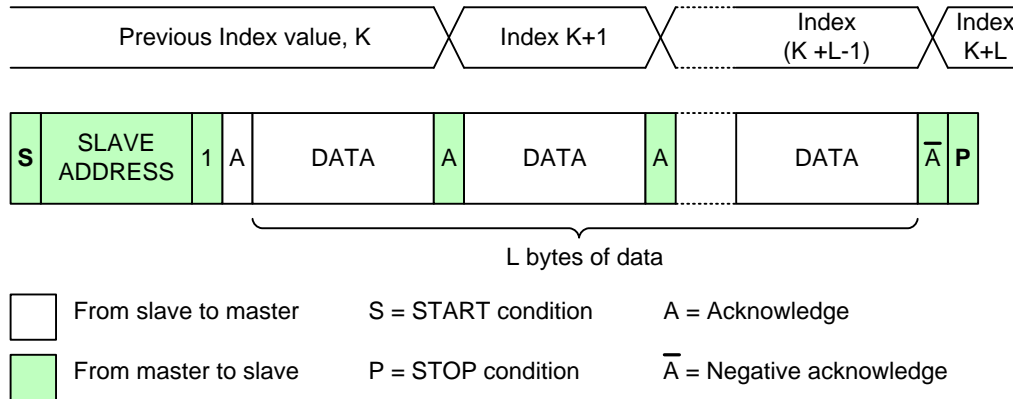
The sequential read starting from a random location is illustrated in Figure 6. The master does a dummy write to the desired index, issues a repeated start condition after an acknowledge from the slave and then addresses the slave again with a read operation. If a master issues an acknowledge after received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



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

### 6.1.2.4 Sequential Read Starting from the Current Location

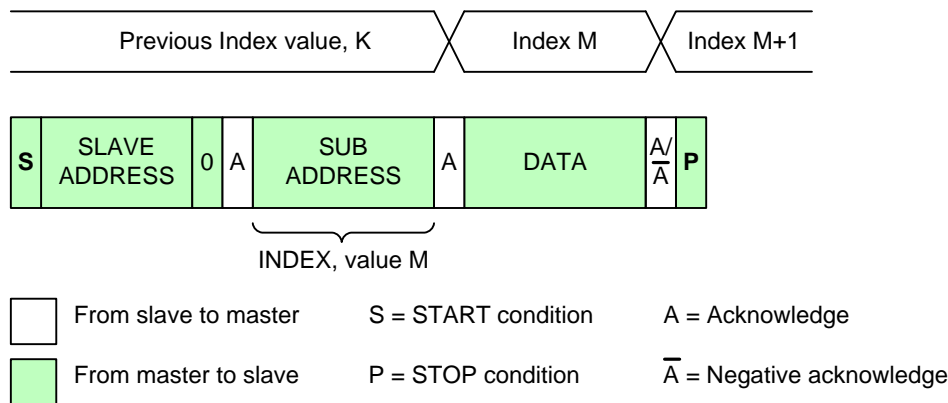
A sequential read starting from the current location is similar to a sequential read from a random location. The only exception is there is no dummy write operation. The command sequence is illustrated in Figure 7. The master terminates the read operation by issuing a negative acknowledge and stop condition.



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

### 6.1.2.5 Single Write to a Random Location

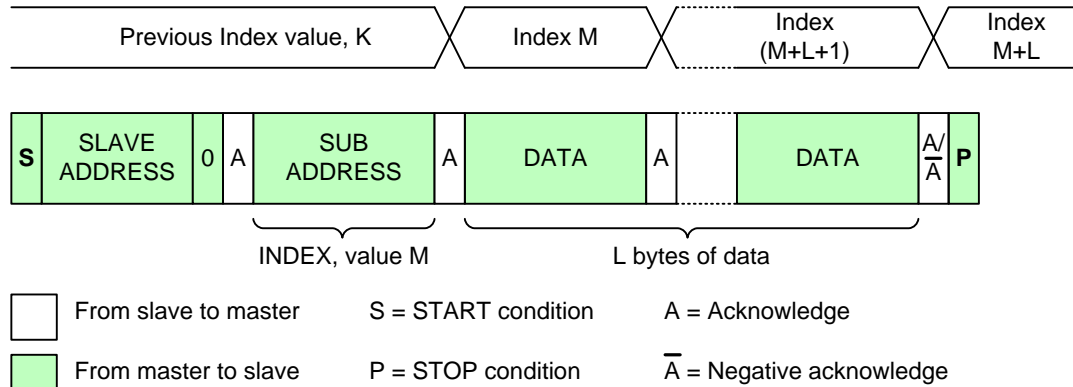
A write operation to a random location is illustrated in Figure 8. The master issues a write operation to the slave then issues the index and data after the slave has acknowledged the write operation. The write operation is terminated with a stop condition from the master.



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

### 6.1.2.6 Sequential Write

The sequential write operation is illustrated in Figure 9. The slave auto-increments the index after each data 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 011011Xb, where X = 0 or 1. For all other camera modules the 7-bit slave address should be 011110Xb.

## 6.3 CCI Multi-Byte Registers

### 6.3.1 Overview

Peripherals contain a wide range of different register widths for various control and setup purposes. The CSI-2 specification supports the following register widths:

- 8-bit – generic setup registers
- 16-bit – parameters like line-length, frame-length and exposure values
- 32-bit – high precision setup values
- 64-bit – for needs of future sensors

In general, the byte oriented access protocols described in the sections above 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 to be described below provides a way of transferring 16-bit, 32-bit or 64-bit values over a 16-bit index, 8-bit data, two-wire serial link while ensuring that the bytes of data transferred for a multi-byte register value are always consistent (temporally coherent).

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

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

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

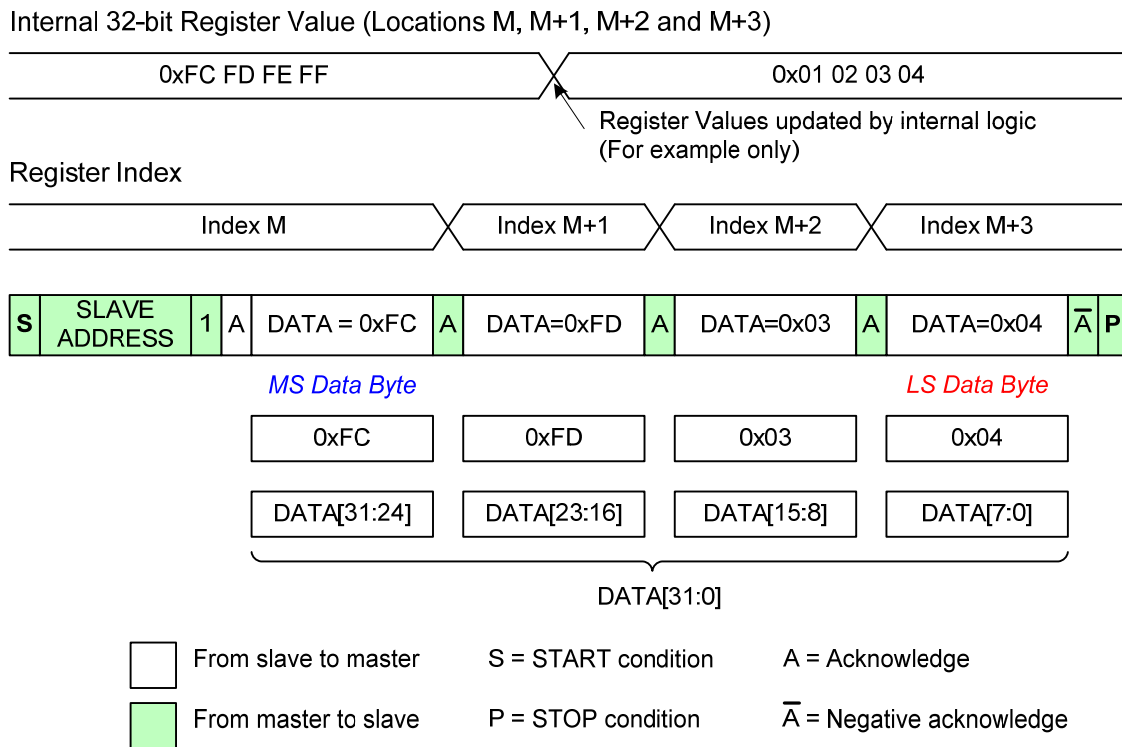
Partial access to multi-byte registers is not allowed. A multi-byte register shall only be accessed by a single sequential message. When a multi-byte register is accessed, its first byte is accessed first, its second byte is accessed second, etc.

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

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

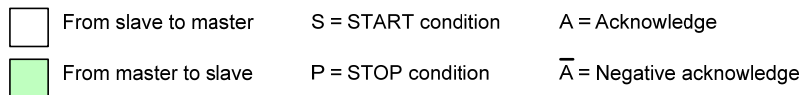
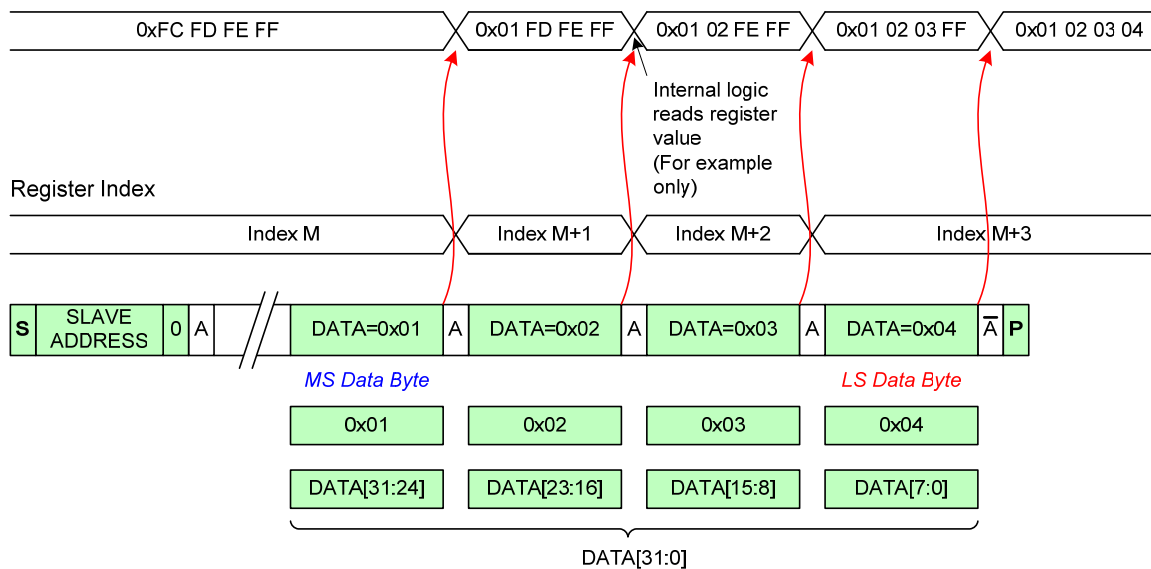
Section 6.3.3 describes example behavior for the re-timing of multi-byte register accesses.

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



**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)

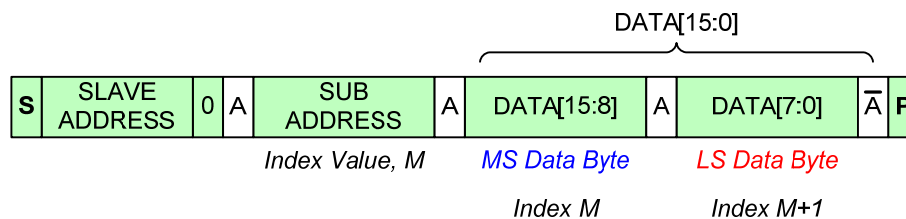


**Figure 11 Corruption of a 32-bit Wide Register during a Write Message**

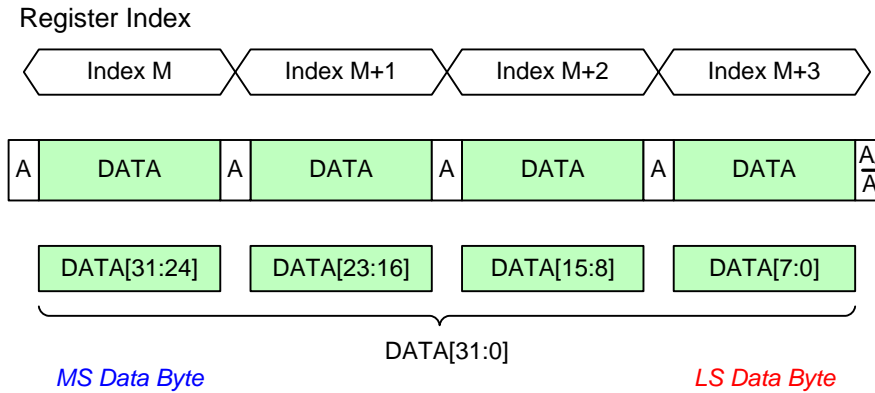
### 6.3.2 The Transmission Byte Order for Multi-byte Register Values

This is a normative section.

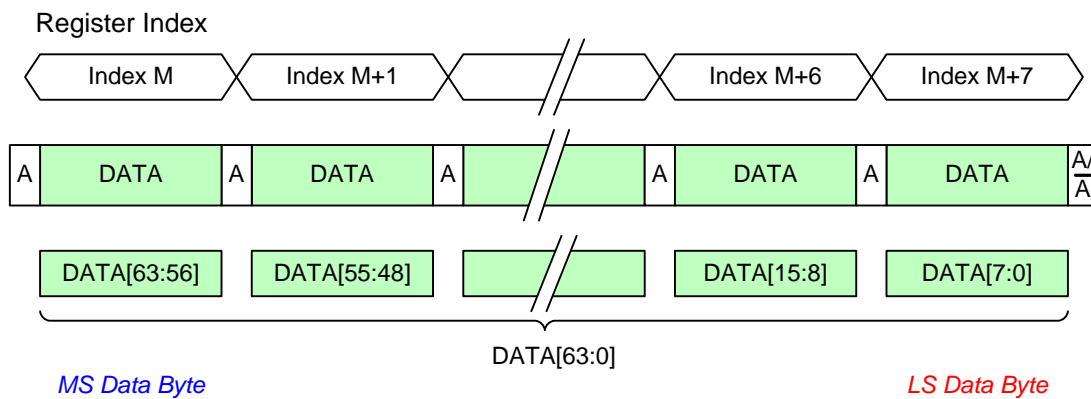
The first byte of a CCI message is always the MS byte of a multi-byte register and the last byte is always the LS byte.



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



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



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

### 6.3.3 Multi-Byte Register Protocol

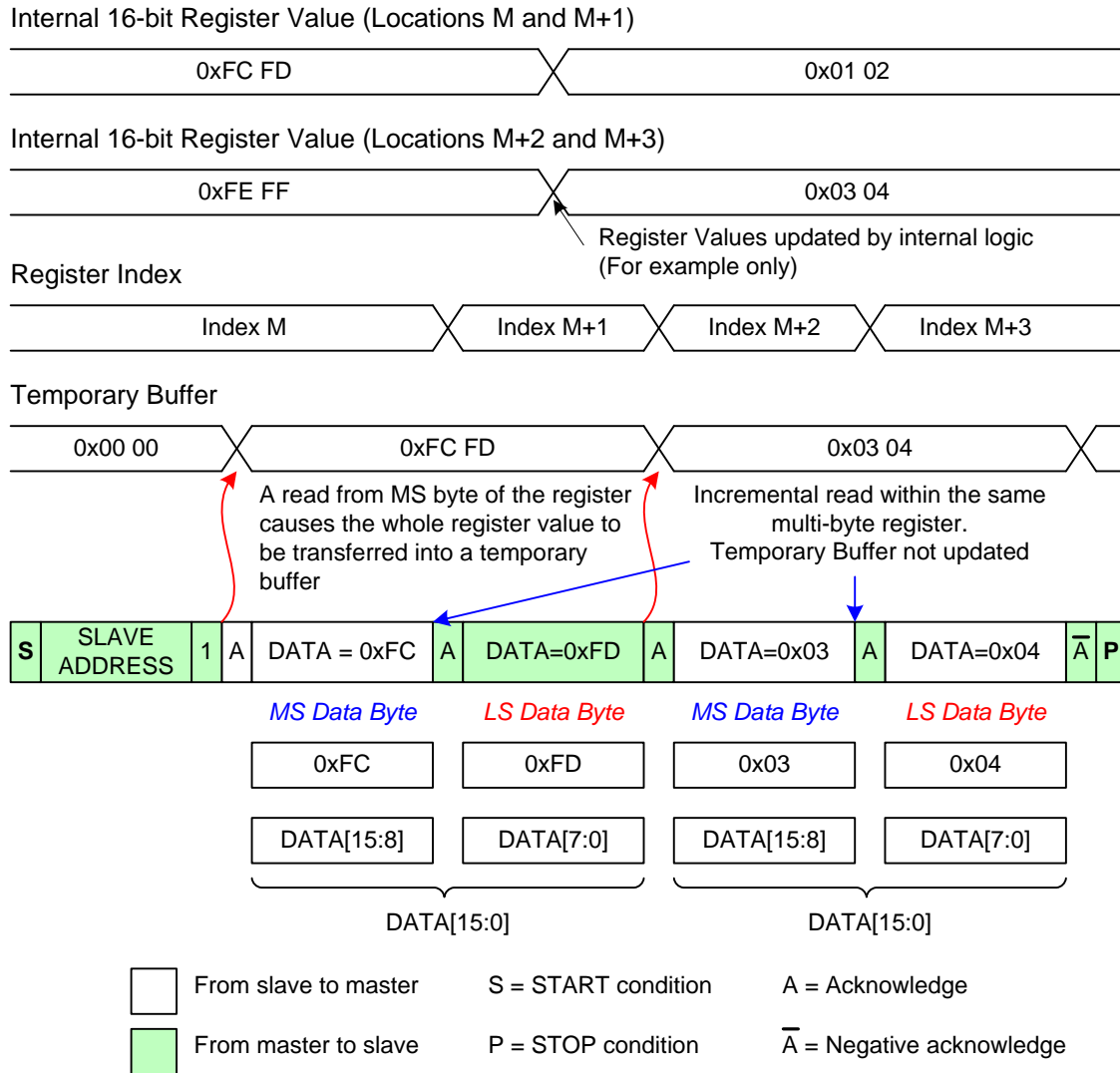
This is an informative section.

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

#### 6.3.3.1 Reading Multi-byte Registers

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

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



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

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

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

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

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

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



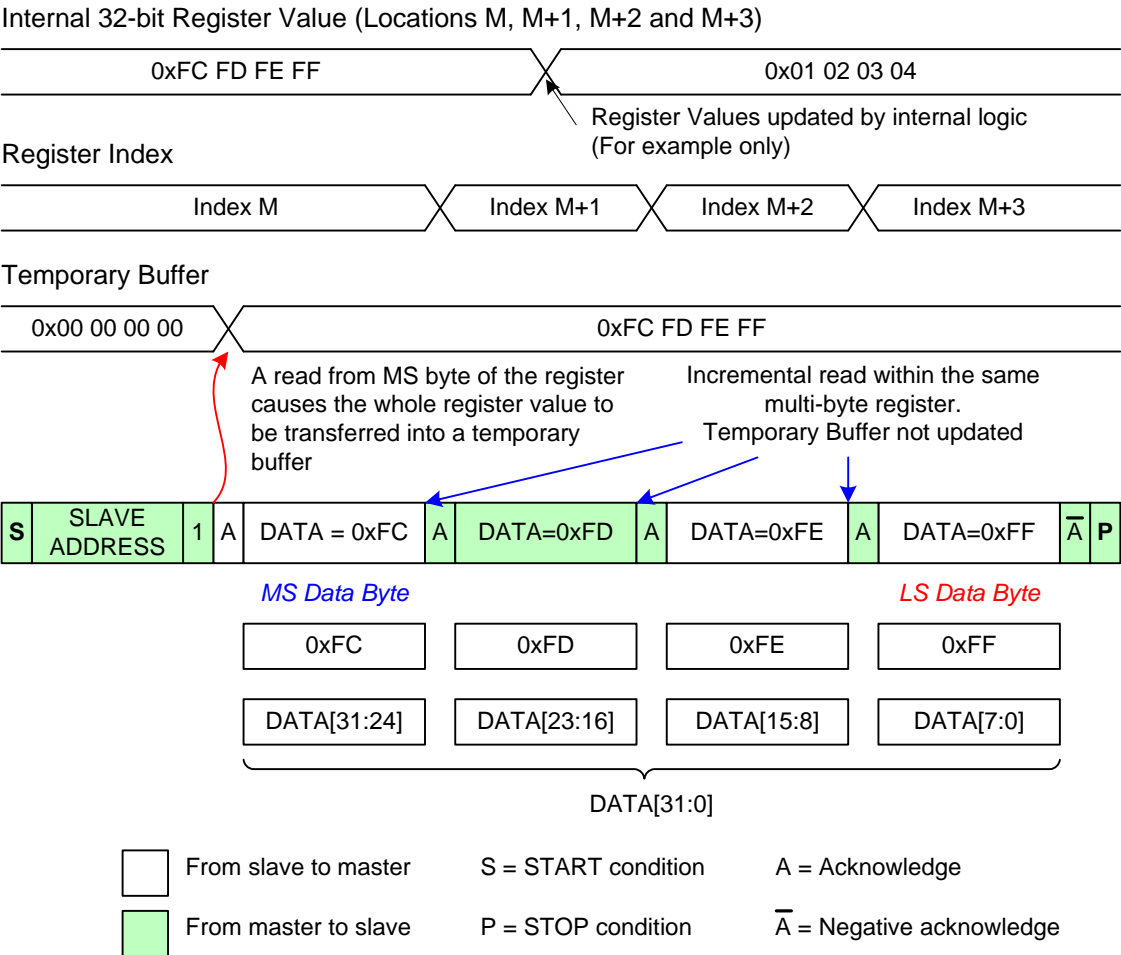


Figure 16 Example 32-bit Register Read

6.3.3.2 Writing Multi-byte Registers

To ensure that the value written is consistent, the bytes of data of a multi-byte register are written into a temporary buffer. Only after the LS byte of the register is written is the full multi-byte value transferred into the internal register location. Figure 17 and Figure 18 illustrate multi-byte register write operations.

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

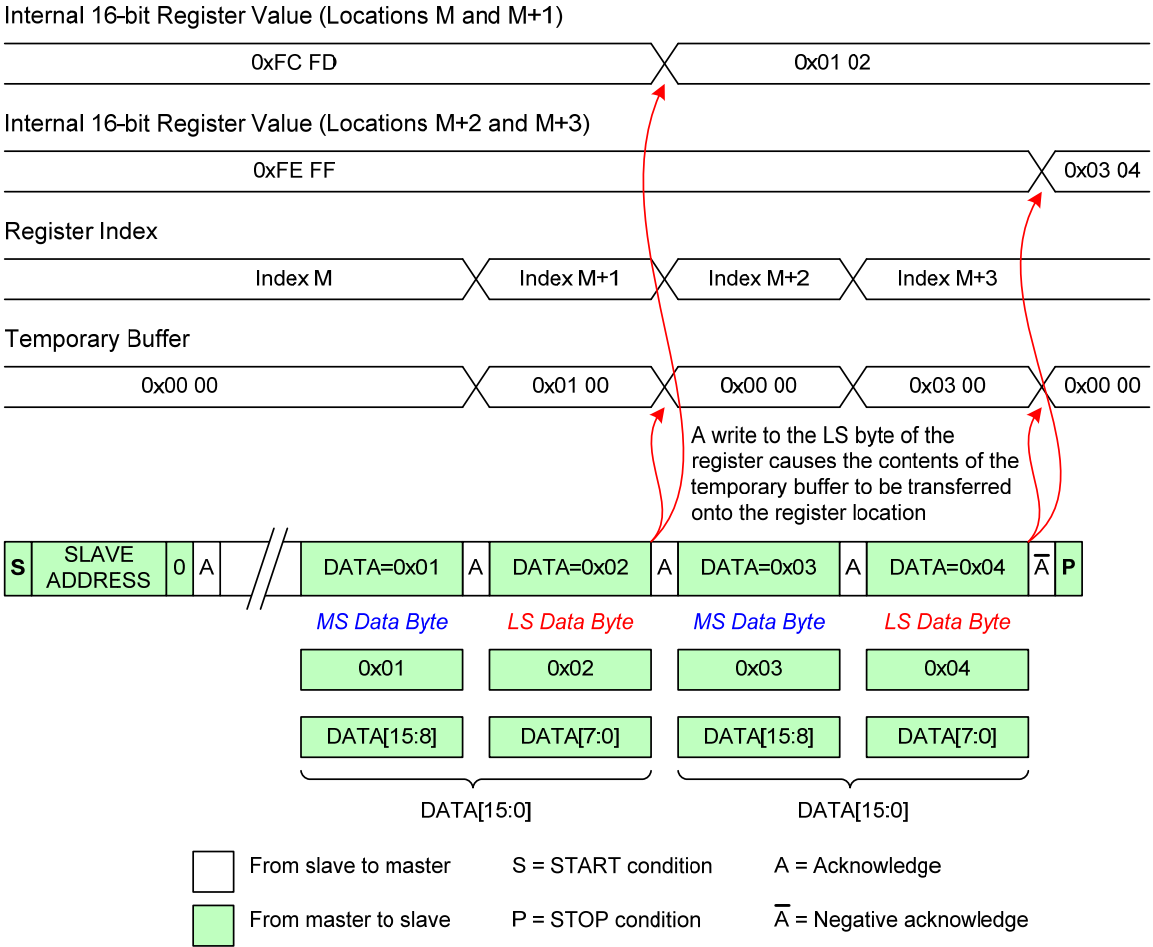


Figure 17 Example 16-bit Register Write

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

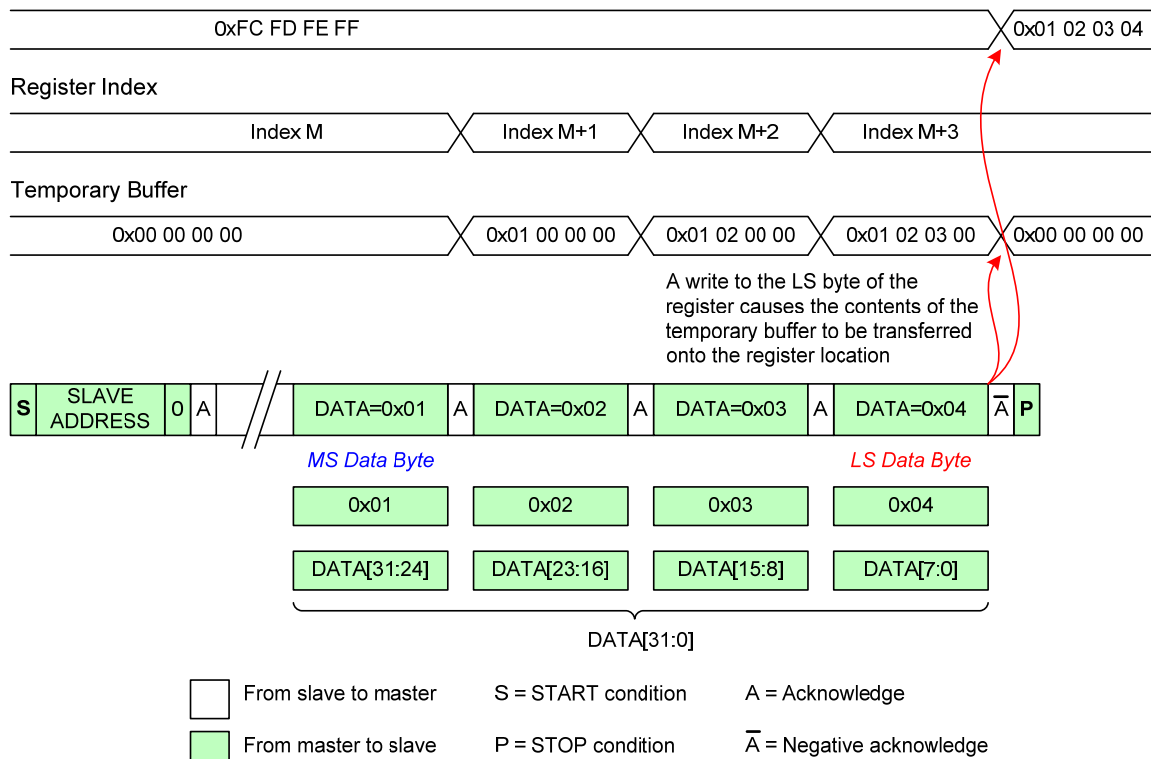


Figure 18 Example 32-bit Register Write

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

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

Table 1 CCI I/O Characteristics

Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
LOW level input voltage	V <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	$V_{OH}$	N/A	N/A	$0.8V_{DD}$		V
Output fall time from $V_{IHmin}$ to $V_{ILmax}$ with bus capacitance from 10 pF to 400 pF	$t_{OF}$	-	250	$20+0.1C_B$ Note 2	250	ns
Pulse width of spikes which shall be suppressed by the input filter	$t_{SP}$	N/A	N/A	0	50	ns
Input current each I/O pin with an input voltage between 0.1 $V_{DD}$ and 0.9 $V_{DD}$	$I_I$	-10	10	-10 Note 3	10 Note 3	$\mu A$
Input/Output capacitance (SDA)	$C_{I/O}$	-	8	-	8	pF
Input capacitance (SCL)	$C_I$	-	6	-	6	pF

711 Notes:

712 1. Maximum  $V_{IH} = V_{DDmax} + 0.5V$

713 2.  $C_B$  = capacitance of one bus line in pF

714 3. I/O pins of Fast-mode devices shall not obstruct the SDA and SCL line if  $V_{DD}$  is switched off

715

**Table 2 CCI Timing Specification**

Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
SCL clock frequency	$f_{SCL}$	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	0.4	-	0.6	-	$\mu s$
LOW period of the SCL clock	$t_{LOW}$	4.7	-	1.3	-	$\mu s$
HIGH period of the SCL clock	$t_{HIGH}$	4.0	-	0.6	-	$\mu s$
Setup time for a repeated START condition	$t_{SU;STA}$	4.7	-	0.6	-	$\mu s$
Data hold time	$t_{HD;DAT}$	0 Note 2	3.45 Note 3	0 Note 2	0.9 Note 3	$\mu s$
Data set-up time	$t_{SU;DAT}$	250	-	100 Note 4	-	ns
Rise time of both SDA and SCL signals	$t_R$	-	1000	$20+0.1C_B$ Note 5	300	ns
Fall time of both SDA and SCL signals	$t_F$	-	300	$20+0.1C_B$ Note 5	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	-	0.6	-	$\mu s$
Bus free time between a STOP	$t_{BUF}$	4.7	-	1.3	-	$\mu s$

Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
and START condition						
Capacitive load for each bus line	$C_B$	-	400	-	400	pF
Noise margin at the LOW level for each connected device (including hysteresis)	$V_{nL}$	$0.1V_{DD}$	-	$0.1V_{DD}$	-	V
Noise margin at the HIGH level for each connected device (including hysteresis)	$V_{nH}$	$0.2V_{DD}$	-	$0.2V_{DD}$	-	V

Notes:

1. All values referred to  $V_{IHmin} = 0.7V_{DD}$  and  $V_{ILmax} = 0.3V_{DD}$
2. A device shall internally provide a hold time of at least 300 ns for the SDA signal (referred to the  $V_{IHmin}$  of the SCL signal) to bridge the undefined region of the falling edge of SCL
3. The maximum  $t_{HD:DAT}$  has only to be met if the device does not the LOW period ( $t_{LOW}$ ) of the SCL signal
4. A Fast-mode I2C-bus device can be used in a Standard-mode I2C-bus system, but the requirement  $t_{SU:DAT} \geq 250$  ns shall be then met. This will be automatically the case if the device does not stretch the LOW period of the SCL signal. If such device does stretch the low period of SCL signal, it shall output the next data bit to the SDA line  $t_{rMAX} + t_{SU:DAT} = 1000 + 250 = 1250$  ns (according to the Standard-mode I2C bus specification) before the SCL line is released.
5.  $C_B$  = total capacitance of one bus line in pF.

The CCI timing is illustrated in Figure 19.

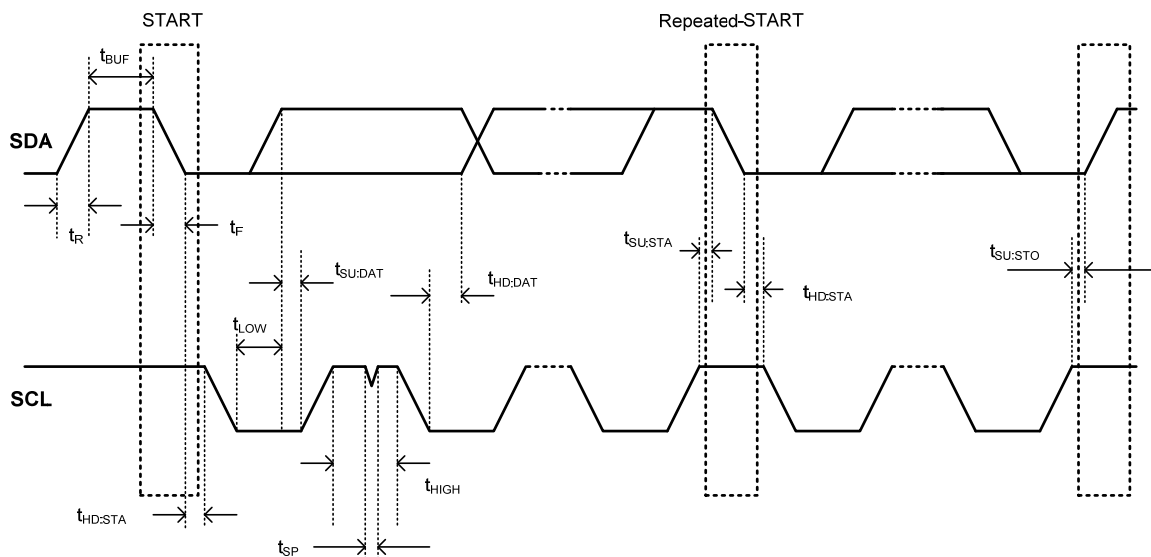


Figure 19 CCI Timing

## 7 Physical Layer

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

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

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

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

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

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

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

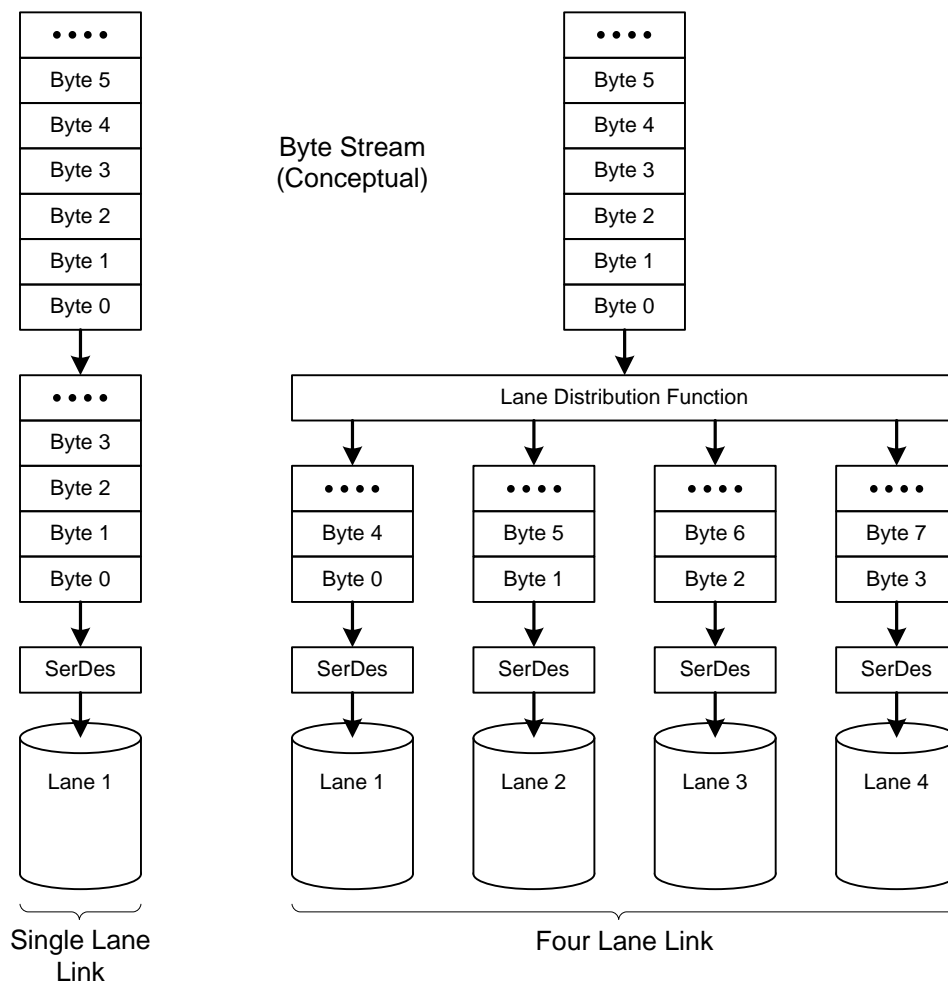
- Data Lane Module: Unidirectional slave, HS-RX, LP-RX, and a CIL-SFEN function
- Clock Lane Module: Unidirectional slave, HS-RX, LP-RX, and a CIL-SCNN function

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

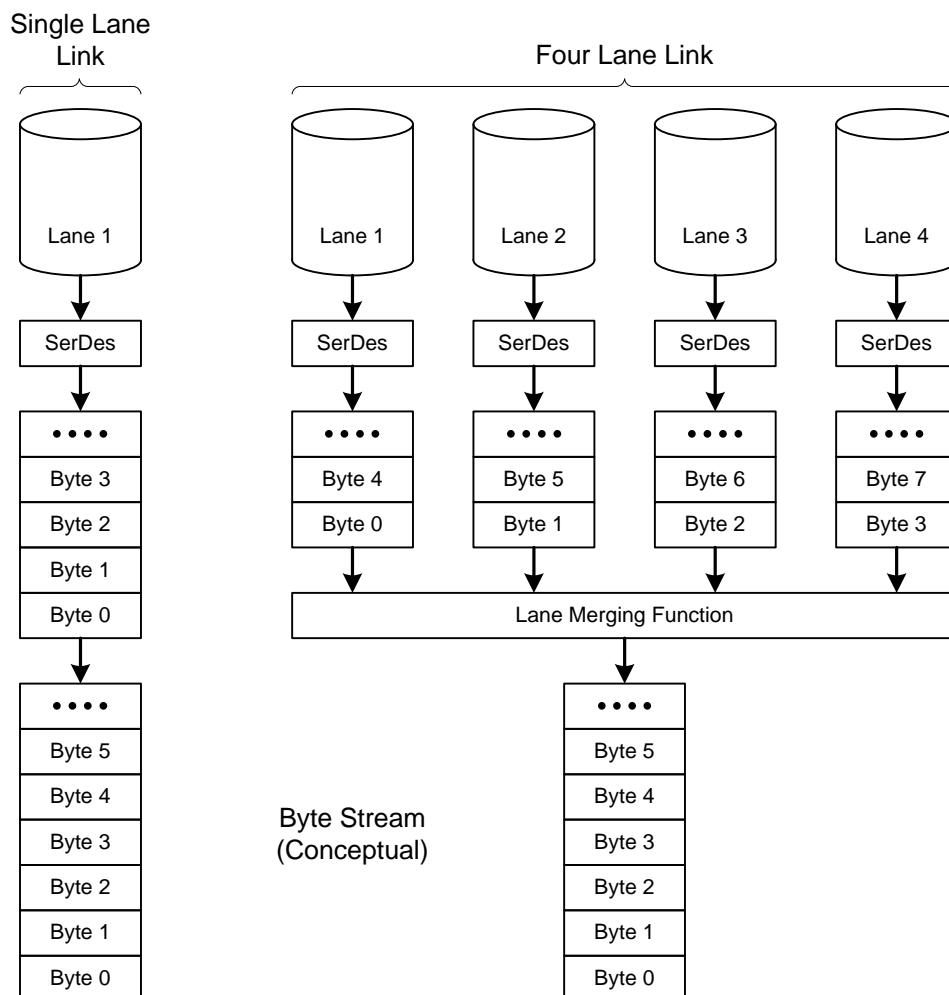
## 8 Multi-Lane Distribution and Merging

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

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



**Figure 20 Conceptual Overview of the Lane Distributor Function**



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

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

Examples:

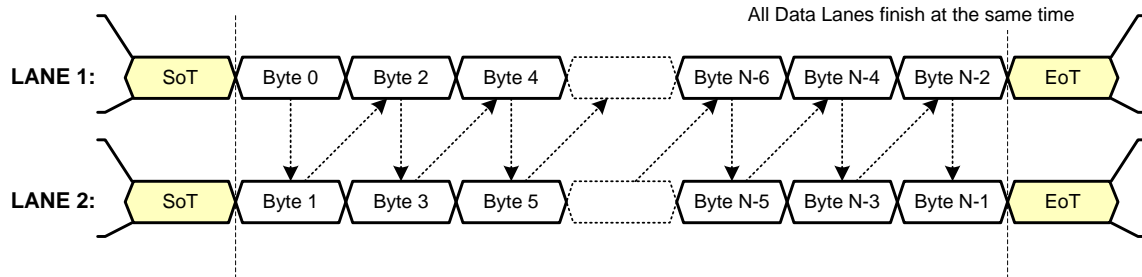
- 2-Lane system (Figure 22): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to Lane 1, byte 3 goes to Lane 2, byte 4 goes to Lane 1 and so on.
- 3-Lane system (Figure 23): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to Lane 3, byte 3 goes to Lane 1, byte 4 goes to Lane 2 and so on.
- 4-Lane system (Figure 24): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to Lane 3, byte 3 goes to Lane 4, byte 4 goes to Lane 1 and so on

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

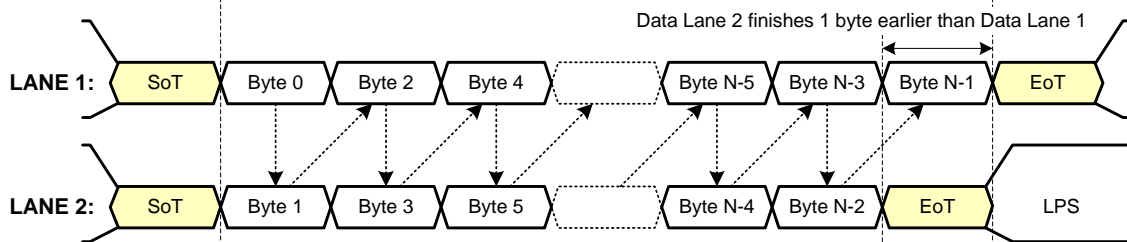


- 779 Each D-PHY data Lane operates autonomously.
- 780 Although multiple Lanes all start simultaneously with parallel “start packet” codes, they may complete the transaction at different times, sending “end packet” codes one cycle (byte) apart.
- 781
- 782 The N PHYs on the receiving end of the link collect bytes in parallel, and feed them into the Lane-merging layer. This reconstitutes the original sequence of bytes in the transmission, which can then be partitioned
- 783 into individual packets for the packet decoder layer.
- 784

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



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



**KEY:**

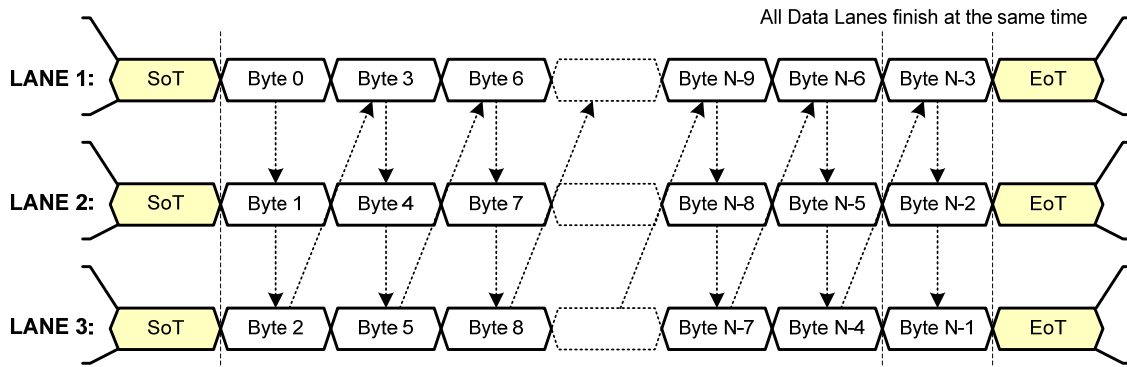
LPS – Low Power State

SoT – Start of Transmission

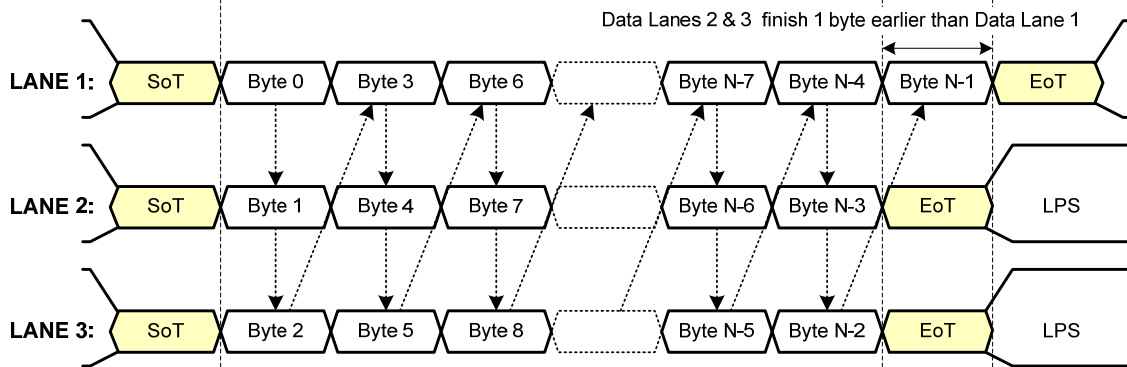
EoT – End of Transmission

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

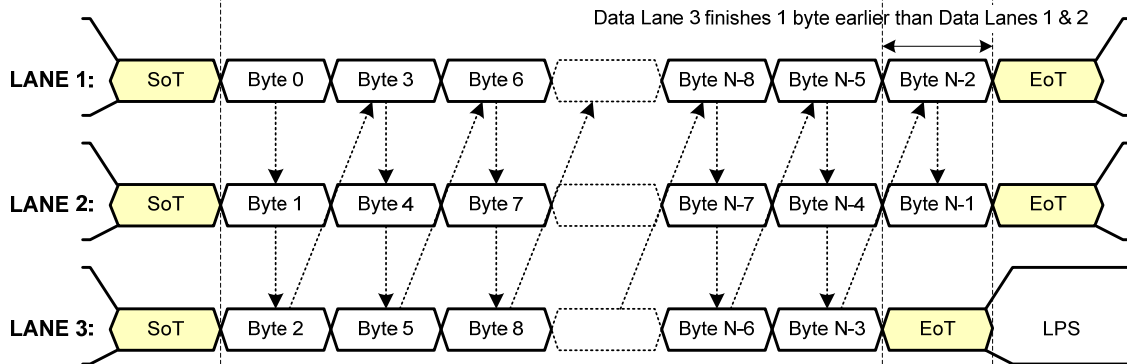
Number of Bytes,  $N$ , transmitted is an integer multiple of the number of lanes:



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



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



KEY:

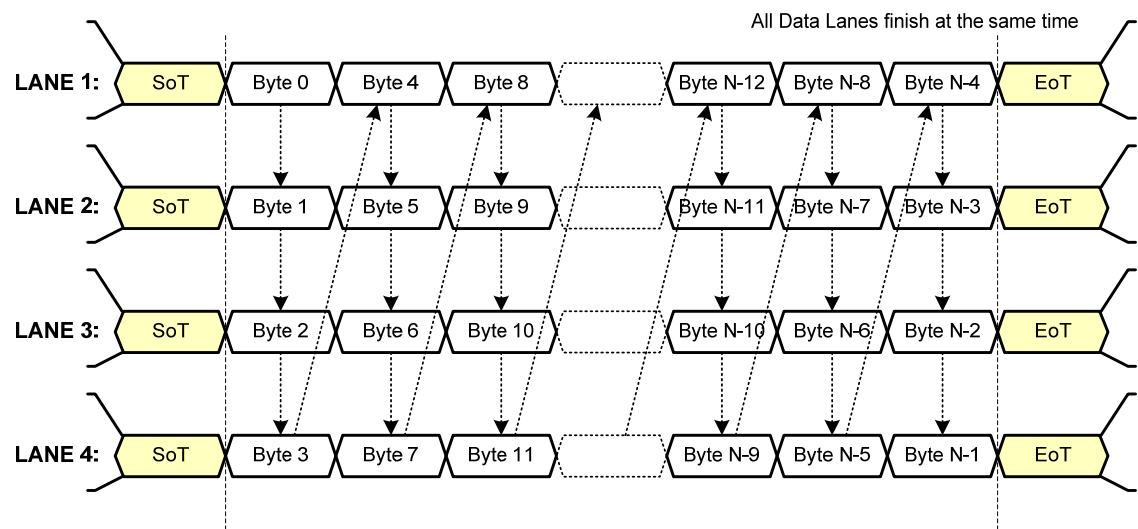
LPS – Low Power State

SoT – Start of Transmission

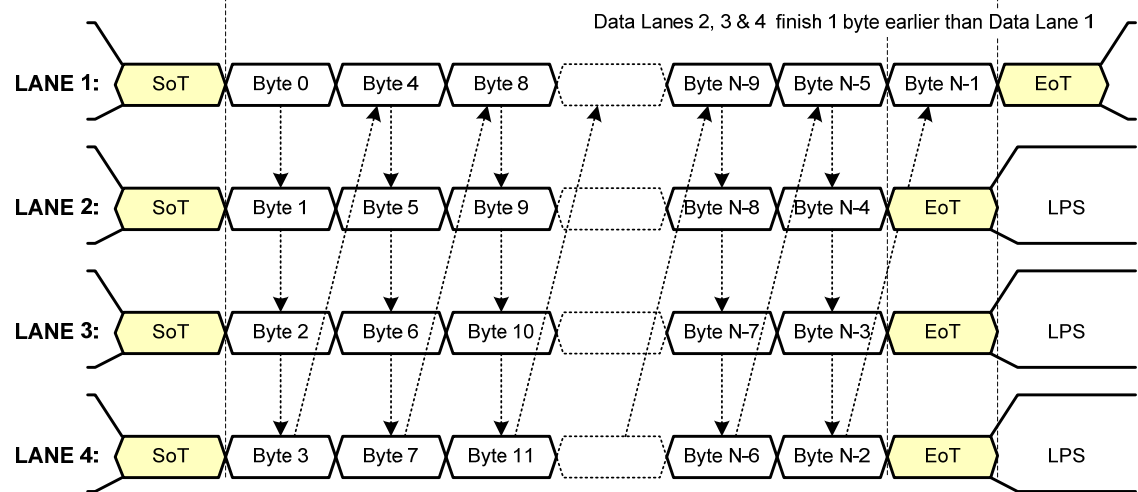
EoT – End of Transmission

Figure 23 Three Lane Multi-Lane Example

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



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



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

Figure 24 Four Lane Multi-Lane Example

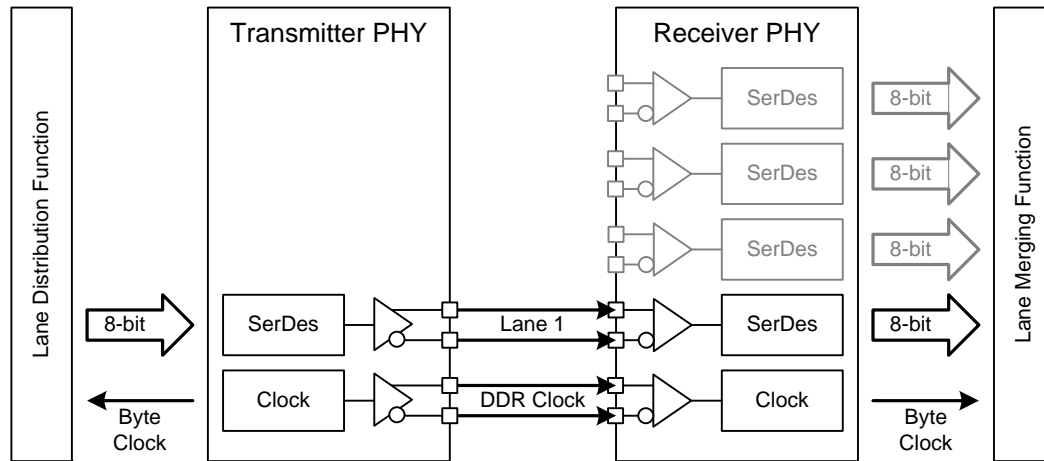
8.1 Multi-Lane Interoperability

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

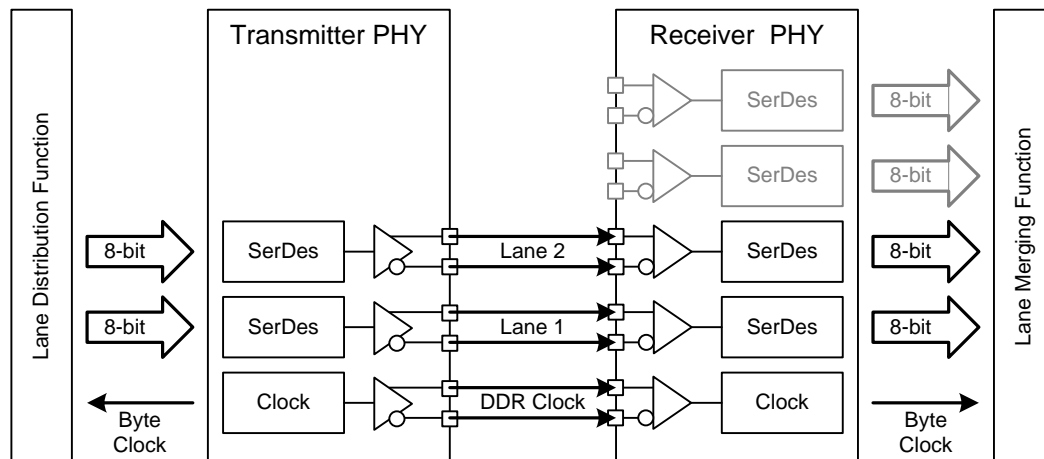
An "N" data Lane receiver shall be connected with an "M" data Lane transmitter, by CCI configuration of the Lane distribution and merging layers within the CSI-2 transmitter and receiver when more than one data Lane is used. Thus, a receiver with four data Lanes shall work with transmitters with one,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

Two cases:

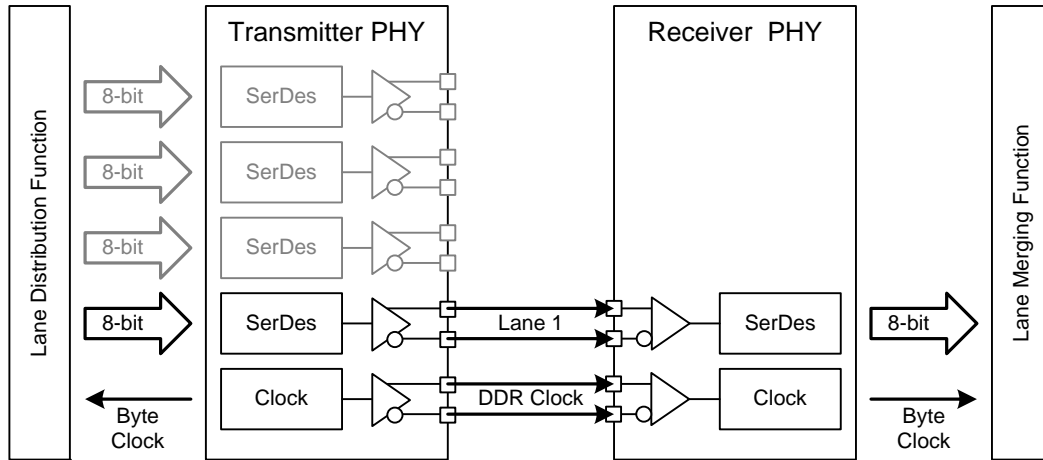
- If  $M \leq N$  then there is no loss of performance – the receiver has sufficient data Lanes to match the transmitter (Figure 25 and Figure 26).
- If  $M > N$  then there may be a loss of performance (e.g. frame rate) as the receiver has fewer data Lanes than the transmitter (Figure 27 and Figure 28).



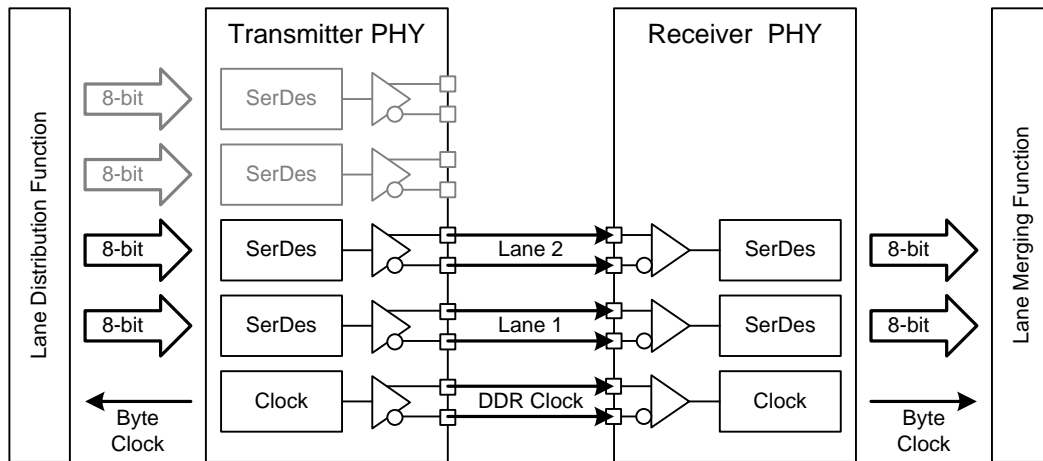
**Figure 25 One Lane Transmitter and Four Lane Receiver Example**



**Figure 26 Two Lane Transmitter and Four Lane Receiver Example**



**Figure 27 Four Lane Transmitter and One Lane Receiver Example**



**Figure 28 Four Lane Transmitter and Two Lane Receiver Example**

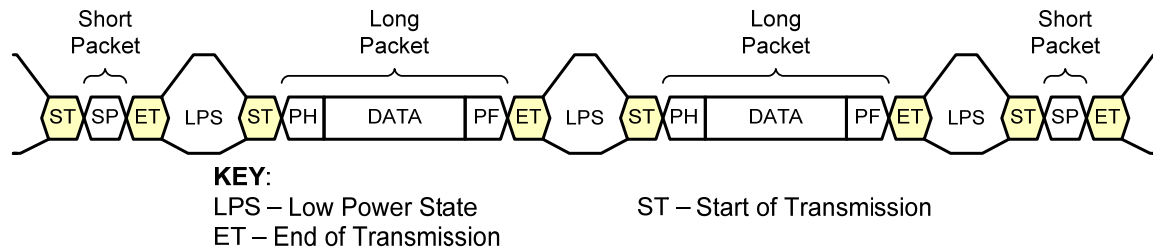
## 9 Low Level Protocol

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

Low Level Protocol Features:

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

**DATA:**



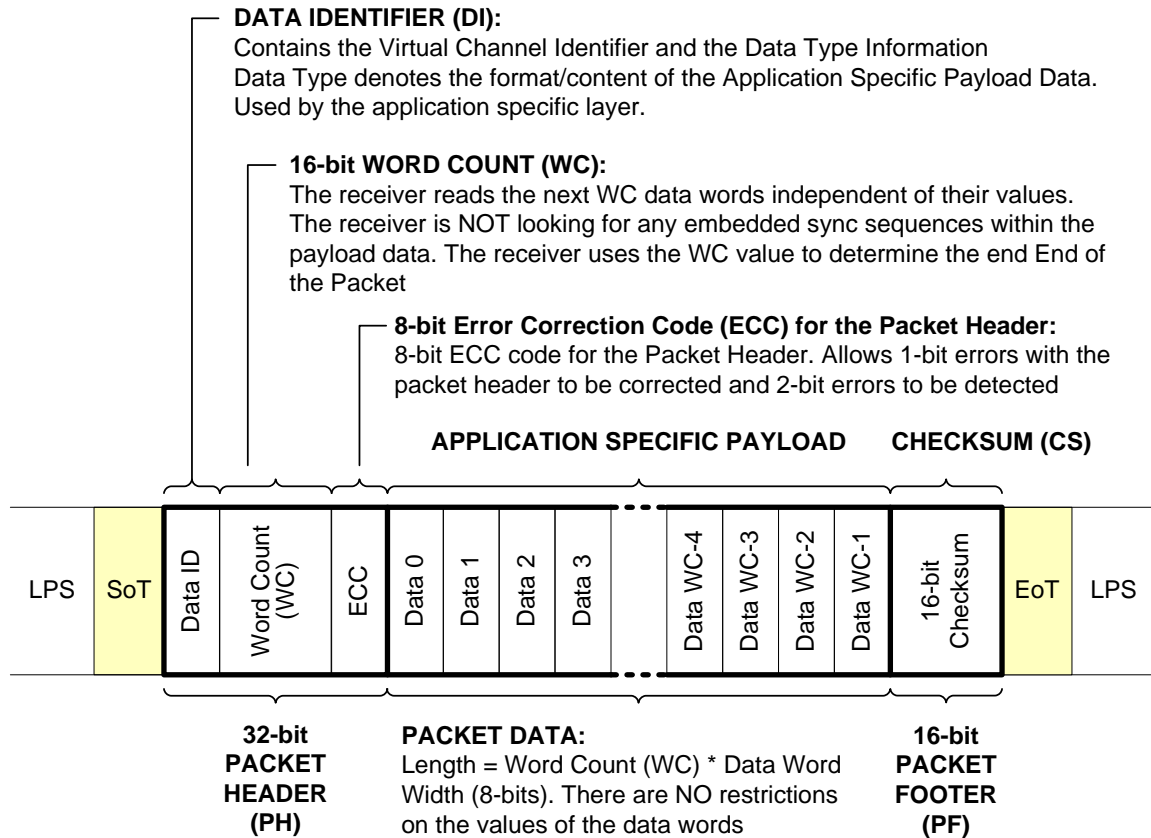
**Figure 29 Low Level Protocol Packet Overview**

### 9.1 Low Level Protocol Packet Format

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

#### 9.1.1 Low Level Protocol Long Packet Format

Figure 30 shows the structure of the Low Level Protocol Long Packet. A Long Packet shall be identified by Data Types 0x10 to 0x37. See Table 3 for a description of the Data Types. A Long Packet shall consist of three elements: a 32-bit Packet Header (PH), an application specific Data Payload with a variable number of 8-bit data words and a 16-bit Packet Footer (PF). The Packet Header is further composed of three elements: an 8-bit Data Identifier, a 16-bit Word Count field and an 8-bit ECC. The Packet footer has one element, a 16-bit checksum. See sections 9.2 through 9.5 for further descriptions of the packet elements.



**Figure 30 Long Packet Structure**

The Data Identifier defines the Virtual Channel for the data and the Data Type for the application specific payload data.

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.

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.

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.

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.

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.

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

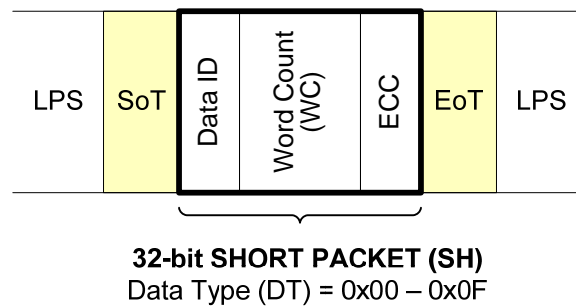
### 9.1.2 Low Level Protocol Short Packet Format

Figure 31 shows the structure of the Low Level Protocol Short Packet. A Short Packet shall be identified by Data Types 0x00 to 0x0F. See Table 3 for a description of the Data Types. A Short Packet shall contain only a Packet Header; a Packet Footer shall not be present. The Word Count field in the Packet Header shall be replaced by a Short Packet Data Field.

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

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

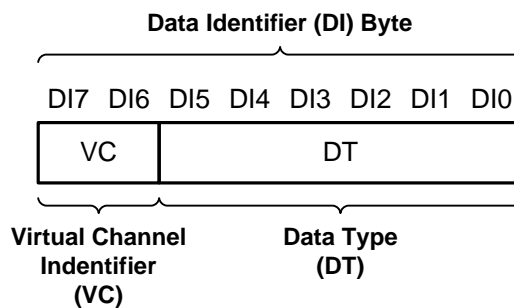
The Error Correction Code (ECC) byte allows single-bit errors to be corrected and 2-bit errors to be detected in the Short Packet.



**Figure 31 Short Packet Structure**

### 9.2 Data Identifier (DI)

The Data Identifier byte contains the Virtual Channel Identifier (VC) value and the Data Type (DT) value as illustrated in Figure 32. The Virtual Channel Identifier is contained in the two MS bits of the Data Identifier Byte. The Data Type value is contained in the six LS bits of the Data Identifier Byte.



**Figure 32 Data Identifier Byte**

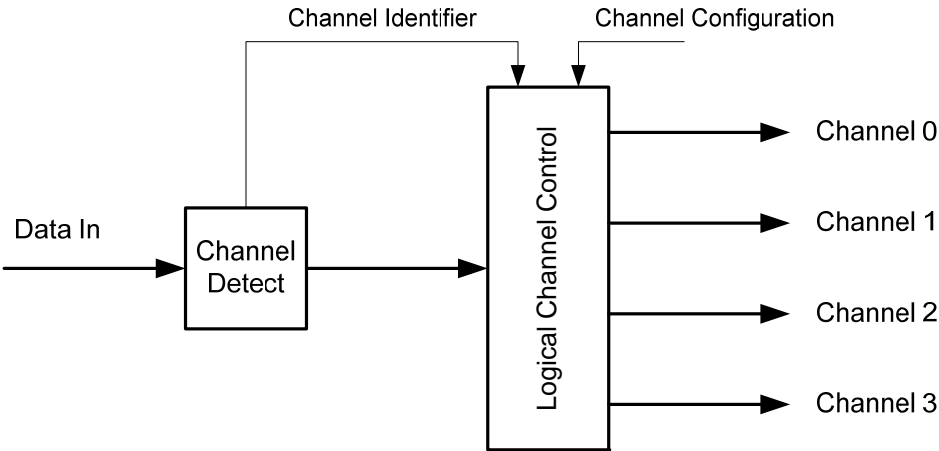
### 9.3 Virtual Channel Identifier

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

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

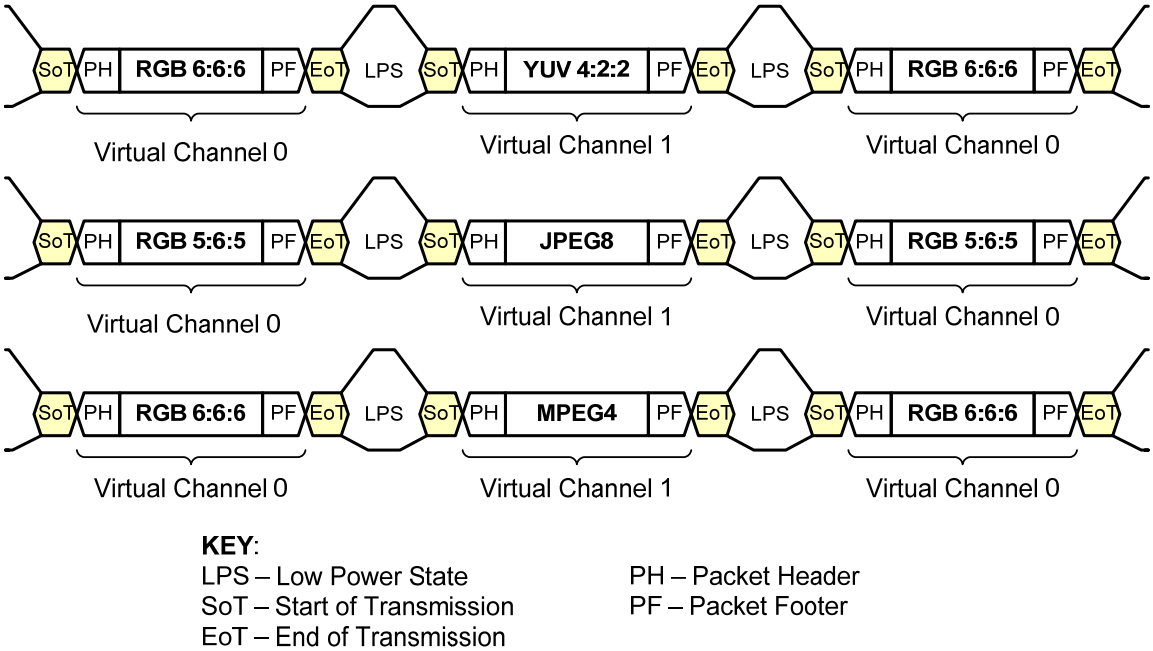


channel. A maximum of four data streams is supported; valid channel identifiers are 0 to 3. The virtual channel identifiers in the peripherals should be programmable to allow the host processor to control how the data streams are de-multiplexed. The principle of logical channels is presented in the Figure 33.



**Figure 33 Logical Channel Block Diagram (Receiver)**

Figure 34 illustrates an example of data streams utilizing virtual channel support.



**Figure 34 Interleaved Video Data Streams Examples**

## 9.4 Data Type (DT)

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

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

896 For details on the short packet data type classes refer to section 9.8.

897 For details on the five long packet data type classes refer to section 11.

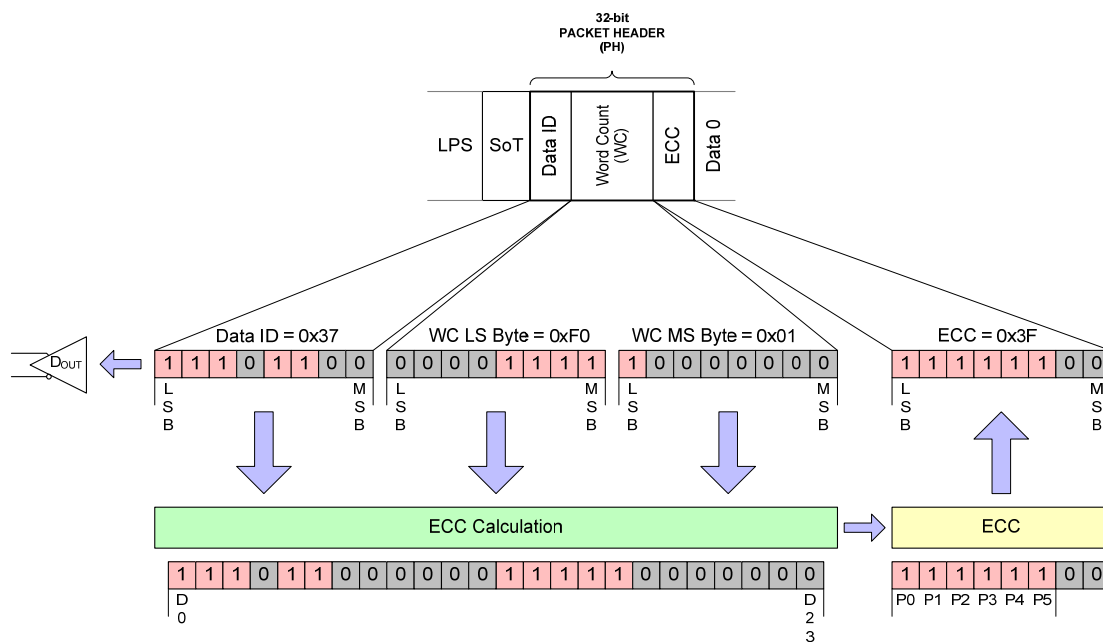
898 **Table 3 Data Type Classes**

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

## 899 9.5 Packet Header Error Correction Code

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

905 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])  
 906 to D[15:8] and the Word Count MS Byte (WC[15:8]) to D[23:16]. This mapping is shown in Figure 35,  
 907 which also serves as an ECC calculation example.



908 **Figure 35 24-bit ECC Generation Example**

### 9.5.1 General Hamming Code Applied to Packet Header

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

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

The result of appending the computed parity bits to the data bits is called the Hamming code word. The size of the code word  $c$  is obviously  $d + p$ , and a Hamming code word is described by the ordered set  $(c, d)$ . A Hamming code word is generated by multiplying the data bits by a generator matrix  $\mathbf{G}$ . This multiplication's result is called the code word vector  $(c_1, c_2, c_3, \dots, c_n)$ , consisting of the original data bits and the calculated parity bits. The generator matrix  $\mathbf{G}$  used in constructing Hamming codes consists of  $\mathbf{I}$  (the identity matrix) and a parity generation matrix  $\mathbf{A}$ :

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

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

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

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

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

### 9.5.2 Hamming-modified Code

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

**Table 4 ECC Syndrome Association Matrix**

	d2d1d0							
d5d4d3	0b000	0b001	0b010	0b011	0b100	0b101	0b110	0b111
0b000	0x07	0x0B	0x0D	0x0E	0x13	0x15	0x16	0x19
0b001	0x1A	0x1C	0x23	0x25	0x26	0x29	0x2A	0x2C
0b010	0x31	0x32	0x34	0x38	0x1F	0x2F	0x37	0x3B
0b011	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

	d2d1d0							
d5d4d3	0b000	0b001	0b010	0b011	0b100	0b101	0b110	0b111
0b110	0x8C	0x91	0x92	0x94	0x98	0xA1	0xA2	0xA4
0b111	0xA8	0xB0	0xC1	0xC2	0xC4	0xC8	0xD0	0xE0

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

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

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

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

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

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

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

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

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

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

**Table 5 ECC Parity Generation Rules**

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

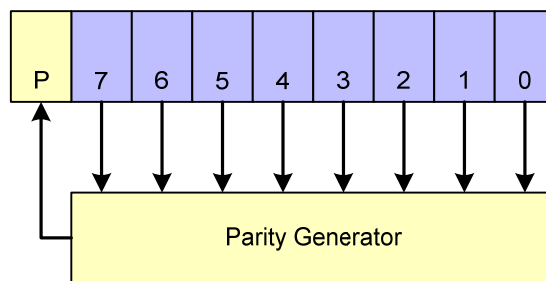
Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
11	0	0	1	0	0	1	0	1	0x25
12	0	0	1	0	0	1	1	0	0x26
13	0	0	1	0	1	0	0	1	0x29
14	0	0	1	0	1	0	1	0	0x2A
15	0	0	1	0	1	1	0	0	0x2C
16	0	0	1	1	0	0	0	1	0x31
17	0	0	1	1	0	0	1	0	0x32
18	0	0	1	1	0	1	0	0	0x34
19	0	0	1	1	1	0	0	0	0x38
20	0	0	0	1	1	1	1	1	0x1F
21	0	0	1	0	1	1	1	1	0x2F
22	0	0	1	1	0	1	1	1	0x37
23	0	0	1	1	1	0	1	1	0x3B
24	0	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

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
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
50	1	0	0	1	0	0	1	0	0x92
51	1	0	0	1	0	1	0	0	0x94
52	1	0	0	1	1	0	0	0	0x98
53	1	0	1	0	0	0	0	1	0xA1
54	1	0	1	0	0	0	1	0	0xA2
55	1	0	1	0	0	1	0	0	0xA4
56	1	0	1	0	1	0	0	0	0xA8
57	1	0	1	1	0	0	0	0	0xB0
58	1	1	0	0	0	0	0	1	0xC1
59	1	1	0	0	0	0	1	0	0xC2
60	1	1	0	0	0	1	0	0	0xC4
61	1	1	0	0	1	0	0	0	0xC8
62	1	1	0	1	0	0	0	0	0xD0
63	1	1	1	0	0	0	0	0	0xE0

### 9.5.3 ECC Generation on TX Side

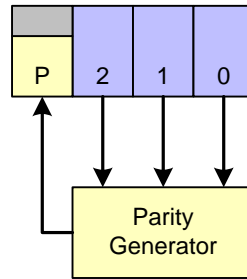
This is an informative section.

The ECC can be easily implemented using a parallel approach as depicted in Figure 36 for a 64-bit header.



**Figure 36 64-bit ECC Generation on TX Side**

And Figure 37 for a 24-bit header:



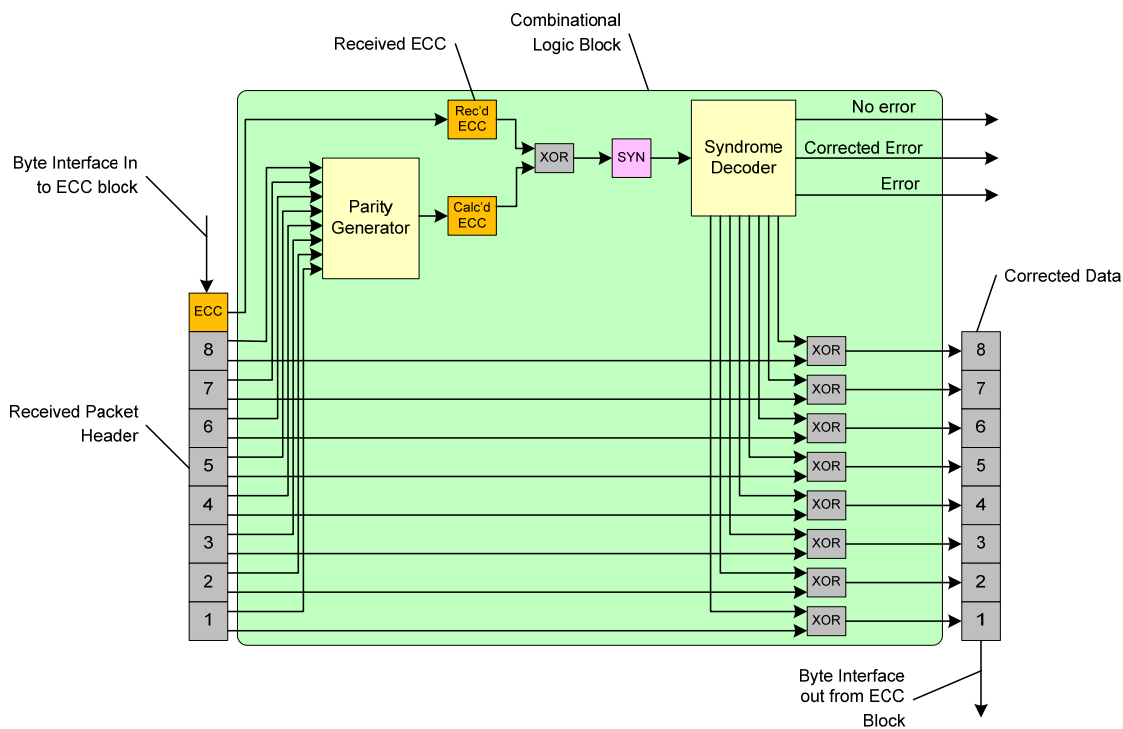
**Figure 37 24-bit ECC Generation on TX Side**

The parity generators are based on Table 5.

$$\text{e.g. } P_{3_{24\text{-bit}}} = D1 \wedge D2 \wedge D3 \wedge D7 \wedge D8 \wedge D9 \wedge D13 \wedge D14 \wedge D15 \wedge D19 \wedge D20 \wedge D21 \wedge D23$$

### 9.5.4 Applying ECC on RX Side

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



**Figure 38 64-bit ECC on RX Side Including Error Correction**

Decoding the syndrome has three aspects:

- Finding if the packet has any errors (if syndrome is 0, no errors are present)
- Checking if a single error has occurred by searching Table 5, if the syndrome is one of the entries in the table, then a single bit error has occurred and the corresponding bit is affected, thus this position in the data stream needs to be complemented. Also, if the syndrome is one of the rows of the identity matrix I, then one of the parity bits are in error. If the syndrome cannot be identified,

980 then a higher order error has occurred and the error flag will be set (the stream is corrupted and  
981 cannot be restored).

982 • Correcting the single error detected, as indicated above.

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

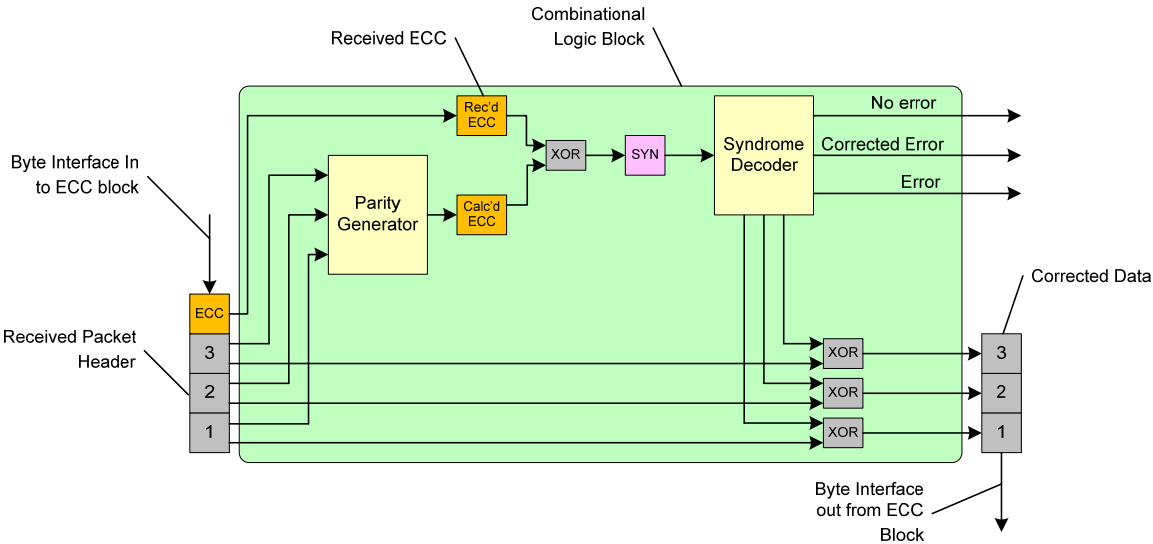


Figure 39 24-bit ECC on RX side Including Error Correction

### 9.6 Checksum Generation

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

991 The transmission of the checksum is illustrated in Figure 40.

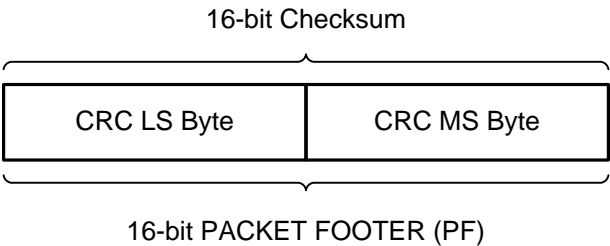
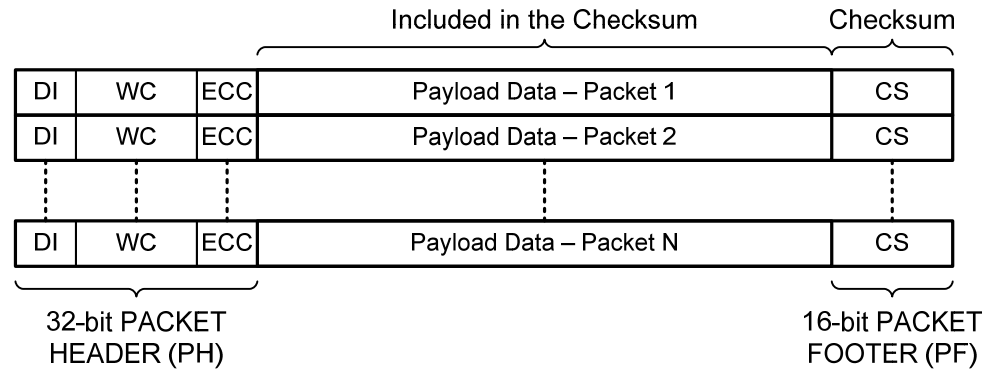


Figure 40 Checksum Transmission

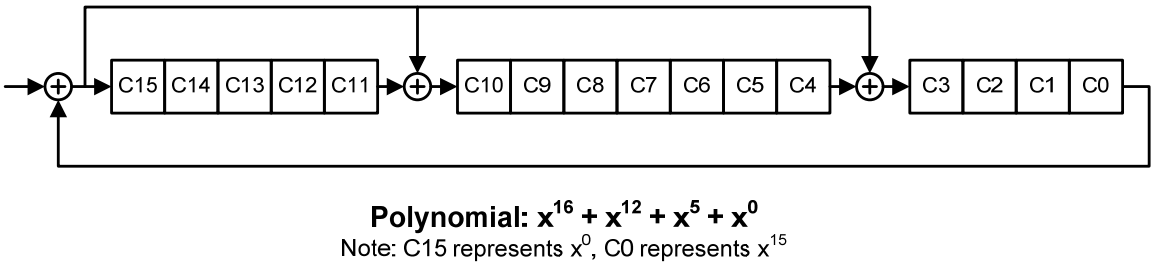
994 The 16-bit checksum sequence is transmitted as part of the Packet Footer. When the Word Count is zero,  
995 the CRC shall be 0xFFFF.





**Figure 41 Checksum Generation for Packet Data**

The definition of a serial CRC implementation is presented in Figure 42. The CRC implementation shall be functionally equivalent with the C code presented in Figure 43. The CRC shift register is initialized to 0xFFFF at the beginning of each packet. After all payload data has passed through the CRC circuitry, the CRC circuitry contains the checksum. The 16-bit checksum produced by the C code in Figure 43 equals the final contents of the C[15:0] shift register shown in Figure 42. The checksum is then sent over CSI-2 bus to the receiver to verify that no errors have occurred in the transmission.



**Figure 42 Definition of 16-bit CRC Shift Register**

```

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

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

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

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

    return (unsigned short)(crc);
}

```

**Figure 43 16-bit CRC Software Implementation Example**

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

```

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

```

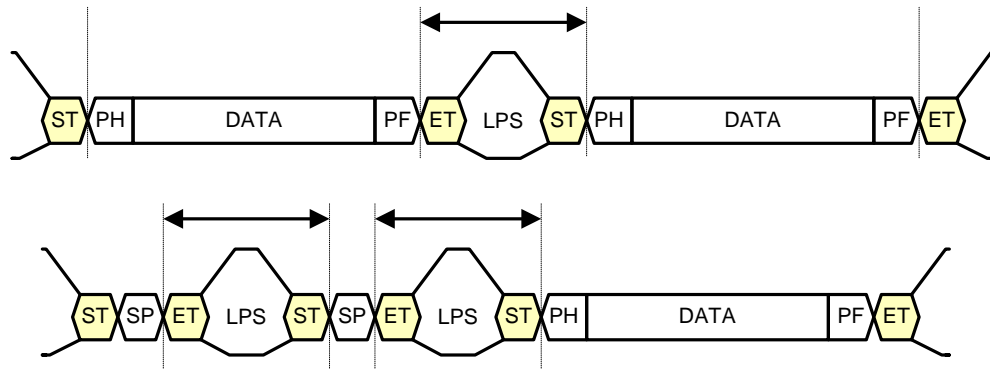
## 9.7 Packet Spacing

Between Low Level Protocol packets there must always be a transition into and out of the Low Power State (LPS). Figure 44 illustrates the packet spacing with the LPS.

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

**SHORT / LONG PACKET SPACING:**

Variable - always a LPS between packets

**KEY:**

LPS – Low Power State  
 ST – Start of Transmission  
 ET – End of Transmission

PH – Packet Header  
 PF – Packet Footer  
 SP – Short Packet

**Figure 44 Packet Spacing****9.8 Synchronization Short Packet Data Type Codes**

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

**Table 6 Synchronization Short Packet Data Type Codes**

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

**9.8.1 Frame Synchronization Packets**

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

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

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

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

- 1043 • Frame number is always zero – frame number is inoperative.
- 1044 • Frame number increments by 1 for every FS packet with the same Virtual Channel and is
- 1045 periodically reset to one e.g. 1, 2, 1, 2, 1, 2, 1, 2 or 1, 2, 3, 4, 1, 2, 3, 4

1046 The frame number must be a non-zero value.

## 1047 **9.8.2 Line Synchronization Packets**

1048 Line synchronization packets are optional.

1049 For Line Start (LS) and Line End (LE) synchronization packets the Short Packet Data Field shall contain a  
 1050 16-bit line number. This line number shall be the same for the LS and LE packets corresponding to a given  
 1051 line. Line numbers are logical line numbers and are not necessarily equal to the physical line numbers

1052 The 16-bit line number, when used, shall be non-zero to distinguish it from the case where line number is  
 1053 inoperative and remains set to zero.

1054 The behavior of the 16-bit line number shall be as one of the following:

- 1055 • Line number is always zero – line number is inoperative.
- 1056 • Line number increments by one for every LS packet within the same Virtual Channel and the same  
 1057 Data Type. The line number is periodically reset to one for the first LS packet after a FS packet.  
 1058 The intended usage is for progressive scan (non- interlaced) video data streams. The line number  
 1059 must be a non-zero value.
- 1060 • Line number increments by the same arbitrary step value greater than one for every LS packet  
 1061 within the same Virtual Channel and the same Data Type. The line number is periodically reset to  
 1062 a non-zero arbitrary start value for the first LS packet after a FS packet. The arbitrary start value  
 1063 may be different between successive frames. The intended usage is for interlaced video data  
 1064 streams.

## 1065 **9.9 Generic Short Packet Data Type Codes**

1066 Table 7 lists the Generic Short Packet Data Types.

1067 **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

1068 The intention of the Generic Short Packet Data Types is to provide a mechanism for including timing  
 1069 information for the opening/closing of shutters, triggering of flashes, etc within the data stream. The intent  
 1070 of the 16-bit User defined data field in the generic short packets is to pass a data type value and a 16-bit

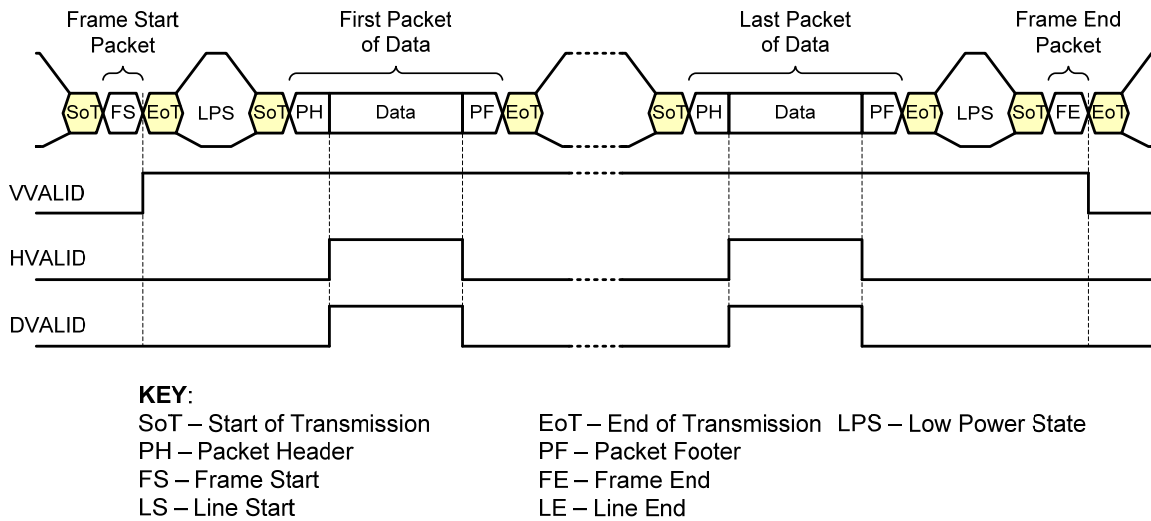
1071 data value from the transmitter to application layer in the receiver. The CSI-2 receiver shall pass the data  
 1072 type value and the associated 16-bit data value to the application layer.

### 1073 9.10 Packet Spacing Examples

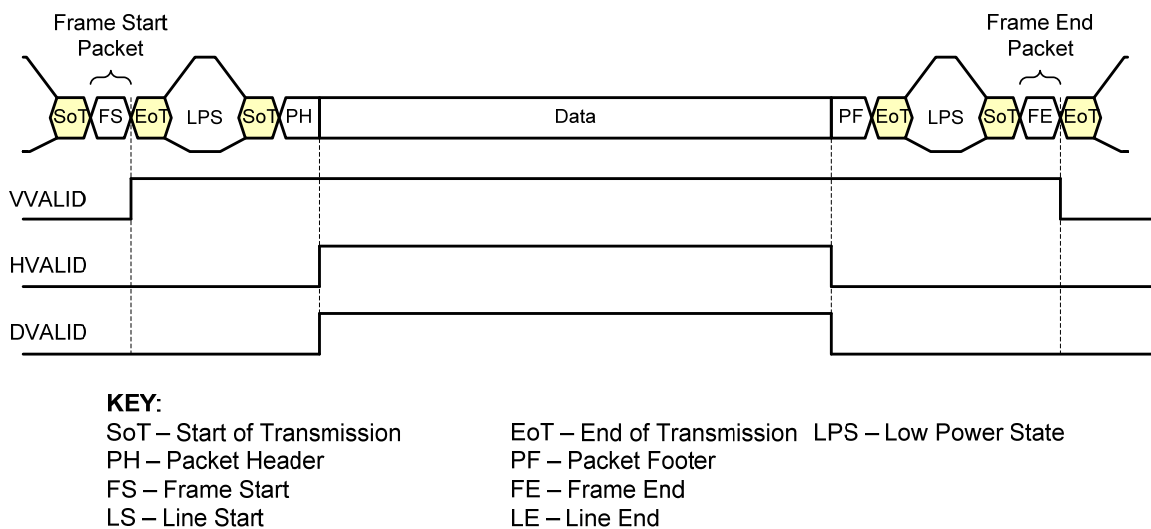
1074 Packets are separated by an EoT, LPS, SoT sequence as defined in [MIPI01].

1075 Figure 45 and Figure 46 contain examples of data frames composed of multiple packets and a single  
 1076 packet, respectively.

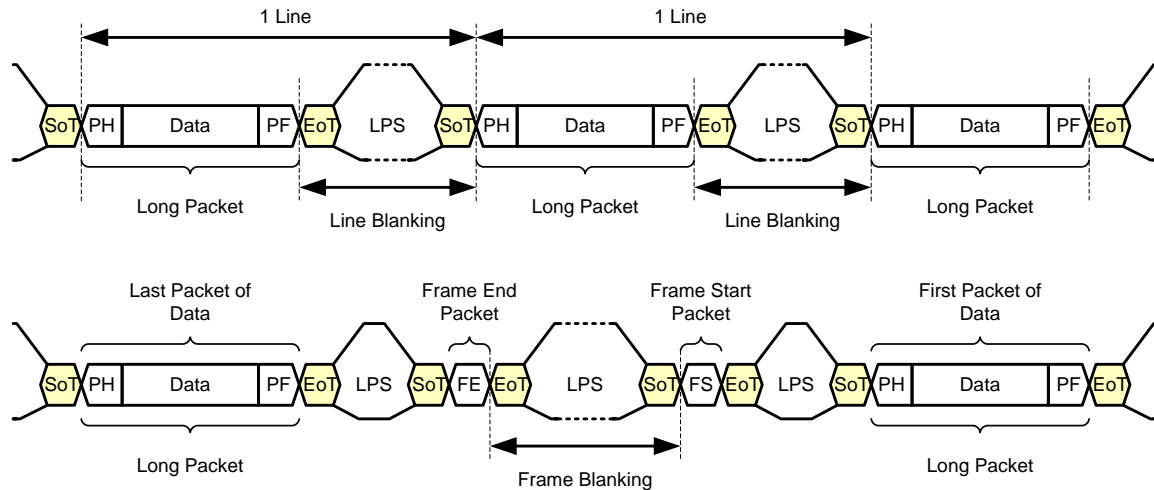
1077 Note that the VVALID, HVALID and DVALID signals in the figures in this section are only concepts to help  
 1078 illustrate the behavior of the frame start/end and line start/end packets. The VVALID, HVALID and  
 1079 DVALID signals do not form part of the specification.



1080  
 1081  
 1082 **Figure 45 Multiple Packet Example**



1083  
 1084 **Figure 46 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

**Figure 47 Line and Frame Blanking Definitions**

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

The period between the Frame End packet in frame N and the Frame Start packet in frame N+1 is called the Frame Blanking Period (Figure 47).

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

Frame Start and Frame End packets shall be used.

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

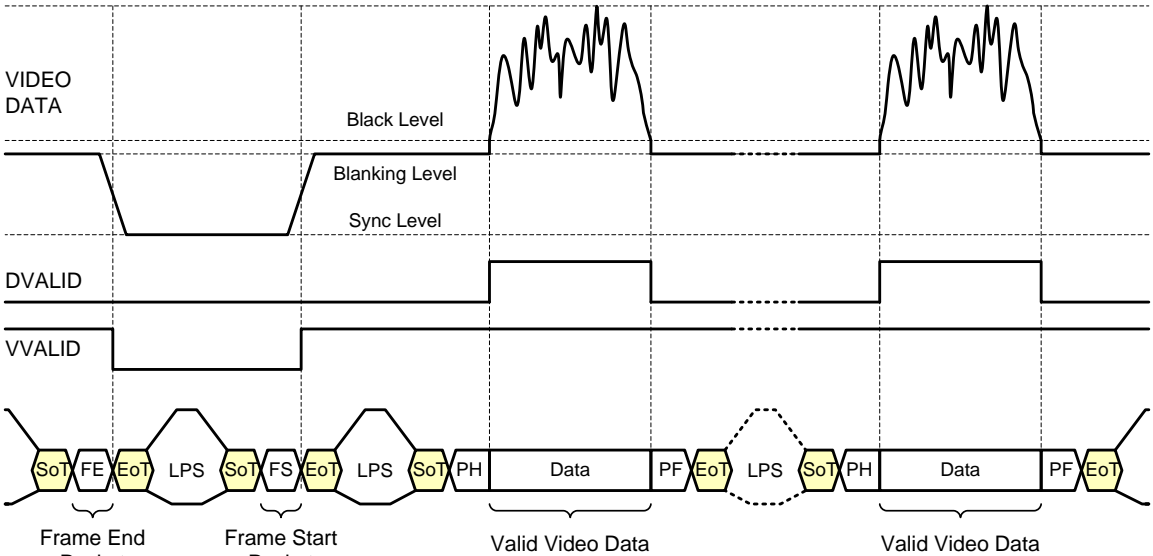
- The Frame Start packet to first data packet spacing should be as close as possible to the minimum packet spacing
- The last data packet to Frame End packet spacing should be as close as possible to the minimum packet spacing

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

The positions of the Frame Start and Frame End packets can be varied within the Frame Blanking Period in order to provide pixel level accurate vertical synchronization timing information. See Figure 48.

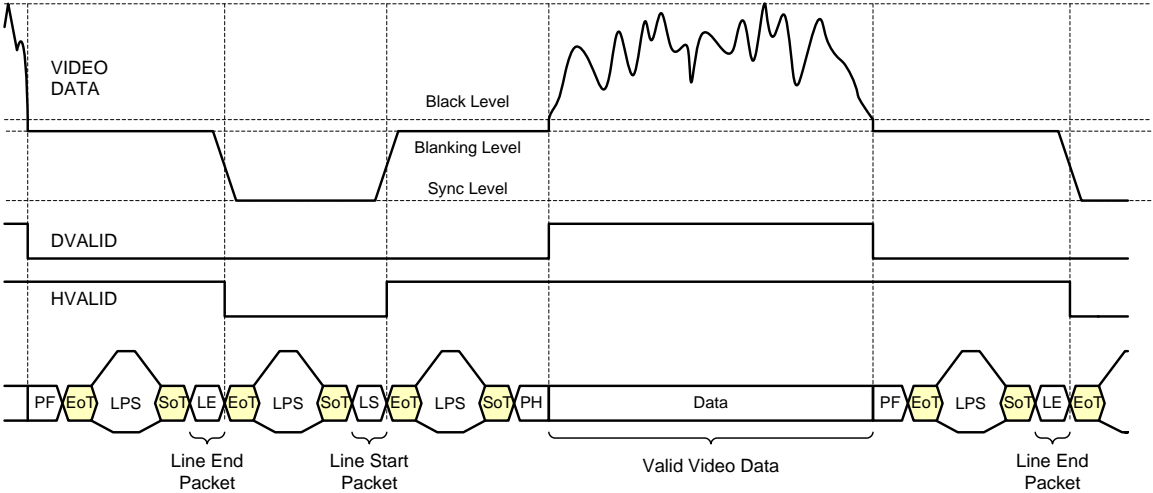
Line Start and Line End packets shall be used for pixel level accurate horizontal synchronization timing information.

1108 The positions of the Line Start and Line End packets, if present, can be varied within the Line Blanking  
1109 Period in order to provide pixel accurate horizontal synchronization timing information. See Figure 49.



1110  
1111  
1112

Figure 48 Vertical Sync Example



1113  
1114  
1115

Figure 49 Horizontal Sync Example

1116 **9.11 Packet Data Payload Size Rules**

1117 For YUV, RGB or RAW data types, one long packet shall contain one line of image data. Each long packet  
1118 of the same Data Type shall have equal length when packets are within the same Virtual Channel and when  
1119 packets are within the same frame. An exception to this rule is the YUV420 data type which is defined in  
1120 section 11.2.2.

1121 For User Defined Byte-based Data Types, long packets can have arbitrary length. The spacing between  
1122 packets can also vary.

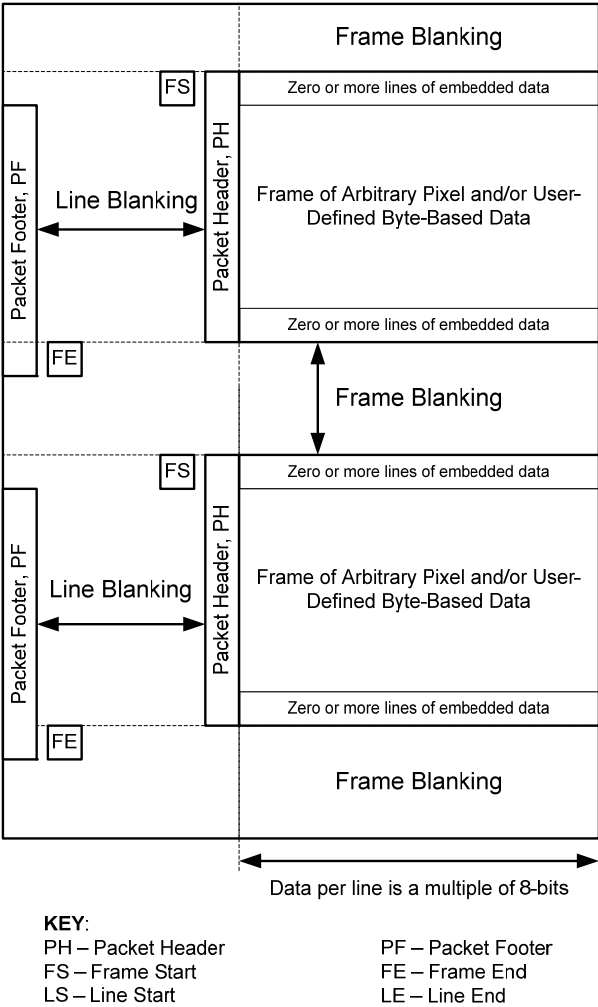
The total size of data within a long packet for all data types shall be a multiple of eight bits. However, it is also possible that a data type's payload data transmission format, as defined elsewhere in this specification, imposes additional constraints on payload size. In order to meet these constraints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

**9.12 Frame Format Examples**

This is an informative section.

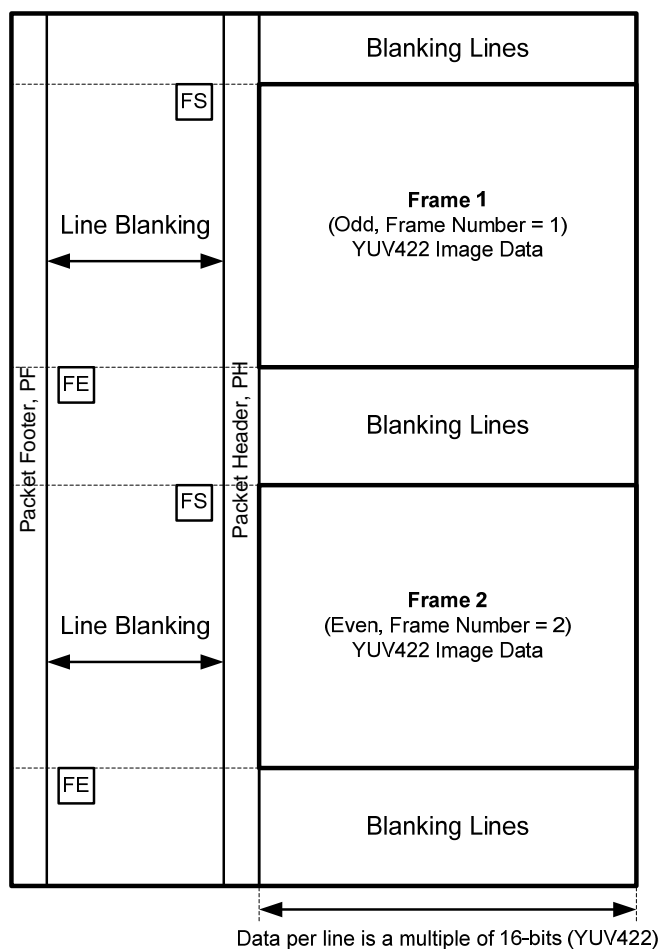
This section contains three examples to illustrate how the CSI-2 features can be used.

- General Frame Format Example, Figure 50
- Digital Interlaced Video Example, Figure 51
- Digital Interlaced Video with accurate synchronization timing information, Figure 52



**Figure 50 General Frame Format Example**





**KEY:**

PH – Packet Header

FS – Frame Start

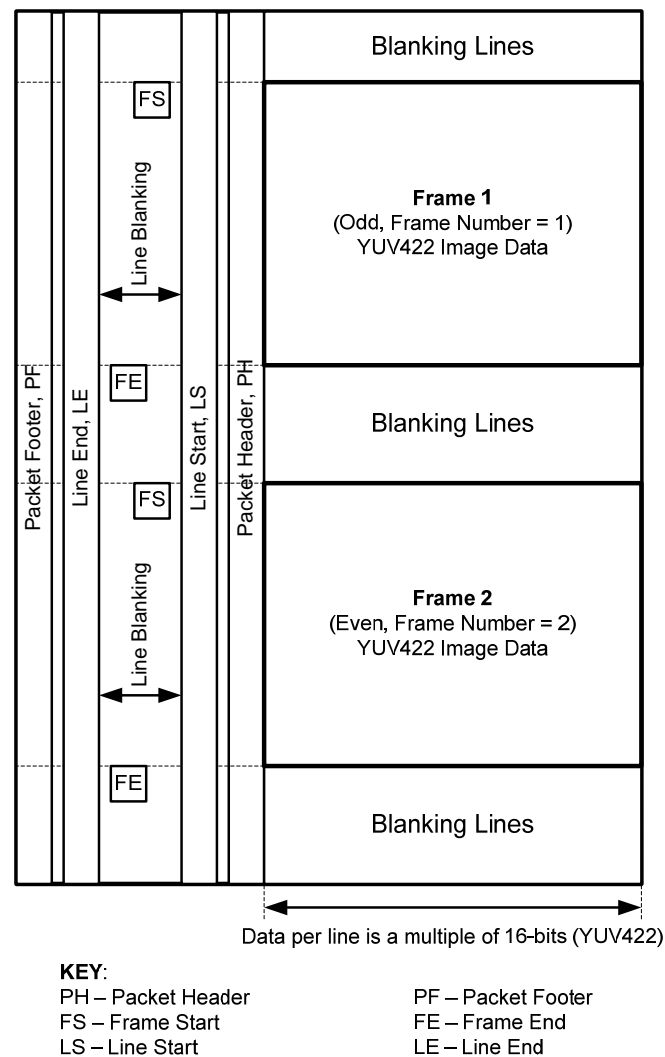
LS – Line Start

PF – Packet Footer

FE – Frame End

LE – Line End

### Figure 51 Digital Interlaced Video Example



**Figure 52 Digital Interlaced Video with Accurate Synchronization Timing Information**

### 9.13 Data Interleaving

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

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

- Data Type
- Virtual Channel Identifier

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

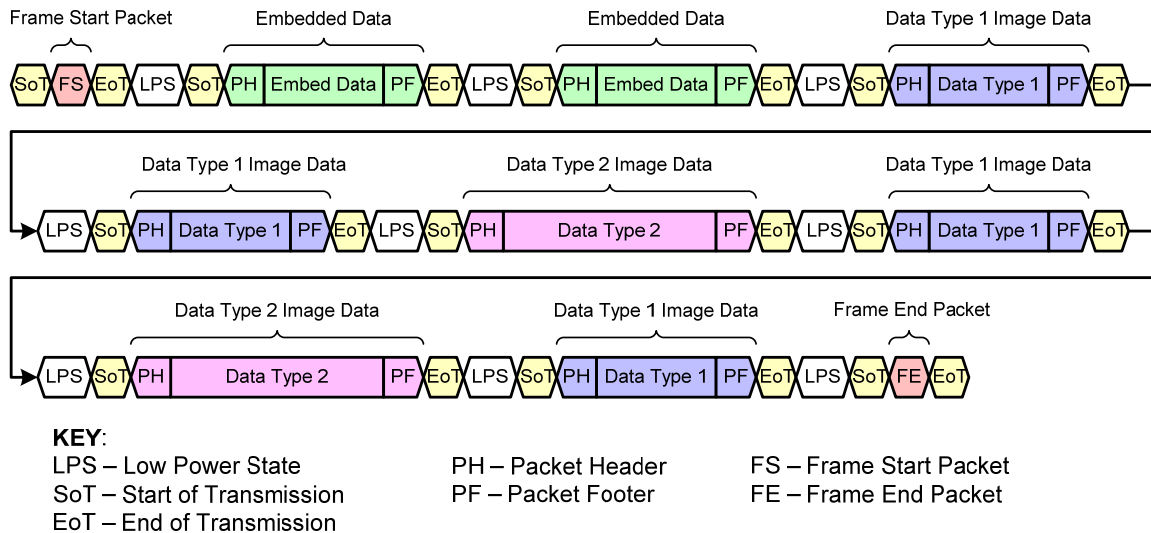
#### 9.13.1 Data Type Interleaving

The Data Type value uniquely defines the data format for that packet of data. The receiver uses the Data Type value in the packet header to de-multiplex data packets containing different data formats as illustrated in Figure 53. Note, in the figure the Virtual Channel Identifier is the same in each Packet Header.

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

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

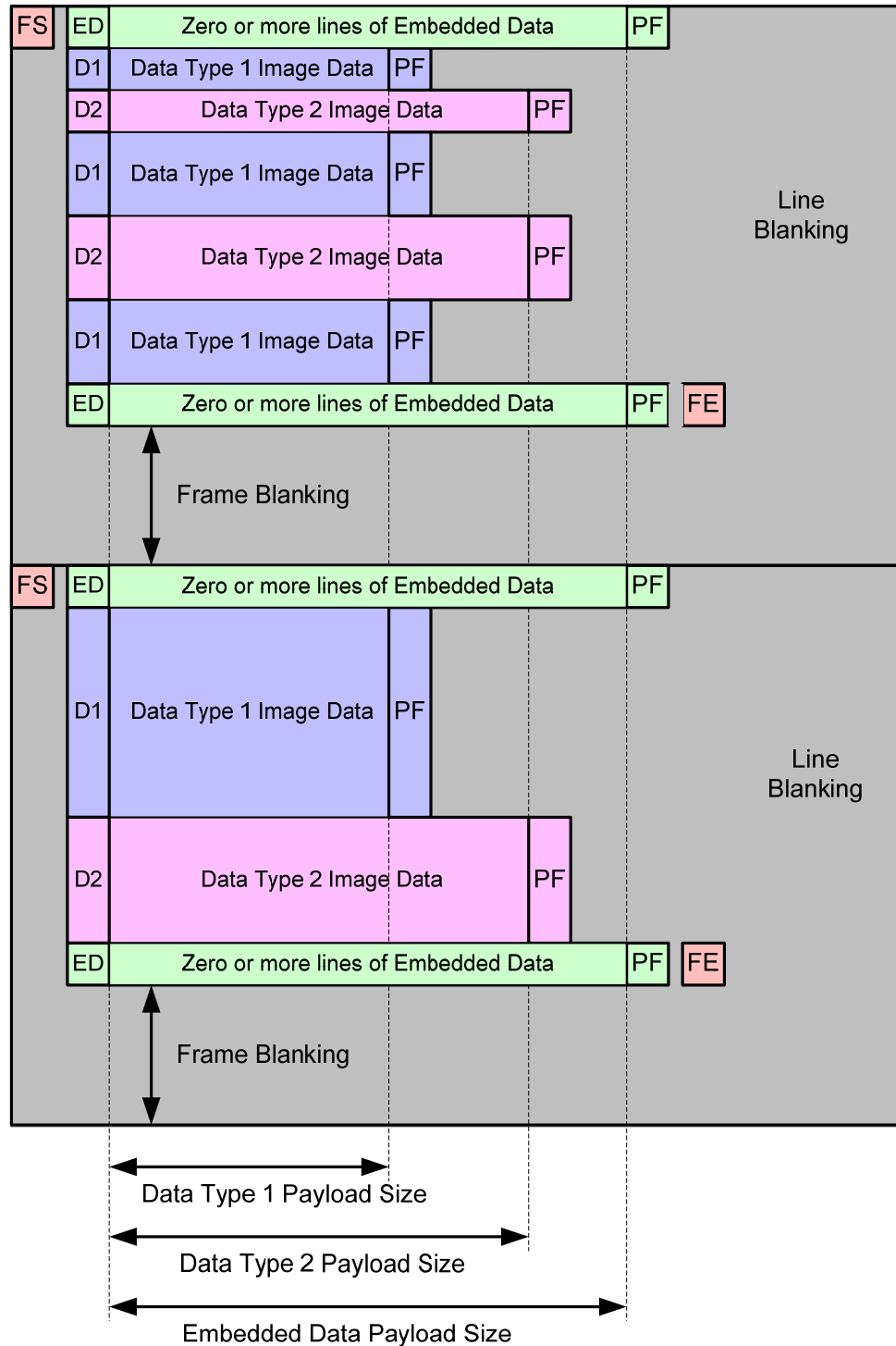
Data formats are defined further in section 11.



**Figure 53 Interleaved Data Transmission using Data Type Value**

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

Packets of different data types may be interleaved at either the packet level as illustrated in Figure 54 or the frame level as illustrated in Figure 55. Data formats are defined in section 11.

**KEY:**

LPS – Low Power State

FS – Frame Start

FE – Frame End

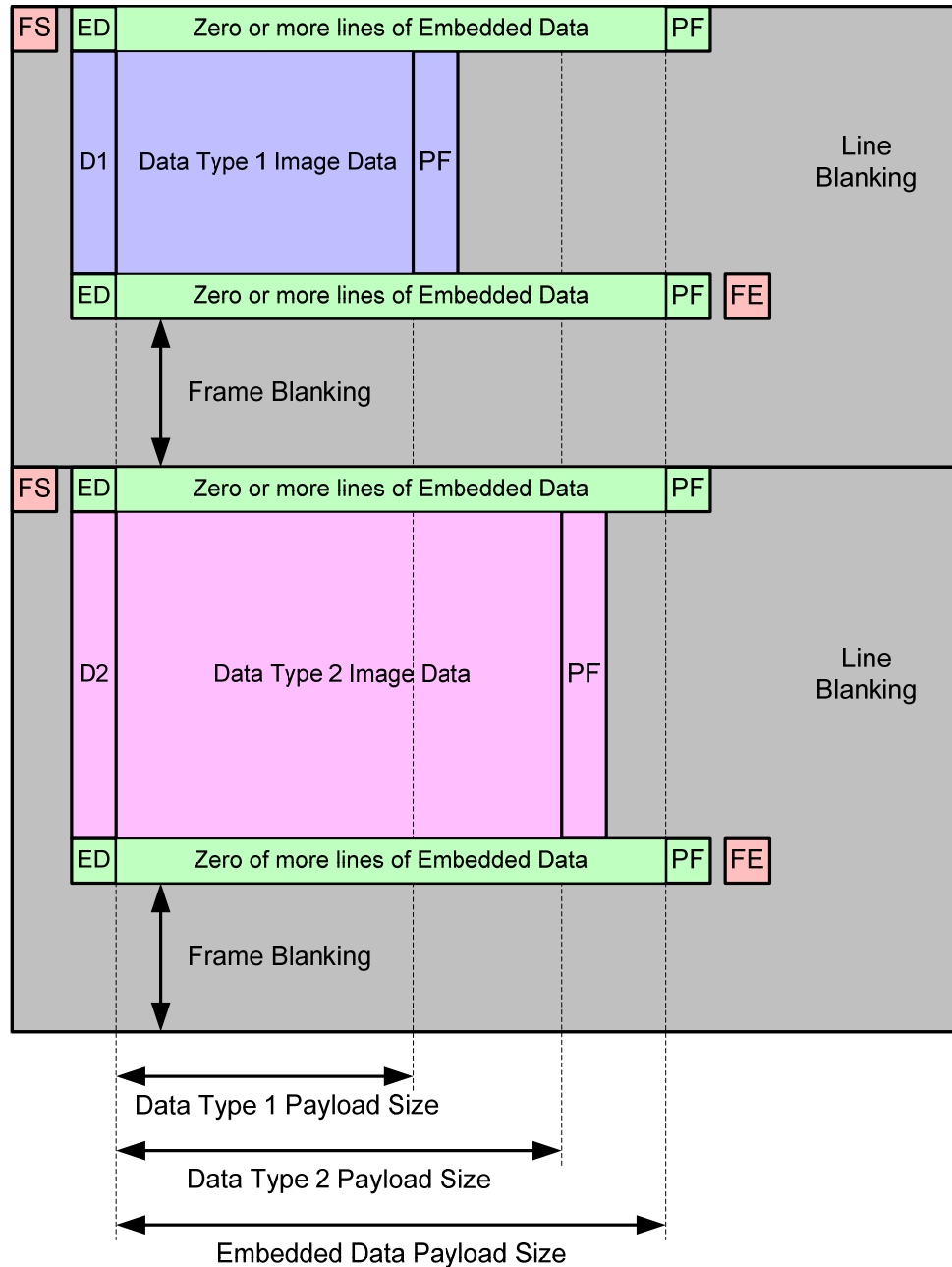
PF – Packet Footer

ED – Packet Header containing Embedded Data type code

D1 – Packet Header containing Data Type 1 Image Data Code

D2 – Packet Header containing Data Type 2 Image Data Code

**Figure 54 Packet Level Interleaved Data Transmission**

**KEY:**

LPS – Low Power State

FS – Frame Start

FE – Frame End

PF – Packet Footer

ED – Packet Header containing Embedded Data type code

D1 – Packet Header containing Data Type 1 Image Data Code

D2 – Packet Header containing Data Type 2 Image Data Code

**Figure 55 Frame Level Interleaved Data Transmission****9.13.2 Virtual Channel Identifier Interleaving**

The Virtual Channel Identifier allows different data types within a single data stream to be logically separated from each other. Figure 56 illustrates data interleaving using the Virtual Channel Identifier.

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

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

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

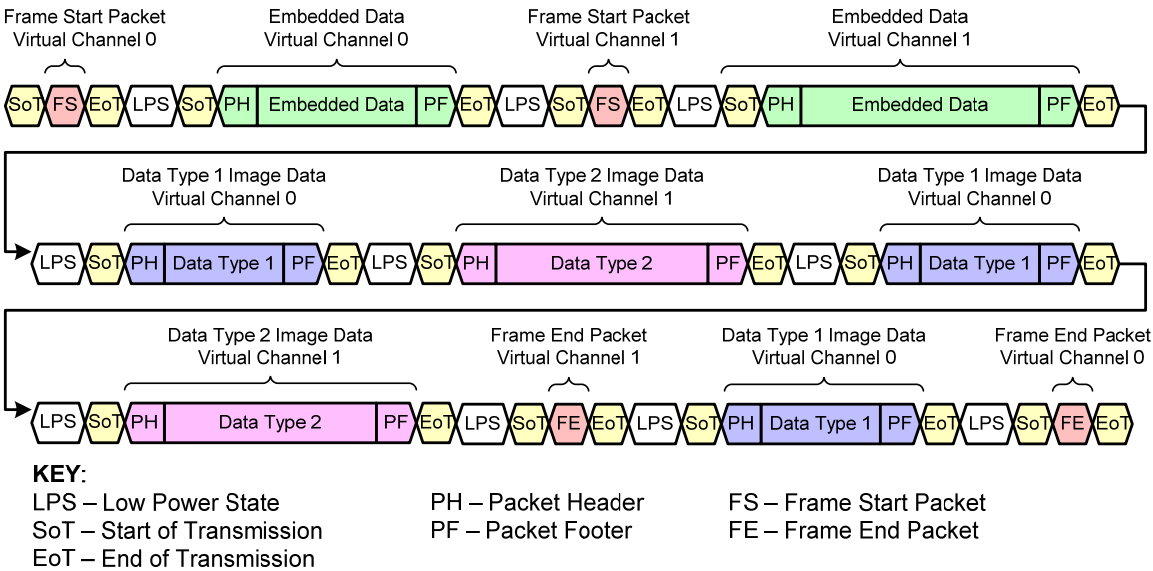


Figure 56 Interleaved Data Transmission using Virtual Channels

## 1195   **10   Color Spaces**

1196   The color space definitions in this section are simply references to other standards. The references are  
1197   included only for informative purposes and not for compliance. The color space used is not limited to the  
1198   references given.

### 1199   **10.1   RGB Color Space Definition**

1200   In this specification, the abbreviation RGB means the nonlinear sR'G'B' color space in 8-bit representation  
1201   based on the definition of sRGB in IEC 61966.

1202   The 8-bit representation results as RGB888. The conversion to the more commonly used RGB565 format is  
1203   achieved by scaling the 8-bit values to five bits (blue and red) and six bits (green). The scaling can be done  
1204   either by simply dropping the LSBs or rounding.

### 1205   **10.2   YUV Color Space Definition**

1206   In this specification, the abbreviation YUV refers to the 8-bit gamma corrected Y'CBCR color space  
1207   defined in ITU-R BT601.4.  
1208

## 11 Data Formats

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.

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.

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

**Table 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
Generic 8-bit Long Packet Data Types	P	
User Defined Byte-based Data (Note 1)	P	

Notes:

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



1222 For clarity the Start of Transmission and End of Transmission sequences in the figures in this section have  
1223 been omitted.

## 1224 **11.1 Generic 8-bit Long Packet Data Types**

1225 Table 9 defines the generic 8-bit Long packet data types.

1226 **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

### 1227 **11.1.1 Null and Blanking Data**

1228 For both the null and blanking data types the receiver must ignore the content of the packet payload data.

1229 A blanking packet differs from a null packet in terms of its significance within a video data stream. A null  
1230 packet has no meaning whereas the blanking packet may be used, for example, as the blanking lines  
1231 between frames in an ITU-R BT.656 style video stream.

### 1232 **11.1.2 Embedded Information**

1233 It is possible to embed extra lines containing additional information to the beginning and to the end of each  
1234 picture frame as presented in the Figure 57. If embedded information exists, then the lines containing the  
1235 embedded data must use the embedded data code in the data identifier.

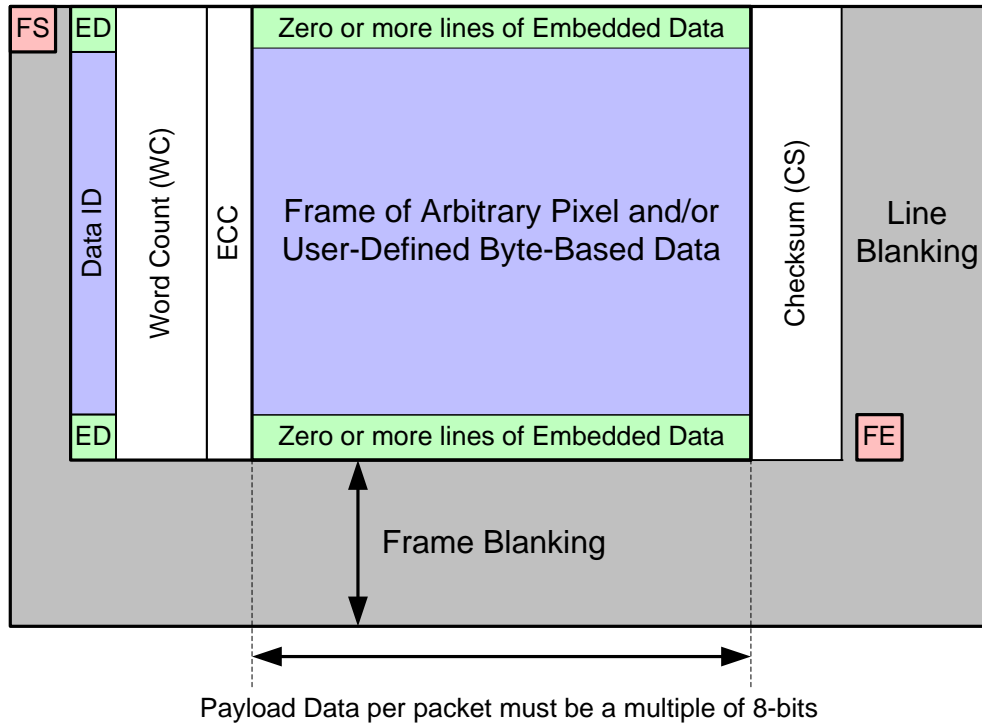
1236 There may be zero or more lines of embedded data at the start of the frame. These lines are termed the  
1237 frame header.

1238 There may be zero or more line of embedded data at the end of the frame. These lines are termed the frame  
1239 footer.

## 1240 **11.2 YUV Image Data**

1241 Table 10 defines the data type codes for YUV data formats described in this section. The number of lines  
1242 transmitted for the YUV420 data type shall be even.

1243 YUV420 data formats are divided into legacy and non-legacy data formats. The legacy YUV420 data  
1244 format is for compatibility with existing systems. The non-legacy YUV420 data formats enable lower cost  
1245 implementations.

**KEY:**

LPS – Low Power State

ECC – Error Correction Code

FS – Frame Start

LS – Line Start

DI – Data Identifier

CS – Checksum

FE – Frame End

LE – Line End

WC – Word Count

ED – Embedded Data

**Figure 57 Frame Structure with Embedded Data at the Beginning and End of the Frame****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

**11.2.1 Legacy YUV420 8-bit**

Legacy YUV420 8-bit data transmission is performed by transmitting UYY... / VYY... sequences in odd / even lines. U component is transferred in odd lines (1,3,5...) and V component is transferred in even lines (2,4,6...). This sequence is illustrated in Figure 58.

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

1255

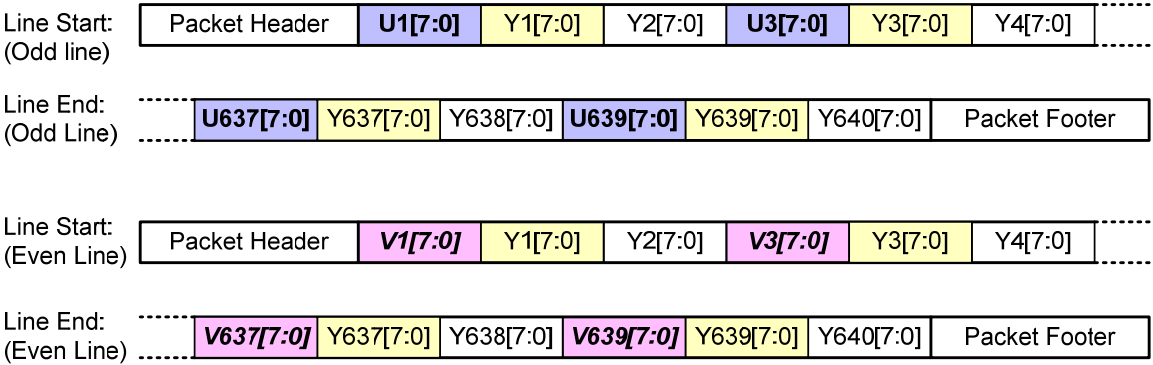
**Table 11 Legacy YUV420 8-bit Packet Data Size Constraints**

Pixels	Bytes	Bits
2	3	24

1256

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in Figure 59.

1257

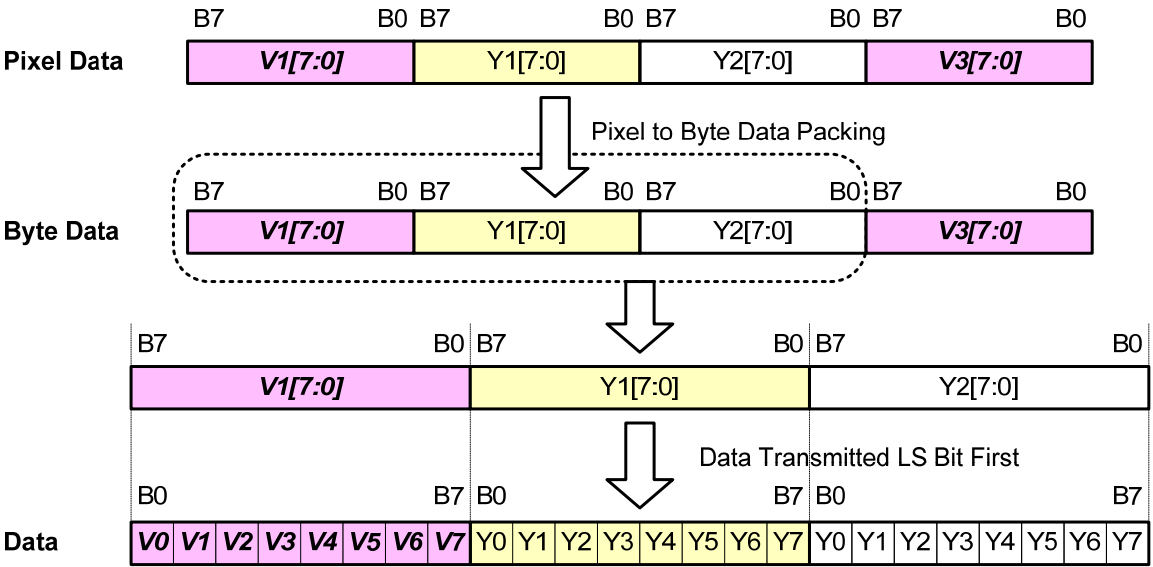


1258

1259

1260

**Figure 58 Legacy YUV420 8-bit Transmission**



1261

1262

1263

**Figure 59 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration**

1264

There is one spatial sampling option

1265

- H.261, H.263 and MPEG1 Spatial Sampling (Figure 60).

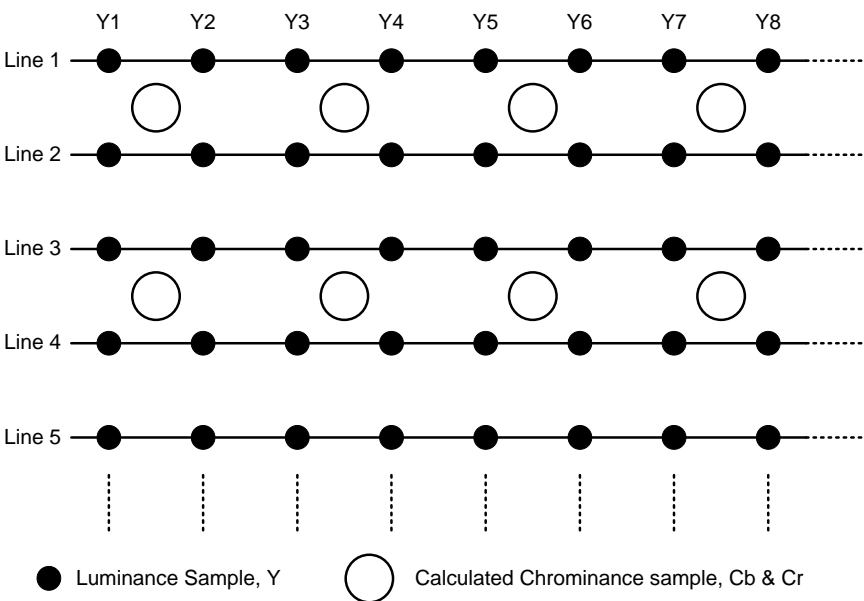


Figure 60 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1

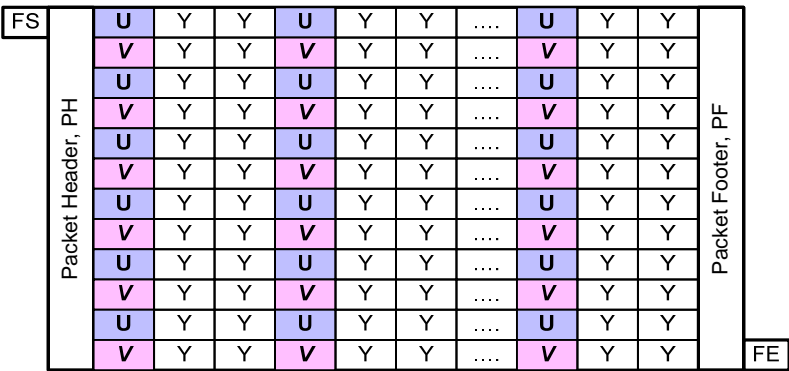


Figure 61 Legacy YUV420 8-bit Frame Format

11.2.2 YUV420 8-bit

YUV420 8-bit data transmission is performed by transmitting YYYY... / UYVYUYVY... sequences in odd / even lines. Only the luminance component (Y) is transferred for odd lines (1, 3, 5...) and both luminance (Y) and chrominance (U and V) components are transferred for even lines (2, 4, 6...). The format for the even lines (UYVY) is identical to the YUV422 8-bit data format. The data transmission sequence is illustrated in Figure 62.

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

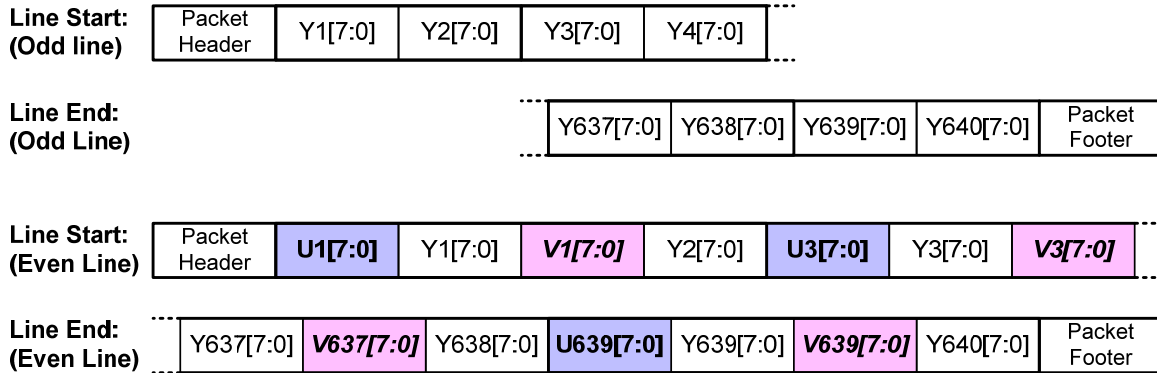
Table 12 specifies the packet size constraints for YUV420 8-bit packets. Each packet must be a multiple of the values in the table.

1280

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

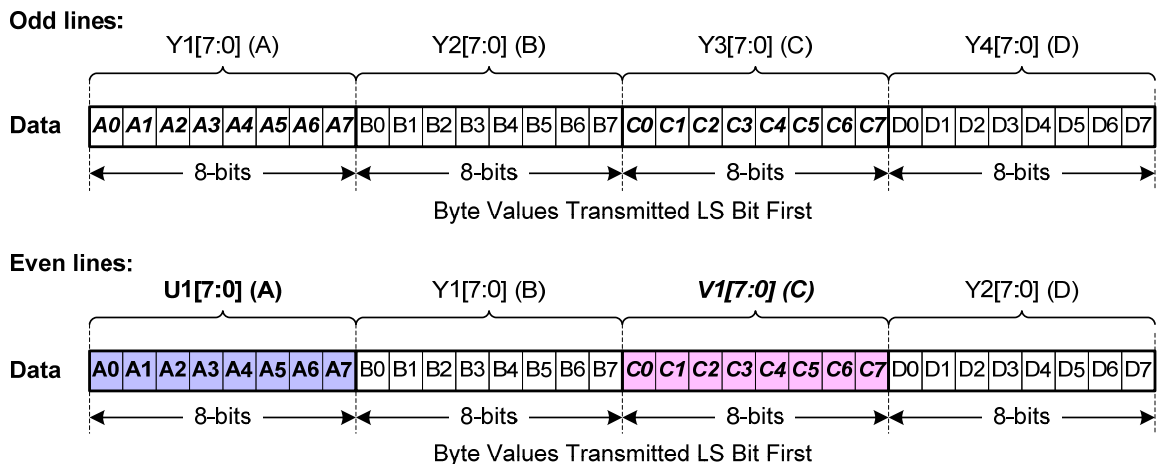
Odd Lines (1, 3, 5...) Luminance Only, Y			Even Lines (2, 4, 6...) Luminance and Chrominance, UYVY		
Pixels	Bytes	Bits	Pixels	Bytes	Bits
2	2	16	2	4	32

1281 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated  
 1282 in Figure 63.



1283  
1284

1285

**Figure 62 YUV420 8-bit Data Transmission Sequence**

1286  
1287

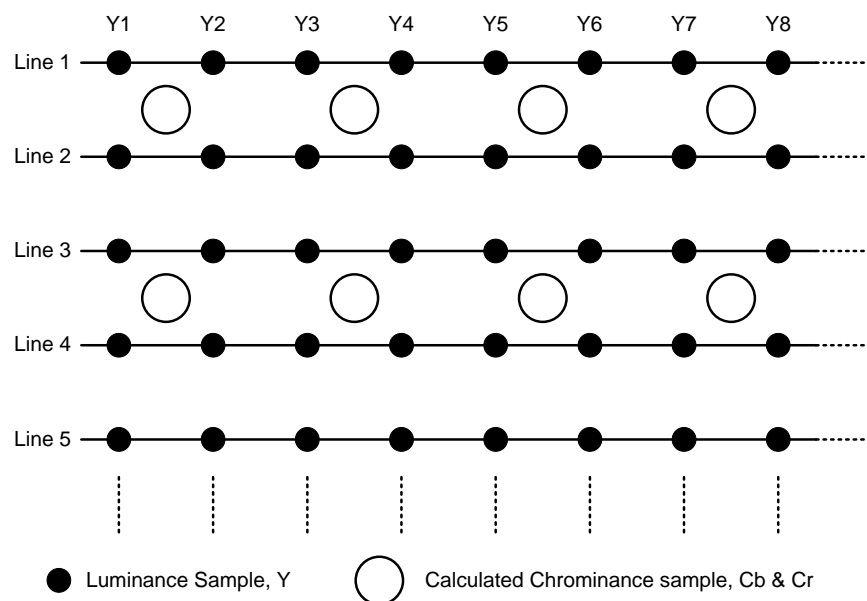
1288

**Figure 63 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration**

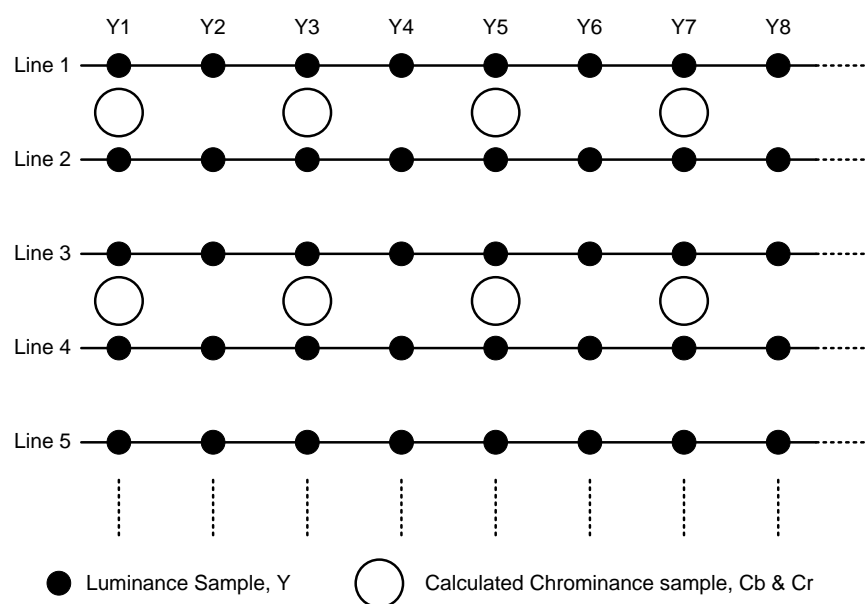
1289 There are two spatial sampling options

- 1290      • H.261, H.263 and MPEG1 Spatial Sampling (Figure 64).  
 1291      • Chroma Shifted Pixel Sampling (CSPS) for MPEG2, MPEG4 (Figure 65).

1292 Figure 66 shows the YUV420 frame format.



**Figure 64 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1**



**Figure 65 YUV420 Spatial Sampling for MPEG 2 and MPEG 4**

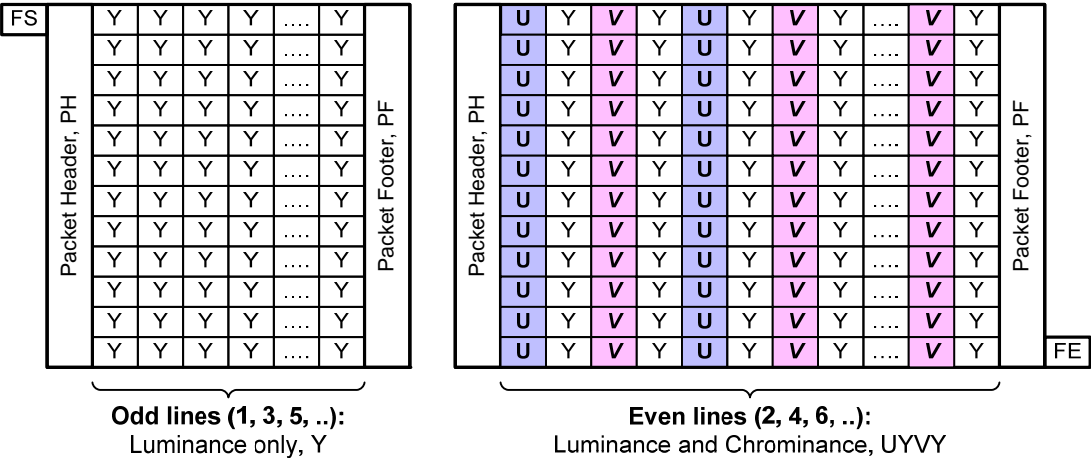


Figure 66 YUV420 8-bit Frame Format

11.2.3 YUV420 10-bit

YUV420 10-bit data transmission is performed by transmitting YYYYY... / 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 67.

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

Table 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.

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

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in Figure 68.

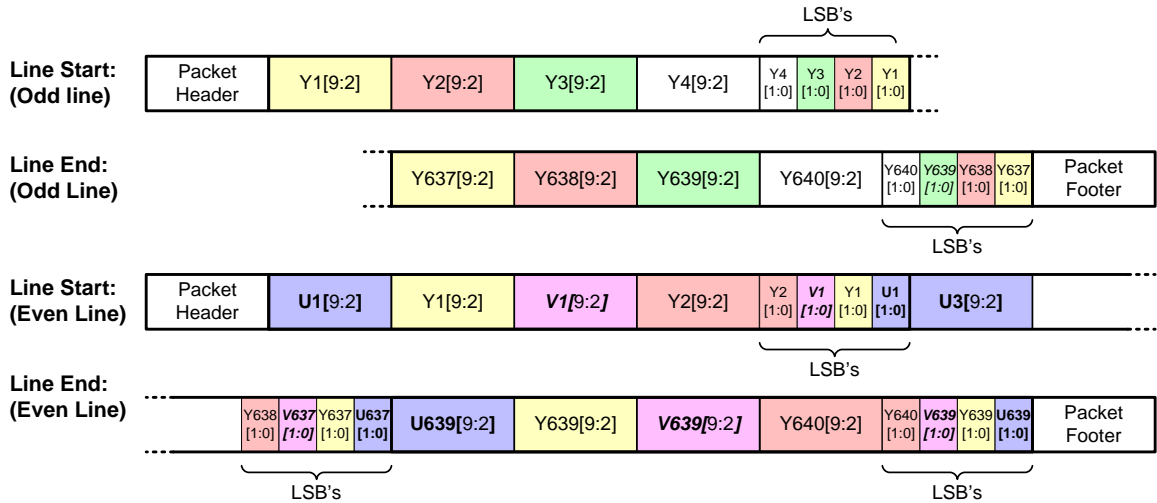


Figure 67 YUV420 10-bit Transmission

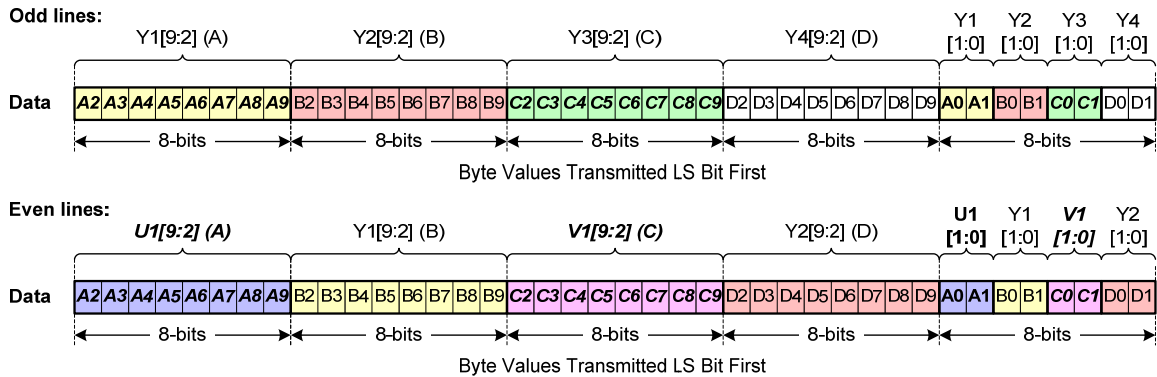


Figure 68 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration

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

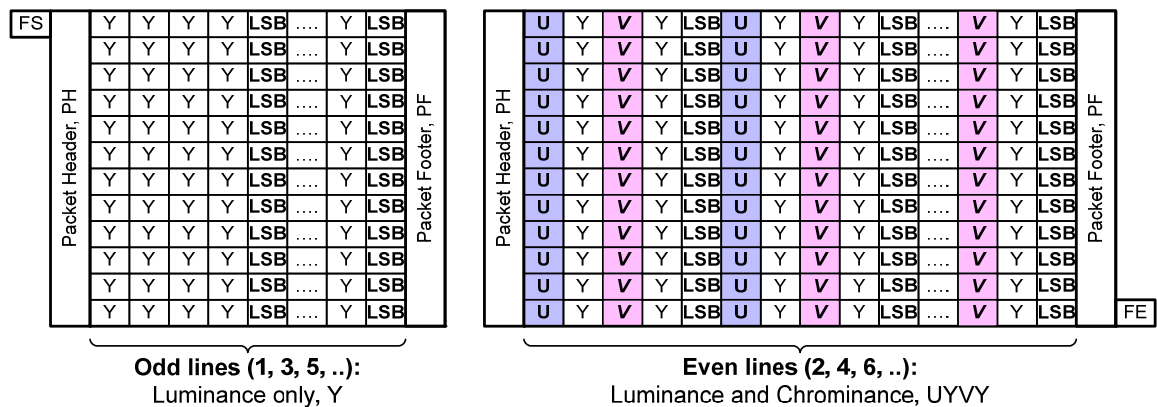


Figure 69 YUV420 10-bit Frame Format



#### 11.2.4 YUV422 8-bit

YUV422 8-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in Figure 70.

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

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

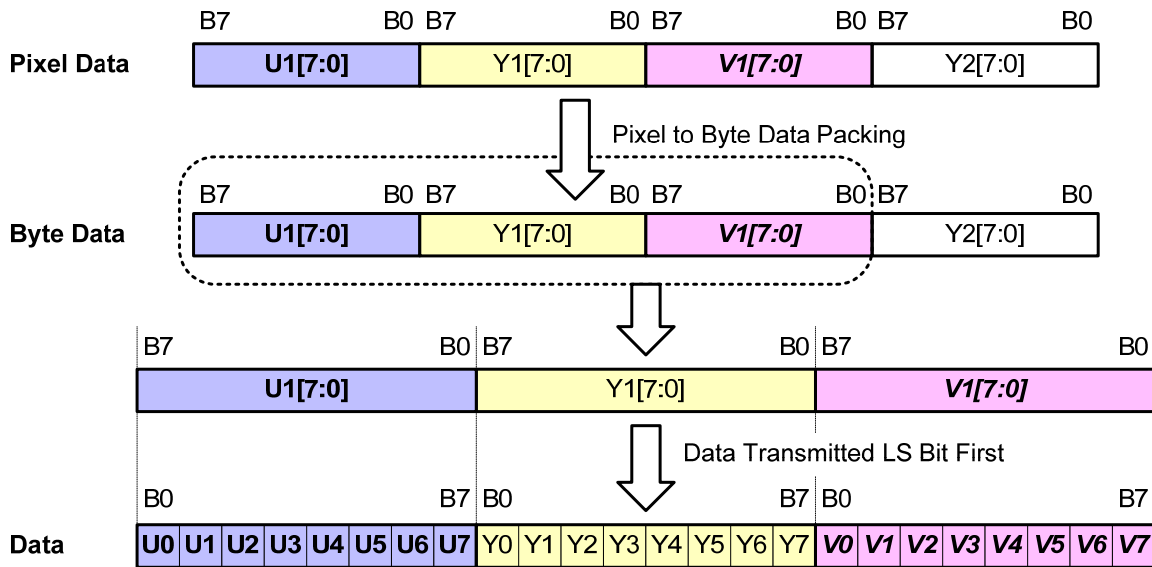
Pixels	Bytes	Bits
2	4	32

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in Figure 71.

Line Start: Packet Header U1[7:0] Y1[7:0] V1[7:0] Y2[7:0] U3[7:0] .....

Line End: ..... Y638[7:0] U639[7:0] Y639[7:0] V639[7:0] Y640[7:0] Packet Footer

**Figure 70 YUV422 8-bit Transmission**



**Figure 71 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration**

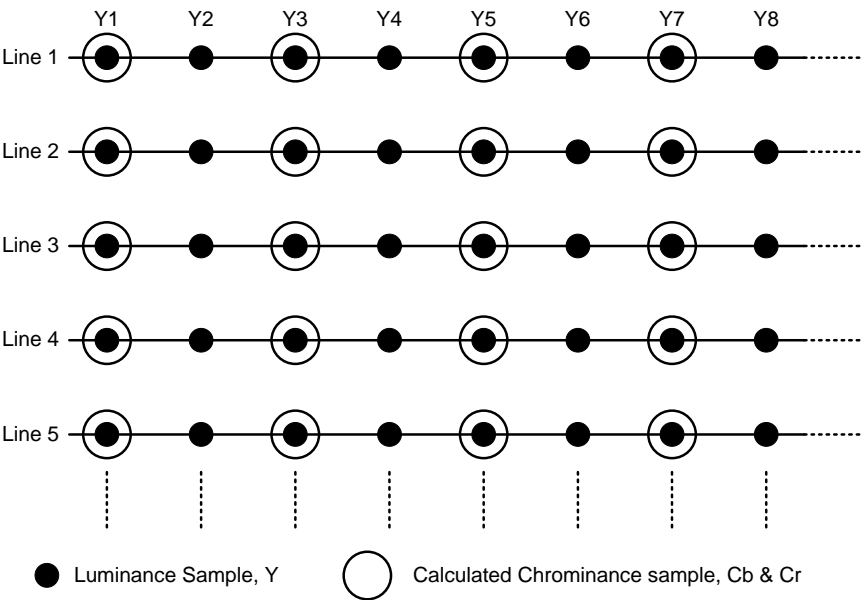


Figure 72 YUV422 Co-sited Spatial Sampling

The pixel spatial alignment is the same as in CCIR-656 standard. The frame format for YUV422 is presented in Figure 73.

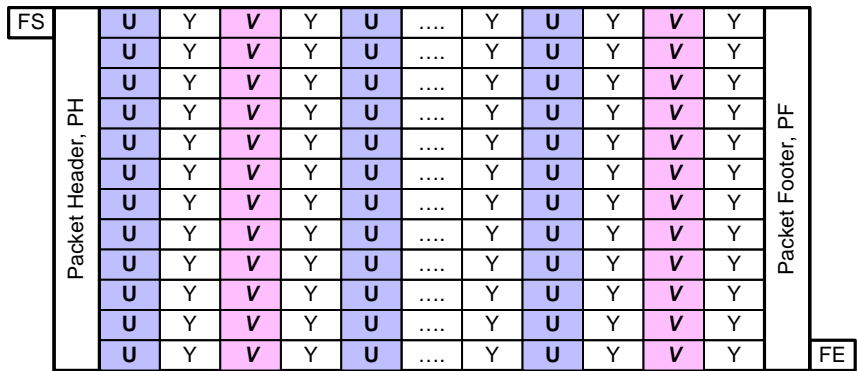


Figure 73 YUV422 8-bit Frame Format

11.2.5 YUV422 10-bit

YUV422 10-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in Figure 74.

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

Table 15 YUV422 10-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	5	40

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in Figure 75.

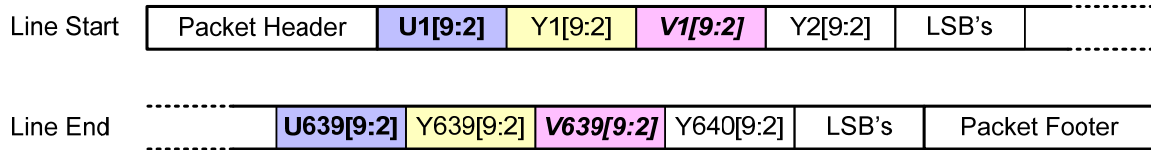
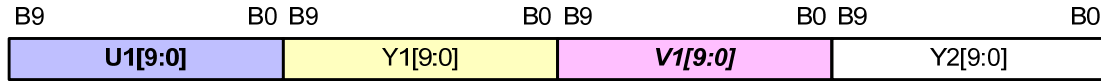
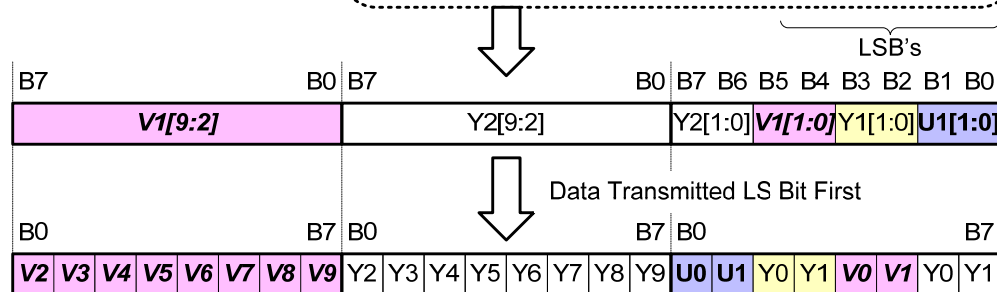
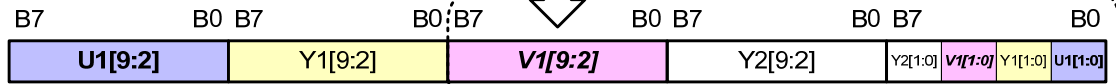


Figure 74 YUV422 10-bit Transmitted Bytes

Pixel Data:



Byte Data:



Data

Figure 75 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 76.

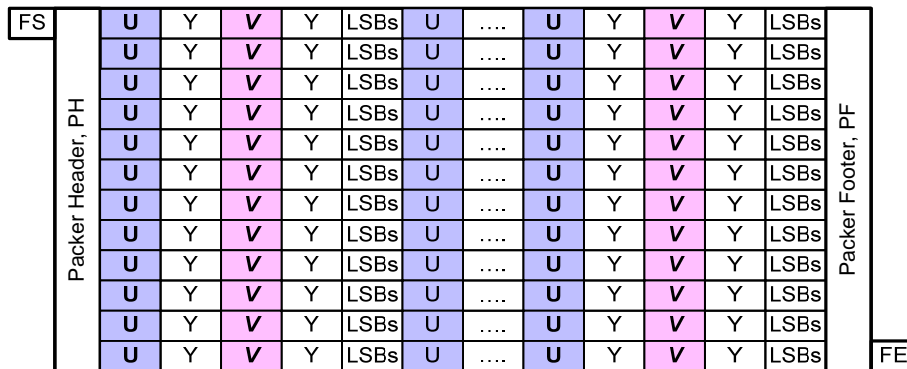


Figure 76 YUV422 10-bit Frame Format

### 11.3 RGB Image Data

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

**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

**11.3.1 RGB888**

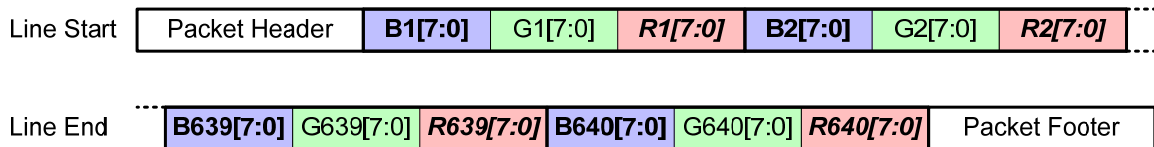
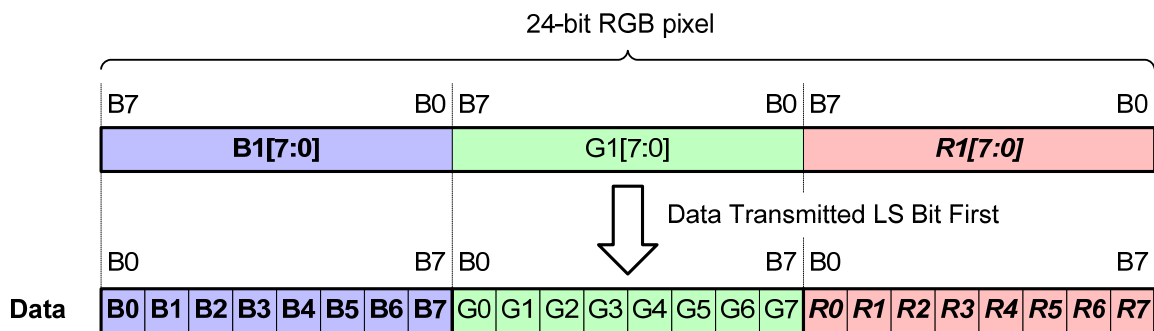
RGB888 data transmission is performed by transmitting a BGR byte sequence. This sequence is illustrated in Figure 77. The RGB888 frame format is illustrated in Figure 79.

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.

**Table 17 RGB888 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	3	24

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in Figure 78.

**Figure 77 RGB888 Transmission****Figure 78 RGB888 Transmission in CSI-2 Bus Bitwise Illustration**

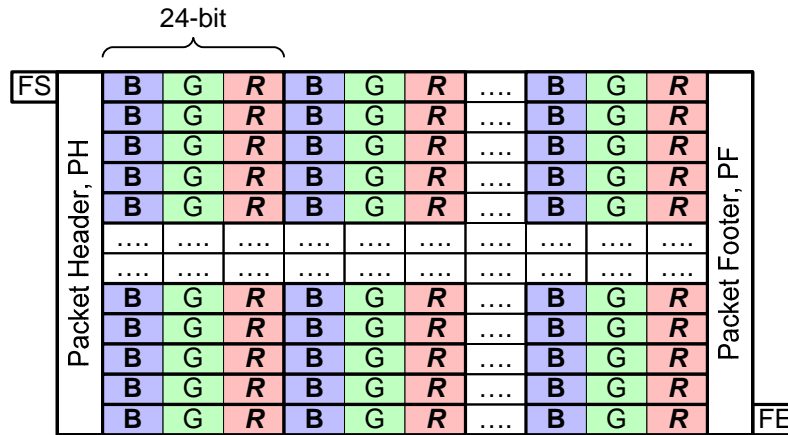


Figure 79 RGB888 Frame Format

### 11.3.2 RGB666

RGB666 data transmission is performed by transmitting B0..5 G0..5 R0..5 (18-bit) sequence. This sequence is illustrated in Figure 80. The frame format for RGB666 is presented in the Figure 82.

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

Table 18 RGB666 Packet Data Size Constraints

Pixels	Bytes	Bits
4	9	72

Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB666 case the length of one data word is 18-bits, not eight bits. The word wise flip is done for 18-bit BGR words i.e. instead of flipping each byte (8-bits), each 18-bits pixel value is flipped. This is illustrated in Figure 81.

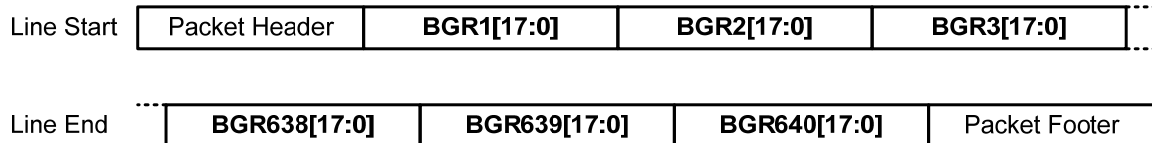


Figure 80 RGB666 Transmission with 18-bit BGR Words

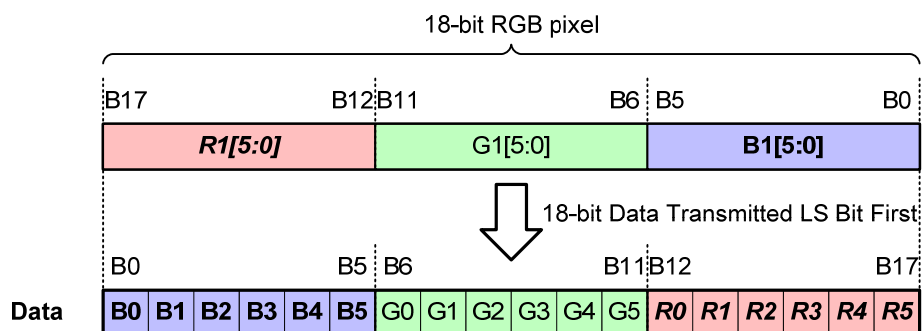
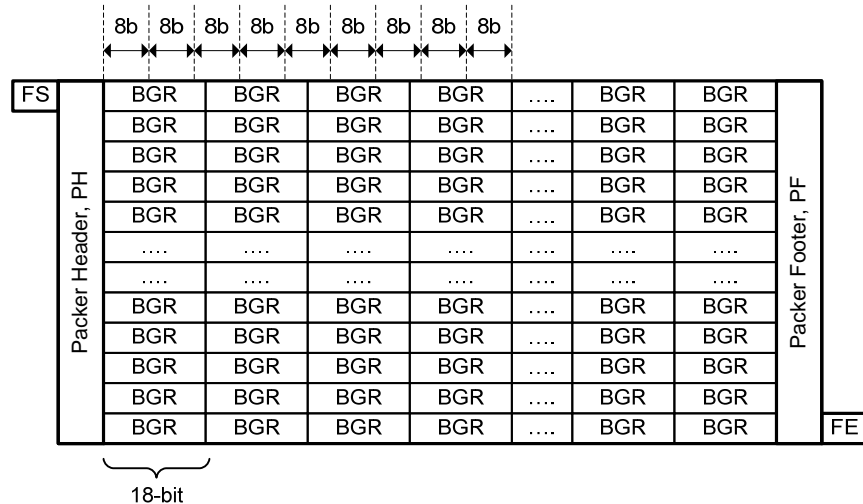


Figure 81 RGB666 Transmission on CSI-2 Bus Bitwise Illustration



**Figure 82 RGB666 Frame Format**

### 11.3.3 RGB565

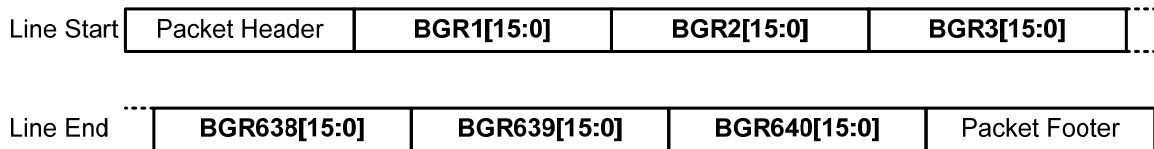
RGB565 data transmission is performed by transmitting B0...B4, G0...G5, R0...R4 in a 16-bit sequence. This sequence is illustrated in Figure 83. The frame format for RGB565 is presented in the Figure 85.

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

**Table 19 RGB565 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	2	16

Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB565 case the length of one data word is 16-bits, not eight bits. The word wise flip is done for 16-bit BGR words i.e. instead of flipping each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 84.



**Figure 83 RGB565 Transmission with 16-bit BGR Words**

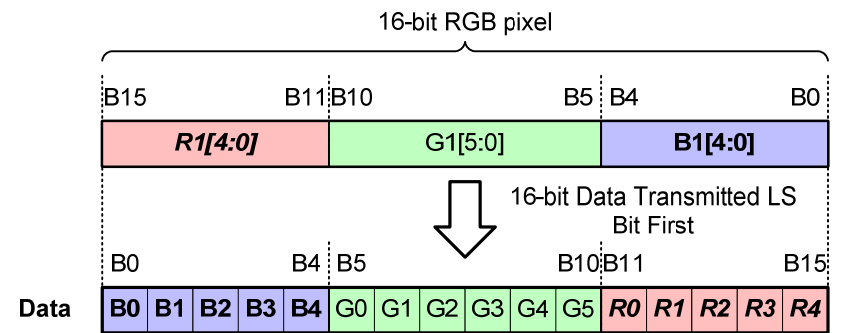


Figure 84 RGB565 Transmission on CSI-2 Bus Bitwise Illustration

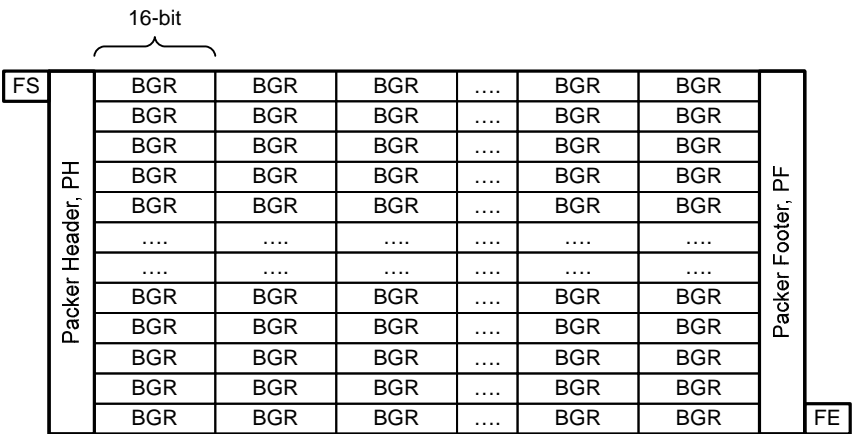


Figure 85 RGB565 Frame Format

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 86.

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 86.

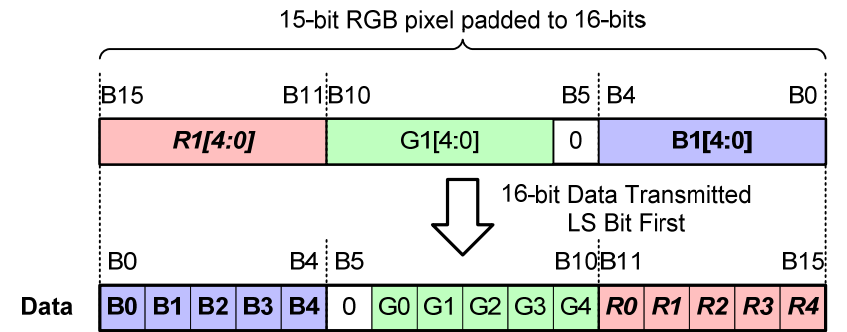


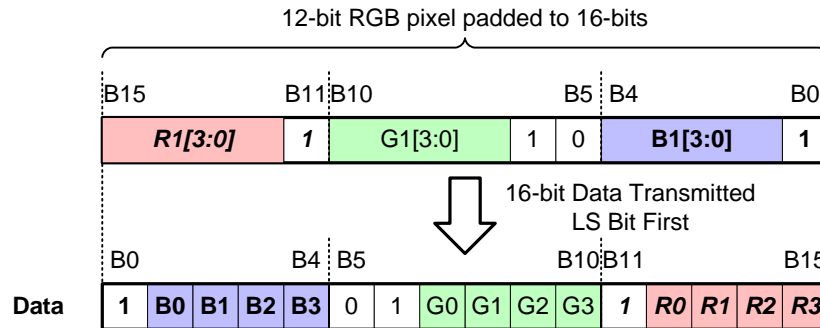
Figure 86 RGB555 Transmission on CSI-2 Bus Bitwise Illustration

### 11.3.5 RGB444

RGB444 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB444 data should be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs of each color component as illustrated in Figure 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 RGB444 case the length of one data word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words i.e. instead of flipping each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 87.



**Figure 87 RGB444 Transmission on CSI-2 Bus Bitwise Illustration**

### 11.4 RAW Image Data

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

The intent is that Raw image data is unprocessed image data for example Raw Bayer data or complementary color data, but RAW image data is not limited to these data types.

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

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

**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



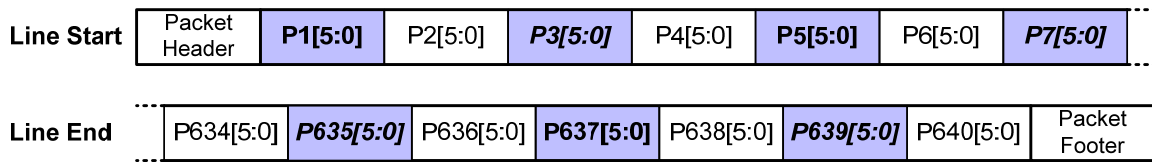
### 11.4.1 RAW6

The 6-bit Raw data transmission is performed by transmitting the pixel data over CSI-2 bus. Each line is separated by line start / end synchronization codes. This sequence is illustrated in Figure 88 (VGA case). Table 21 specifies the packet size constraints for RAW6 packets. The length of each packet must be a multiple of the values in the table.

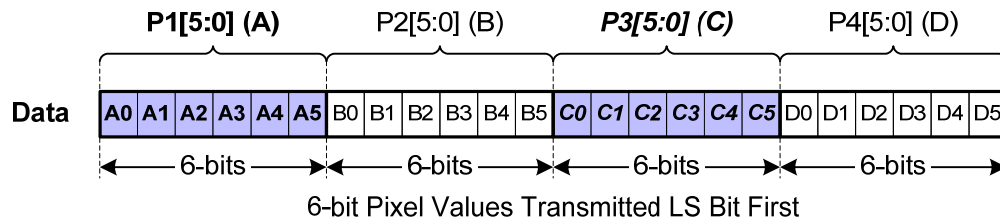
**Table 21 RAW6 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	3	24

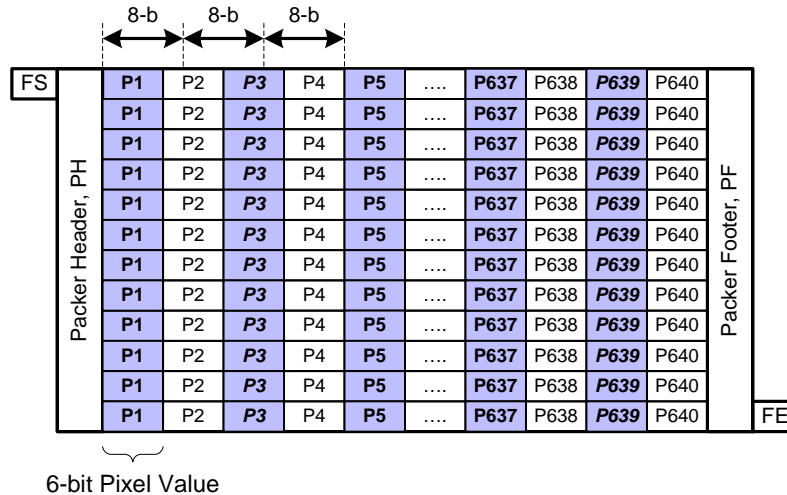
Each 6-bit pixel is sent LSB first. This is an exception to general CSI-2 rule byte wise LSB first.



**Figure 88 RAW6 Transmission**



**Figure 89 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration**



**Figure 90 RAW6 Frame Format**

### 11.4.2 RAW7

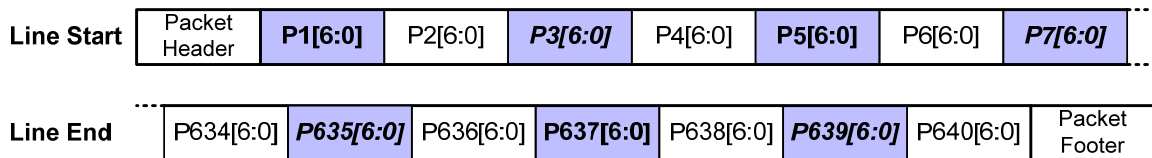
The 7-bit Raw data transmission is performed by transmitting the pixel data over CSI-2 bus. Each line is separated by line start / end synchronization codes. This sequence is illustrated in Figure 91 (VGA case).

1453 Table 22 specifies the packet size constraints for RAW7 packets. The length of each packet must be a  
 1454 multiple of the values in the table.

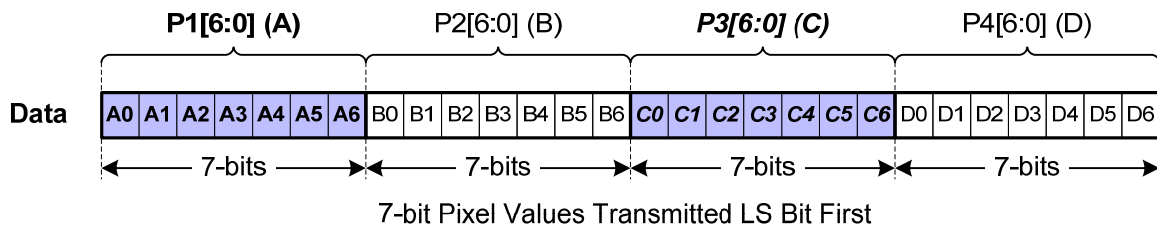
1455 **Table 22 RAW7 Packet Data Size Constraints**

Pixels	Bytes	Bits
8	7	56

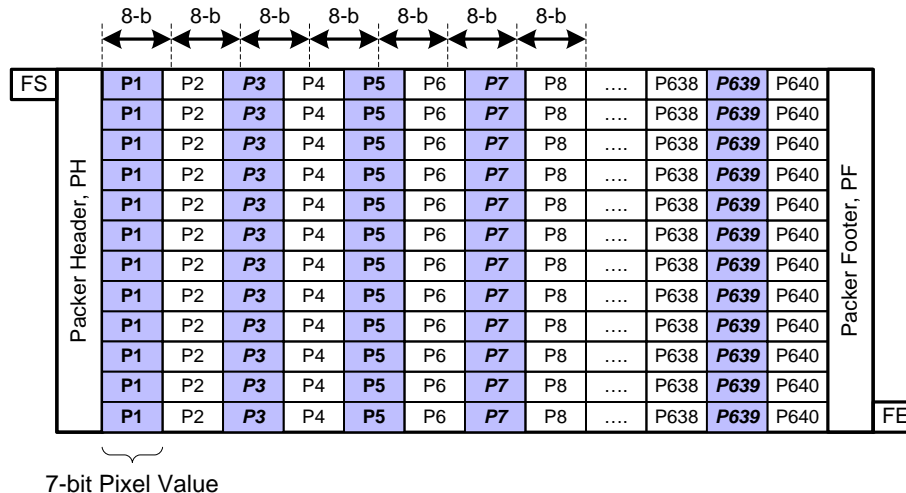
1456 Each 7-bit pixel is sent LSB first. This is an exception to general CSI-2 rule byte-wise LSB first.



**Figure 91 RAW7 Transmission**



**Figure 92 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration**



**Figure 93 RAW7 Frame Format**

### 11.4.3 RAW8

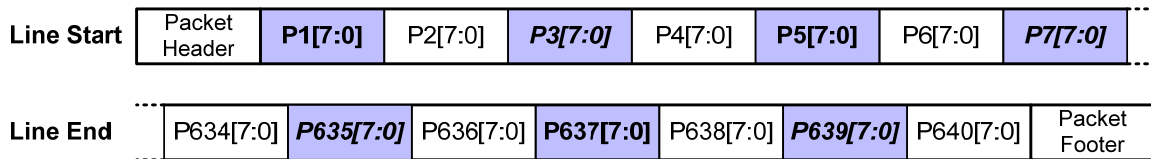
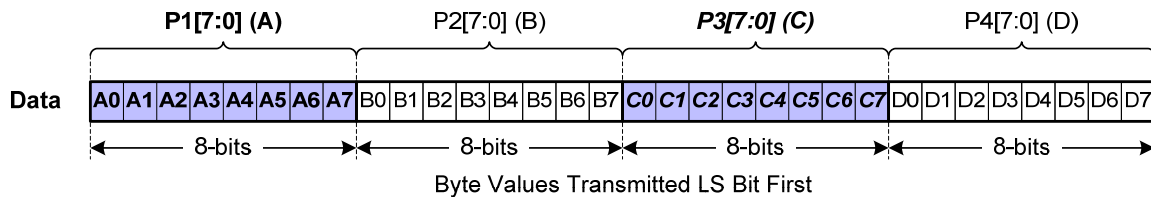
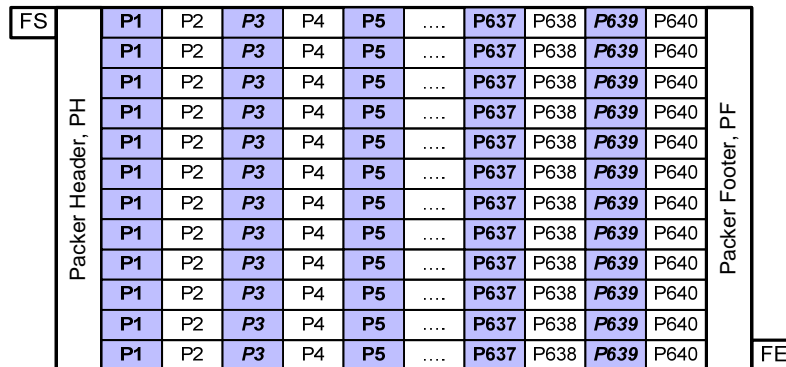
1465 The 8-bit Raw data transmission is performed by transmitting the pixel data over a CSI-2 bus. Table 23  
 1466 specifies the packet size constraints for RAW8 packets. The length of each packet must be a multiple of the  
 1467 values in the table.

**Table 23 RAW8 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	1	8

This sequence is illustrated in Figure 94 (VGA case).

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

**Figure 94 RAW8 Transmission****Figure 95 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration****Figure 96 RAW8 Frame Format**

#### 11.4.4 RAW10

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

**Table 24 RAW10 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	5	40

This sequence is illustrated in Figure 97 (VGA case).

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

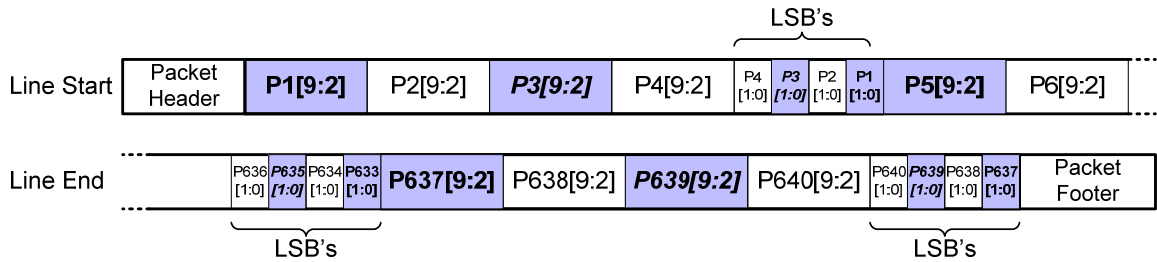


Figure 97 RAW10 Transmission

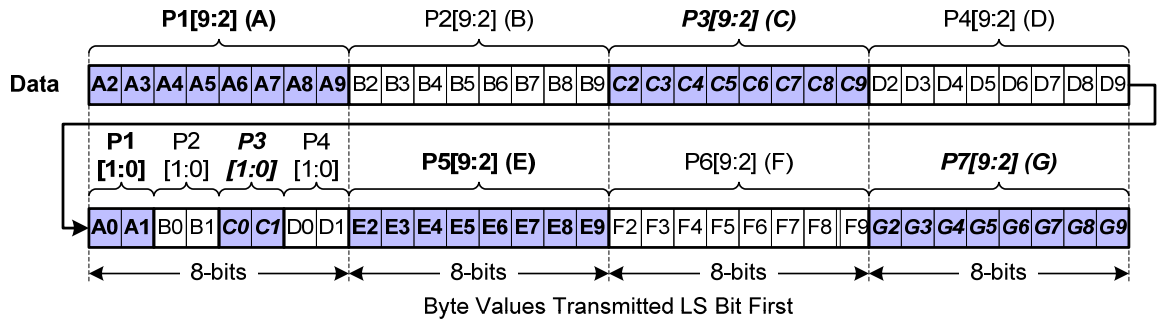


Figure 98 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration

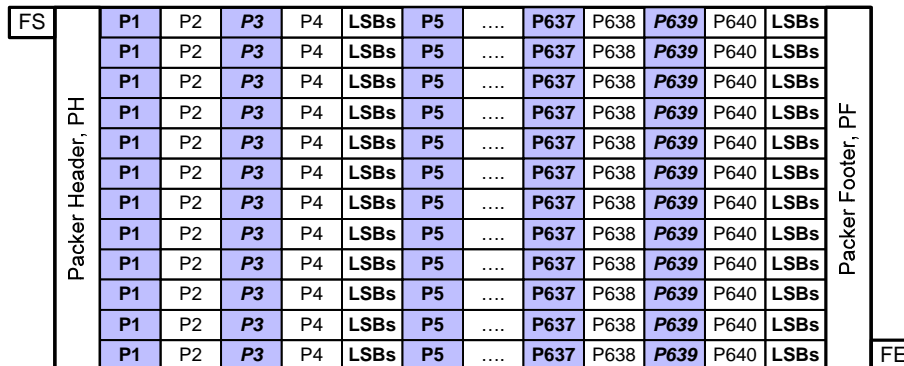


Figure 99 RAW10 Frame Format

## 11.4.5 RAW12

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

Table 25 RAW12 Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

This sequence is illustrated in Figure 100 (VGA case).

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

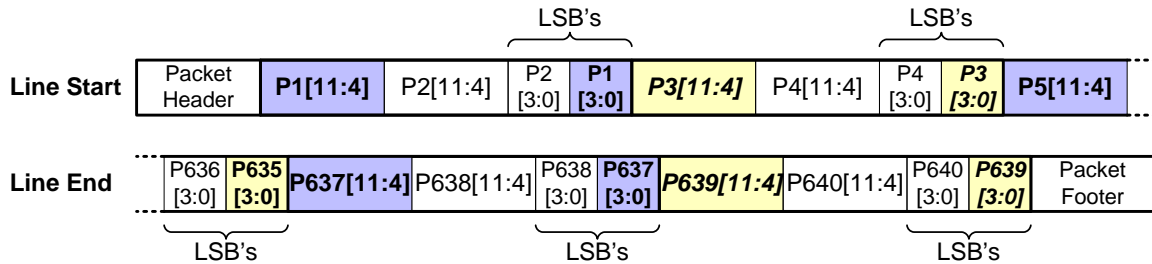


Figure 100 RAW12 Transmission

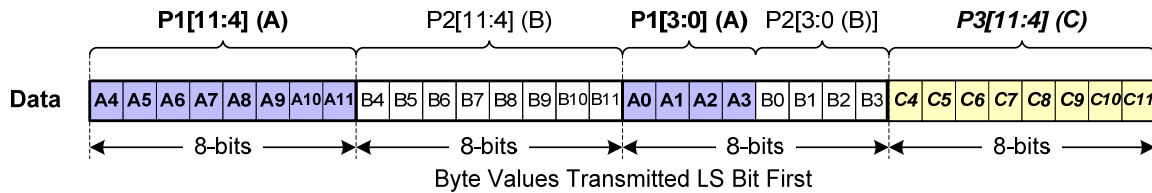


Figure 101 RAW12 Transmission on CSI-2 Bus Bitwise Illustration

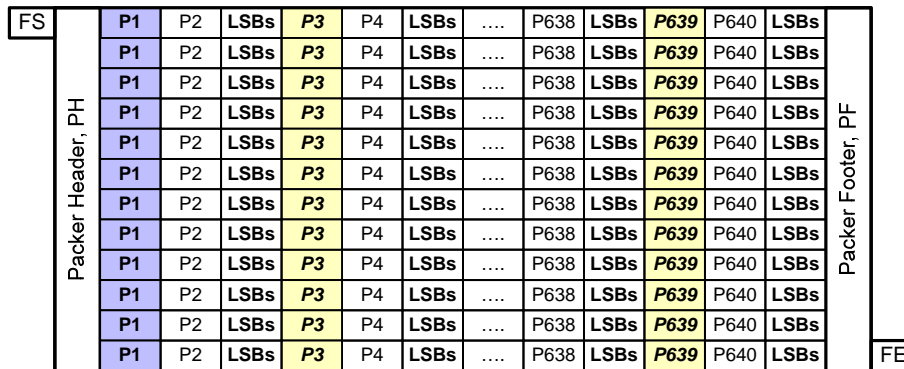


Figure 102 RAW12 Frame Format

#### 11.4.6 RAW14

The transmission of 14-bit Raw data is accomplished 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 103 (VGA case).

The LS bits for P1, P2, P3 and P4 are distributed in three bytes as shown in Figure 104. 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.

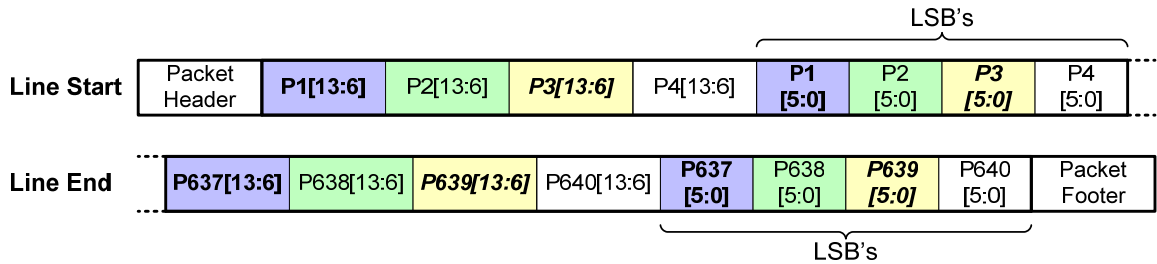


Figure 103 RAW14 Transmission

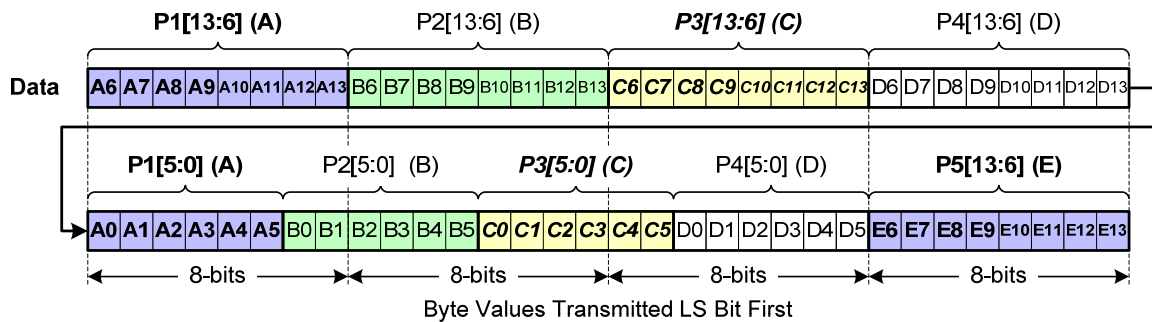


Figure 104 RAW14 Transmission on CSI-2 Bus Bitwise Illustration

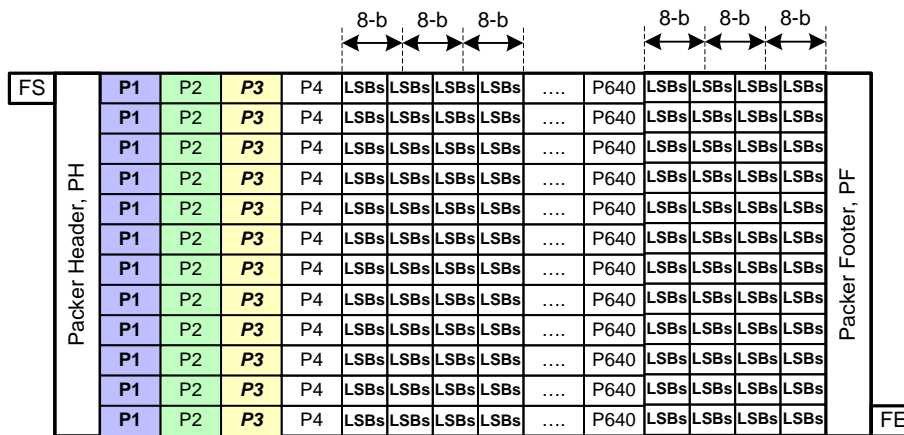


Figure 105 RAW14 Frame Format

## 11.5 User Defined Data Formats

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

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

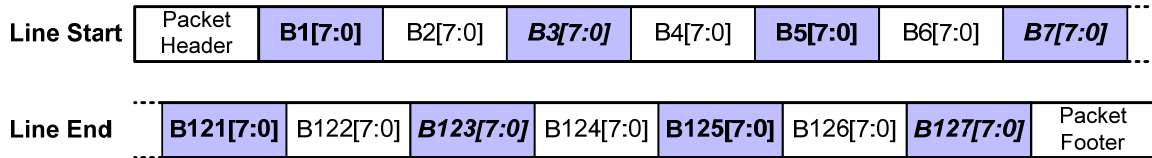


Figure 106 User Defined 8-bit Data (128 Byte Packet)

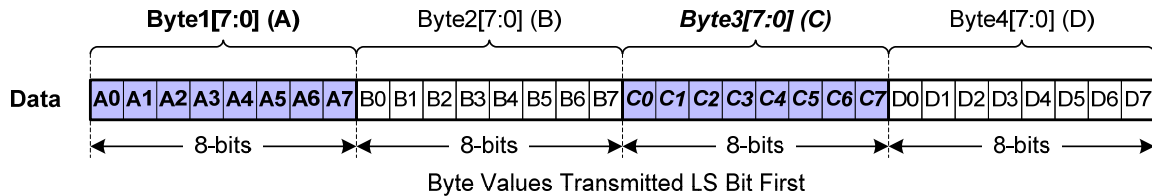


Figure 107 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration

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

For User Defined data:

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

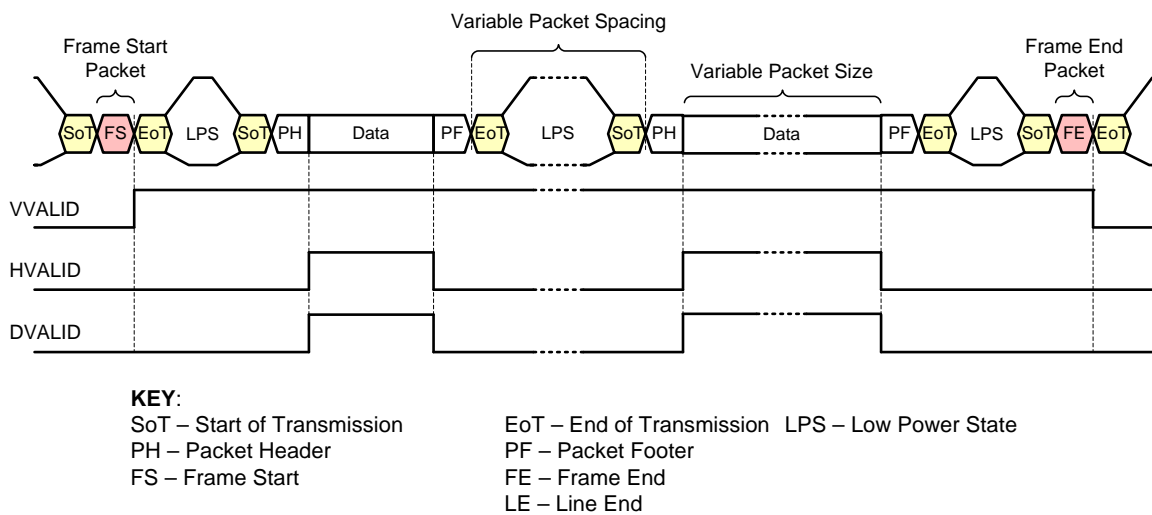


Figure 108 Transmission of User Defined 8-bit Data

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

<b>Data Type</b>	<b>Description</b>
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

1542

1543



## 12 Recommended Memory Storage

This section is informative.

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

### 12.1 General/Arbitrary Data Reception

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

The below is the generic CSI-2 byte to 32-bit memory word mapping rule.

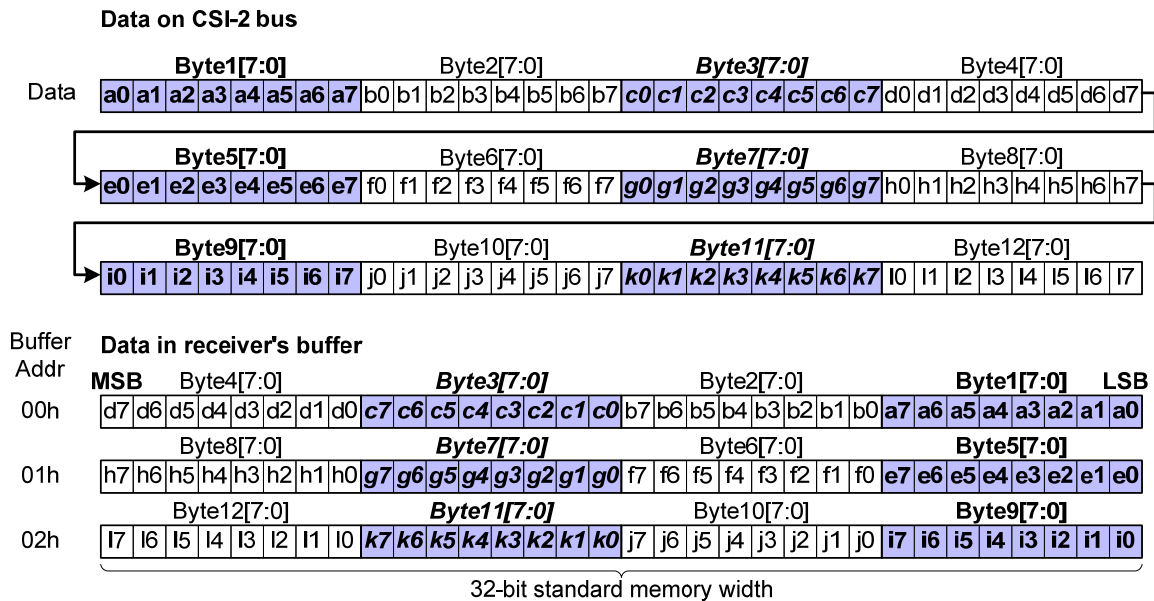


Figure 109 General/Arbitrary Data Reception

### 12.2 RGB888 Data Reception

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

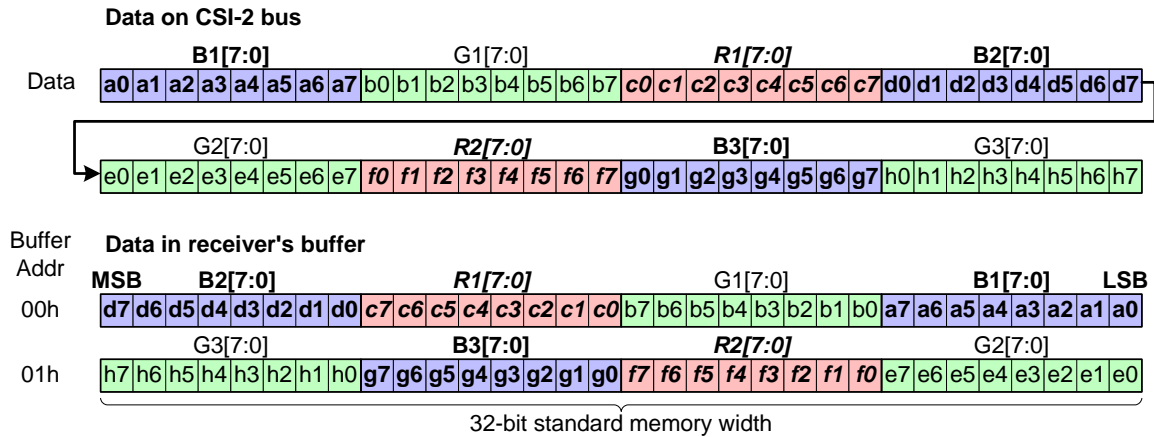


Figure 110 RGB888 Data Format Reception

### 12.3 RGB666 Data Reception

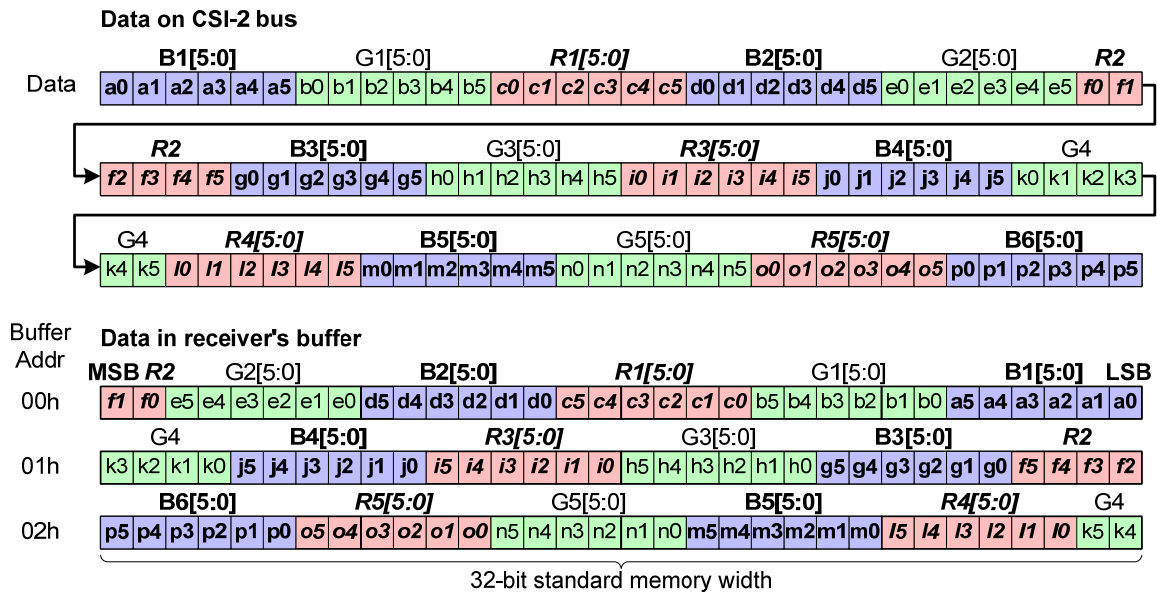


Figure 111 RGB666 Data Format Reception

## 12.4 RGB565 Data Reception

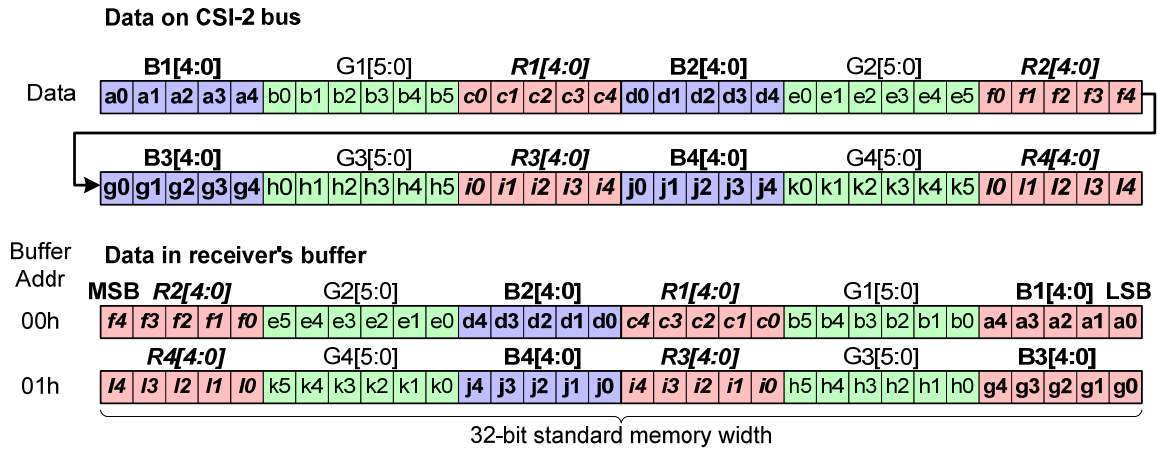


Figure 112 RGB565 Data Format Reception

## 12.5 RGB555 Data Reception

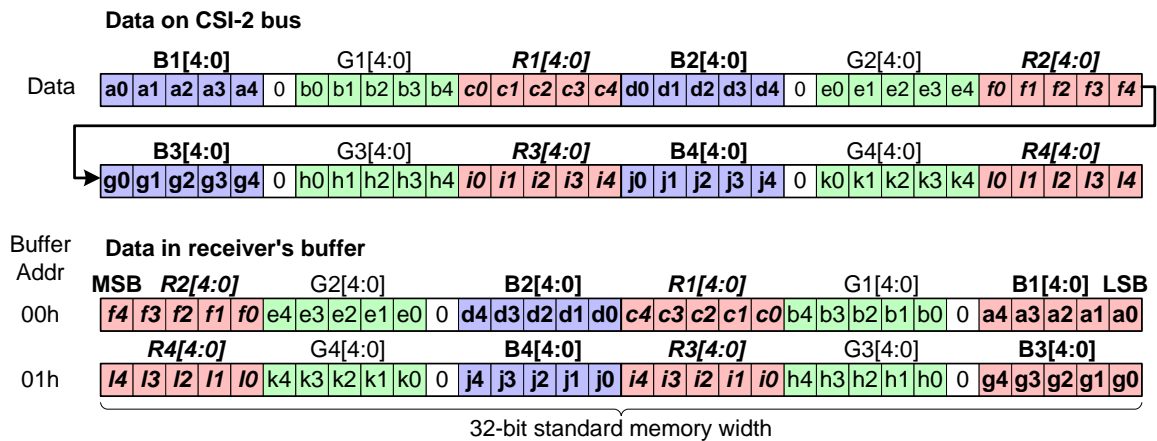


Figure 113 RGB555 Data Format Reception

## 12.6 RGB444 Data Reception

The RGB444 data format byte to 32-bit memory word mapping has a special transform as shown in Figure 114.

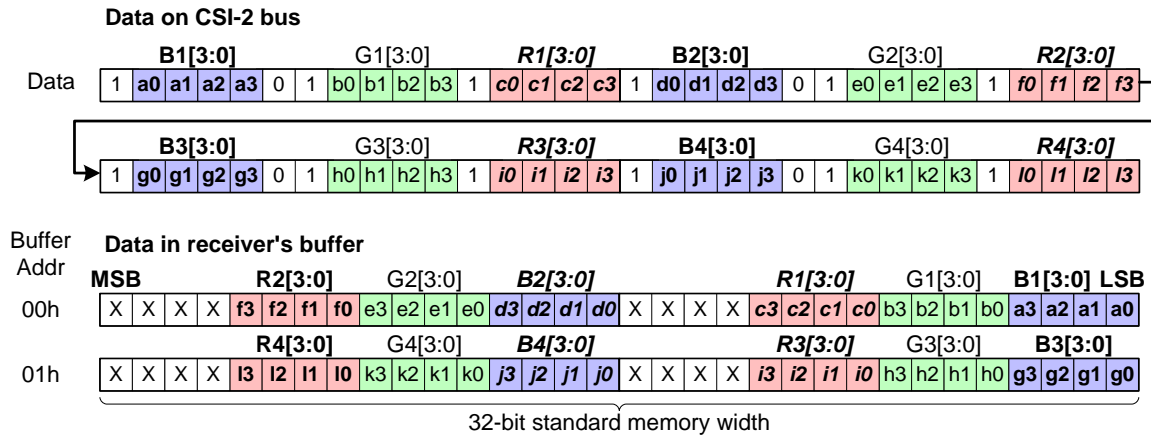


Figure 114 RGB444 Data Format Reception

## 12.7 YUV422 8-bit Data Reception

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

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

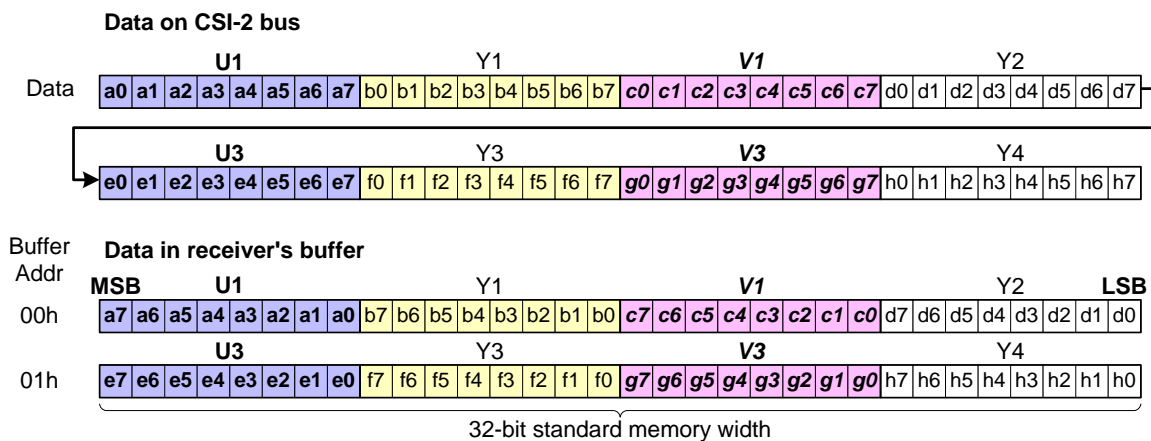
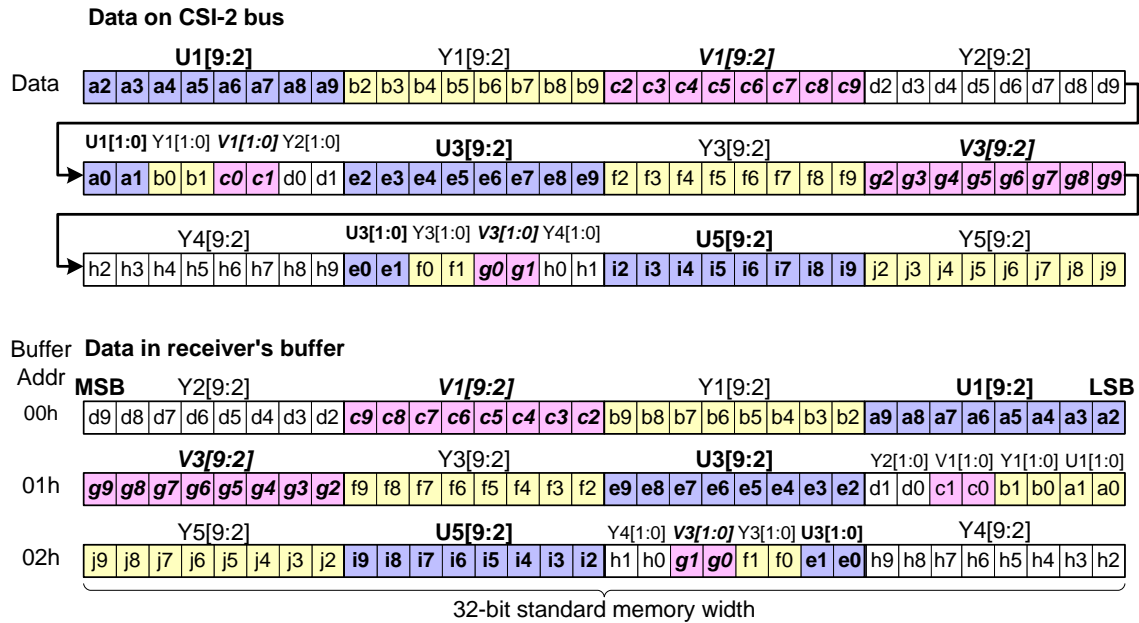


Figure 115 YUV422 8-bit Data Format Reception

## 12.8 YUV422 10-bit Data Reception

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



### Figure 116 YUV422 10-bit Data Format Reception

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

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

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

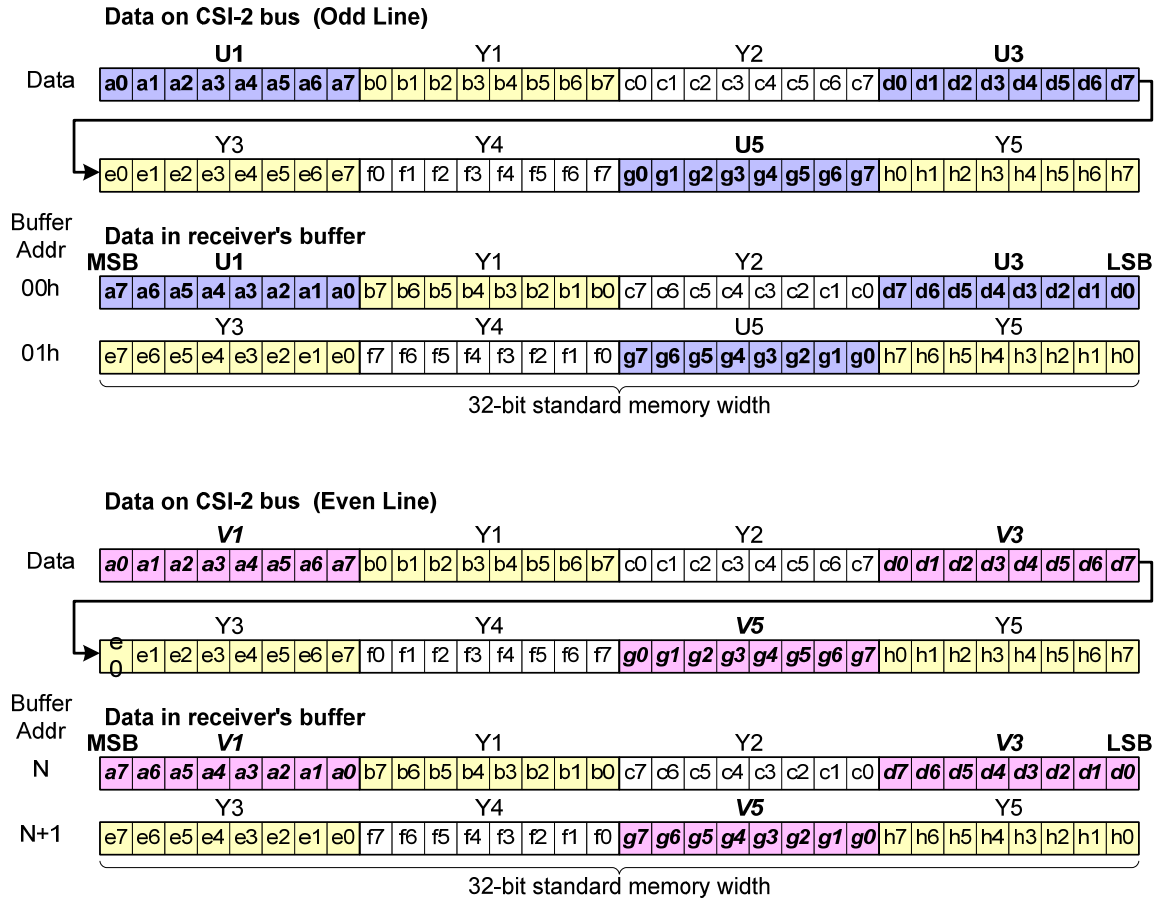


Figure 117 YUV420 8-bit Legacy Data Format Reception

## 12.10 YUV420 8-bit Data Reception

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

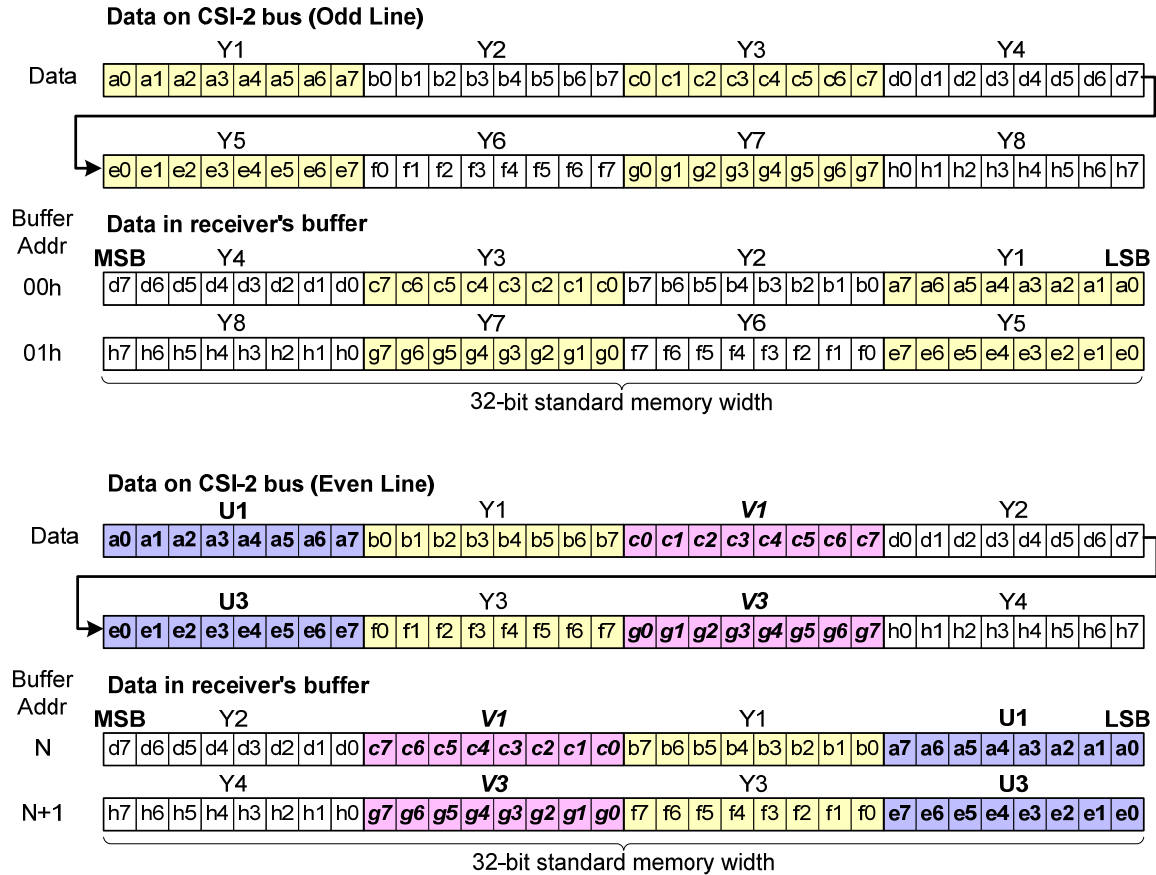


Figure 118 YUV420 8-bit Data Format Reception

## 12.11 YUV420 10-bit Data Reception

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

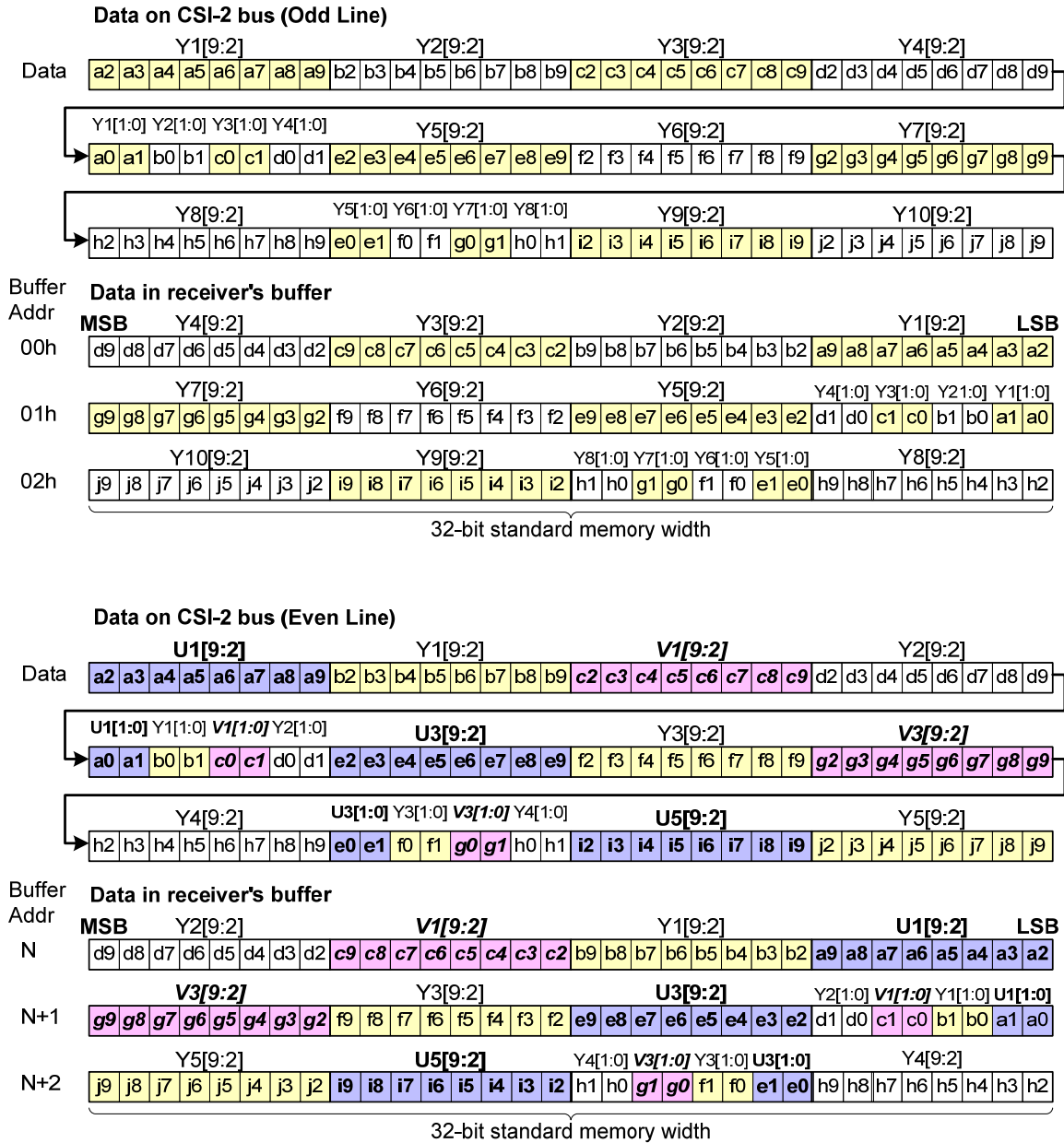


Figure 119 YUV420 10-bit Data Format Reception



## 12.12 RAW6 Data Reception

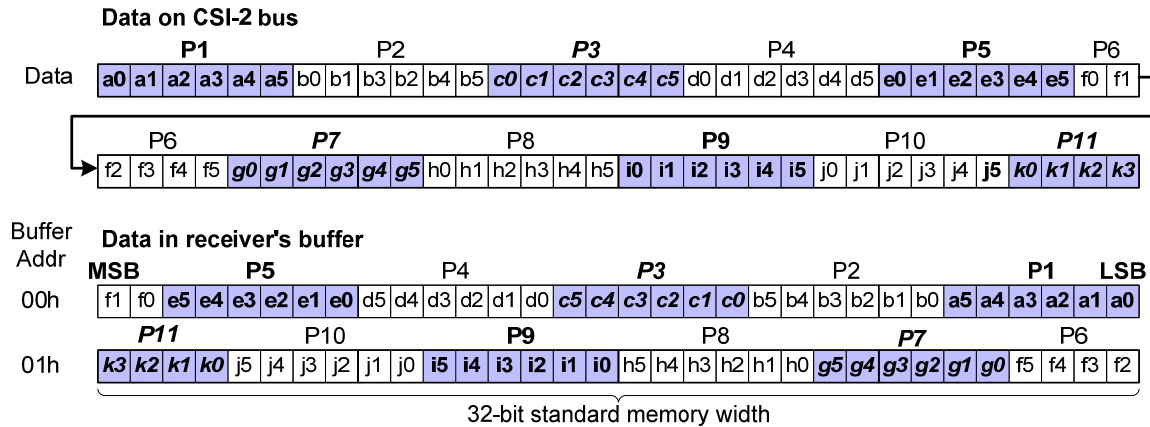


Figure 120 RAW6 Data Format Reception

## 12.13 RAW7 Data Reception

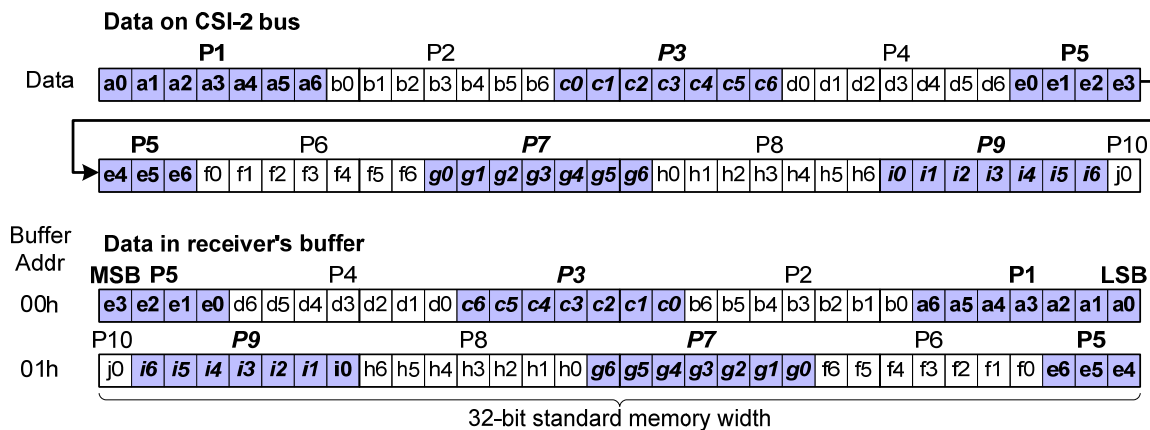


Figure 121 RAW7 Data Format Reception

## 12.14 RAW8 Data Reception

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

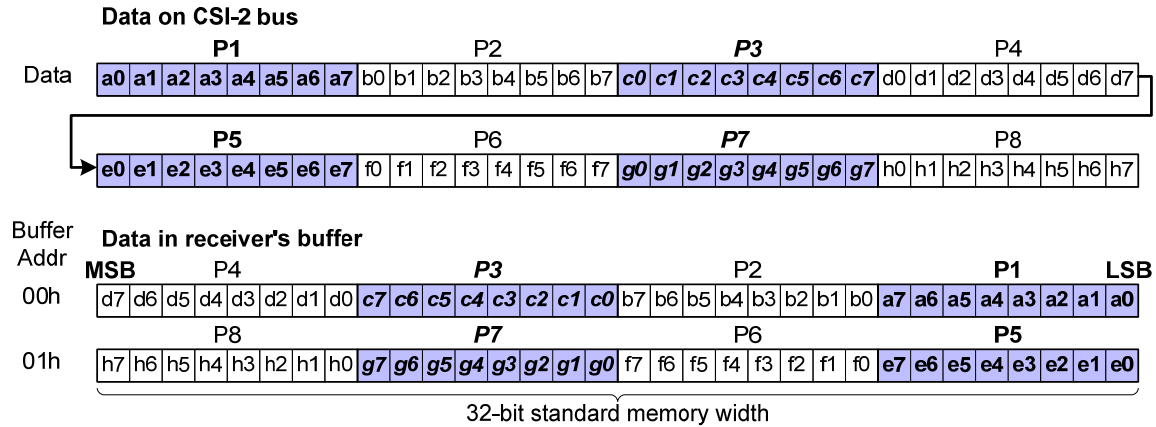


Figure 122 RAW8 Data Format Reception

## 12.15 RAW10 Data Reception

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

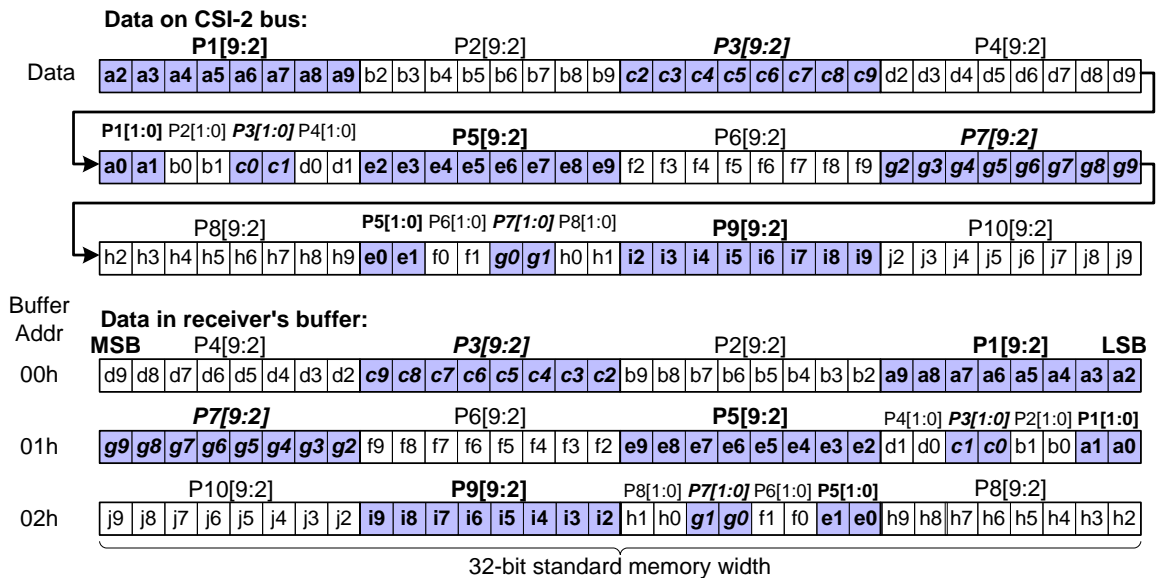


Figure 123 RAW10 Data Format Reception

## 12.16 RAW12 Data Reception

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

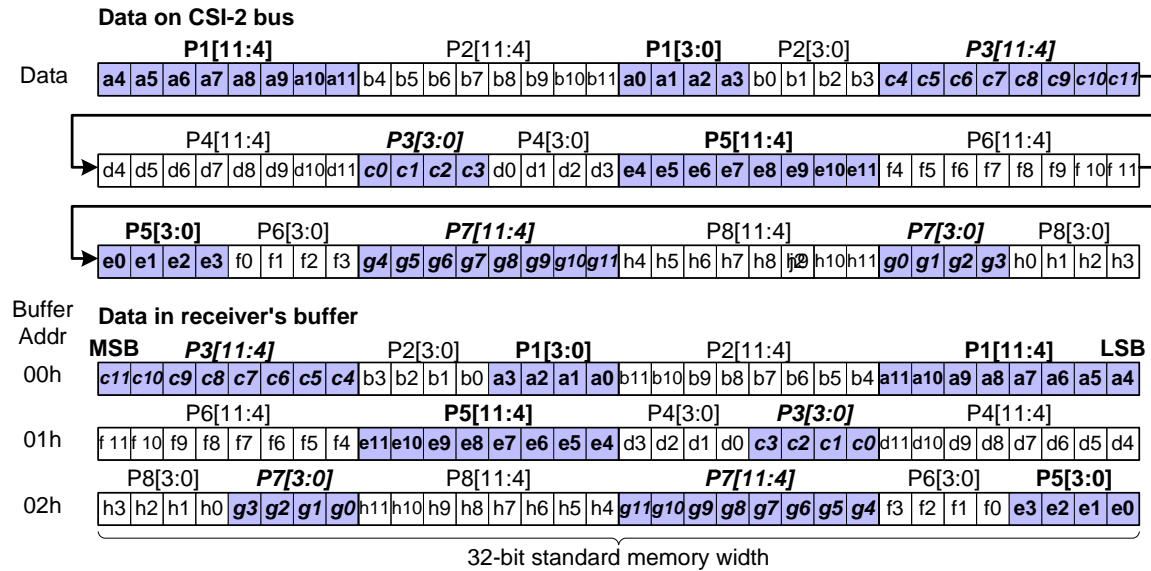


Figure 124 RAW12 Data Format Reception

## 12.17 RAW14 Data Reception

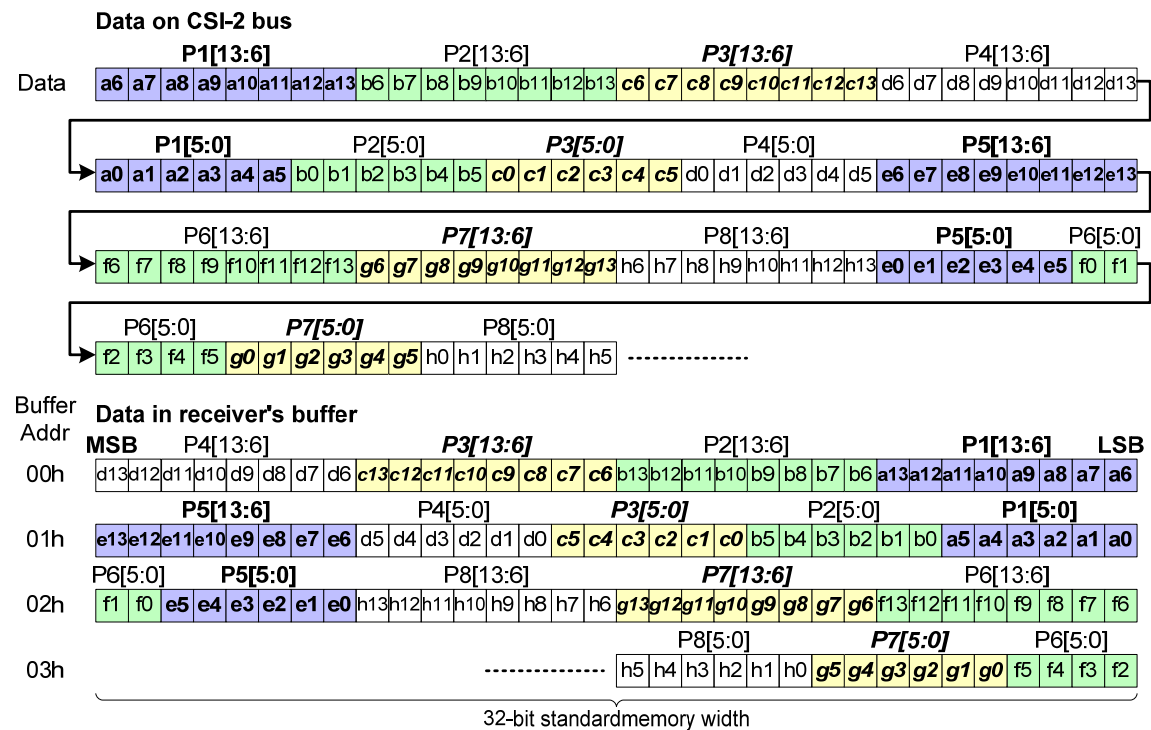


Figure 125 RAW 14 Data Format Reception

## Annex A JPEG8 Data Format (informative)

### A.1 Introduction

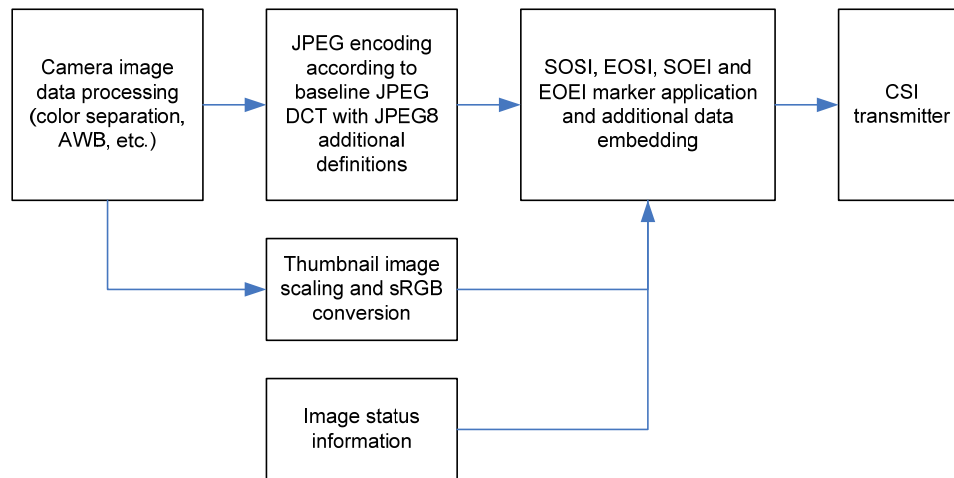
This Annex contains an informative example of the transmission of compressed image data format using the arbitrary Data Type values.

JPEG8 has two non-standard extensions:

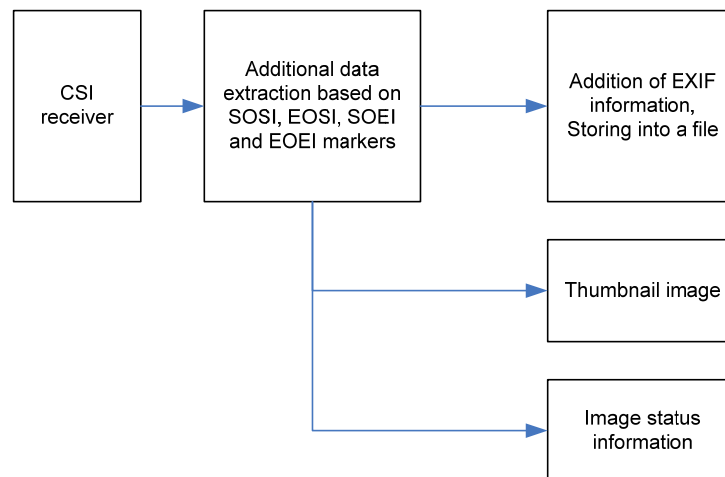
- Status information (mandatory)
- Embedded Image information e.g. a thumbnail image (optional)

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

The JPEG8 data flow is illustrated in the Figure 126 and Figure 127.



**Figure 126 JPEG8 Data Flow in the Encoder**

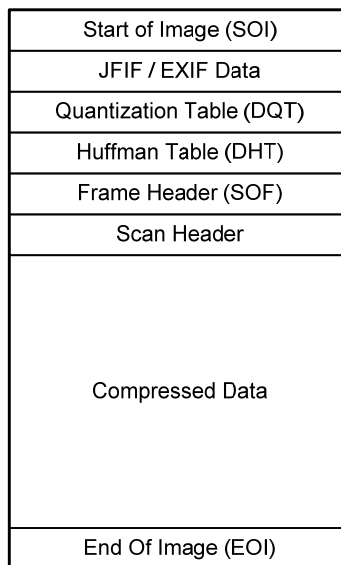


**Figure 127 JPEG8 Data Flow in the Decoder**

## A.2 JPEG Data Definition

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

- sRGB color space shall be used. The JPEG is generated from YCbCr format after sRGB to YCbCr conversion.
- The JPEG metadata has to be EXIF compatible, i.e. metadata within application segments has to be placed in beginning of file, in the order illustrated in Figure 128.
- A status line is added in the end of JPEG data as defined in section A.3.
- If needed, an embedded image is interlaced in order which is free of choice as defined in section A.4.
- Prior to storing into a file, the CSI-2 JPEG data is processed by the data separation process described in section A.1.



**Figure 128 EXIF Compatible Baseline JPEG DCT Format**

## A.3 Image Status Information

Information of at least the following items has to be stored in the end of the JPEG sequence as illustrated in Figure 129:

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

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

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

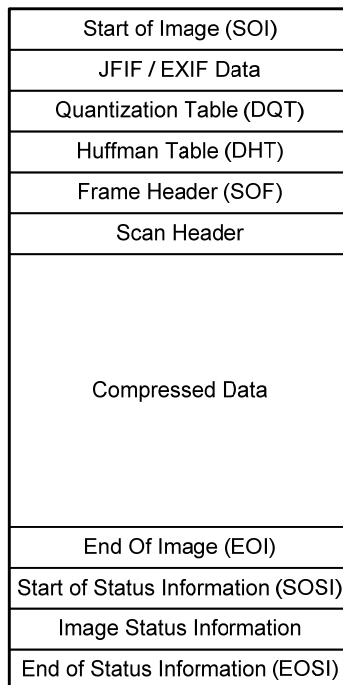
The status information content has to be defined in the product specification of each camera module containing the JPEG8 feature. The format and content is manufacturer specific.

The image status data should be arranged so that each byte is split into two 4-bit nibbles and "1010" padding sequence is added to MSB, as presented in the Table 28. This ensures that no JPEG escape sequences (0xFF 0x00) are present in the status data.

The SOSI and EOSI markers are defined in section A.5.

**Table 28 Status Data Padding**

Data Word	After Padding
D7D6D5D4 D3D2D1D0	1010D7D6D5D4 1010D3D2D1D0



**Figure 129 Status Information Field in the End of Baseline JPEG Frame**

#### A.4 Embedded Images

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

The philosophy of embedded / interleaved thumbnail additions is to minimize the needed frame memory. The EI (Embedded Image) data can be included in any part of the compressed image data segment and in as many pieces as needed. See Figure 130.

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

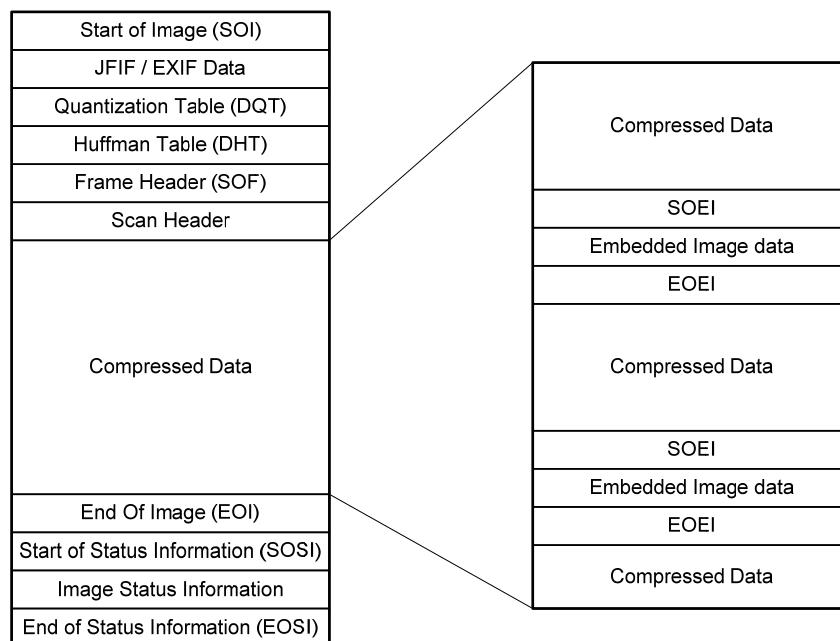
The pixel to byte packing for image data within an EI data field should be as specified for the equivalent CSI-2 data format. However there is an additional restriction; the embedded image data must not generate any false JPEG marker sequences (0xFFXX).

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

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

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

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



**Figure 130 Example of TN Image Embedding Inside the Compressed JPEG Data Block**

## A.5 JPEG8 Non-standard Markers

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

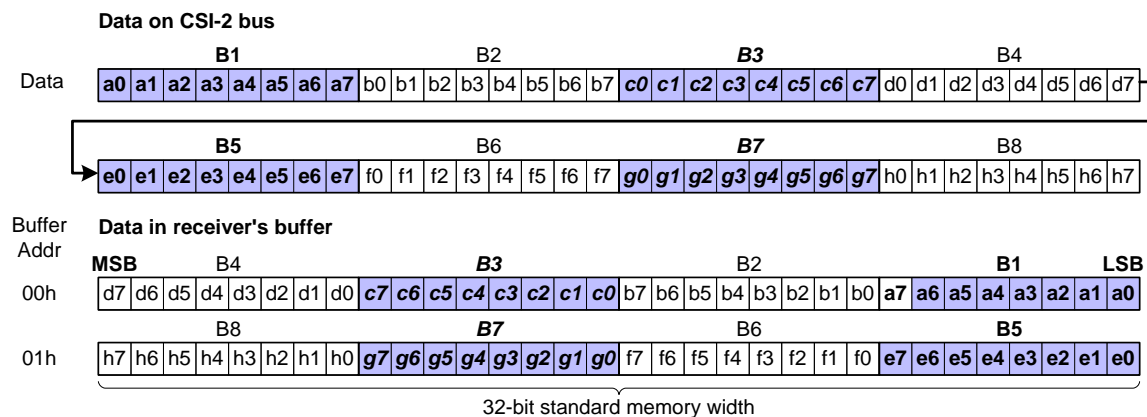
The use of the non-standard markers is always internal to a product containing the JPEG8 camera module, and these markers are always removed from the JPEG data before storing it into a file.

**Table 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

## A.6 JPEG8 Data Reception

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



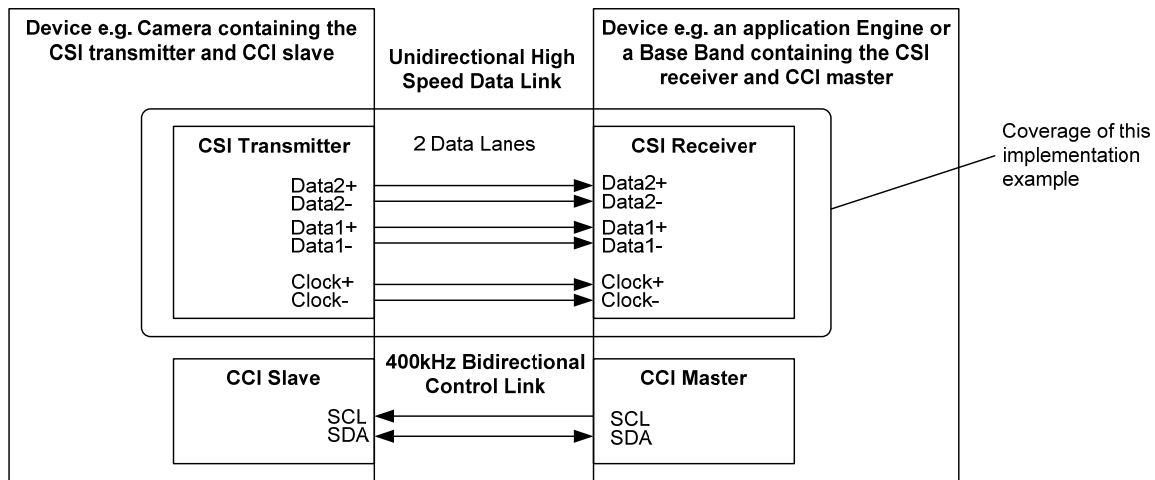
**Figure 131 JPEG8 Data Format Reception**



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

### B.1 Overview

The CSI-2 implementation example assumes that the interface comprises of D-PHY unidirectional Clock and Data, with forward escape mode functionality. The scope in this implementation example refers only to the unidirectional data link without any references to the CCI interface, as it can be seen in Figure 132. This implementation example varies from the informative PPI example in [MIP101].



**Figure 132 Implementation Example Block Diagram and Coverage**

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

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

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

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

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

Using the layered structure described in the overview the CSI-2 transmitter could have the block diagram in Figure 133.

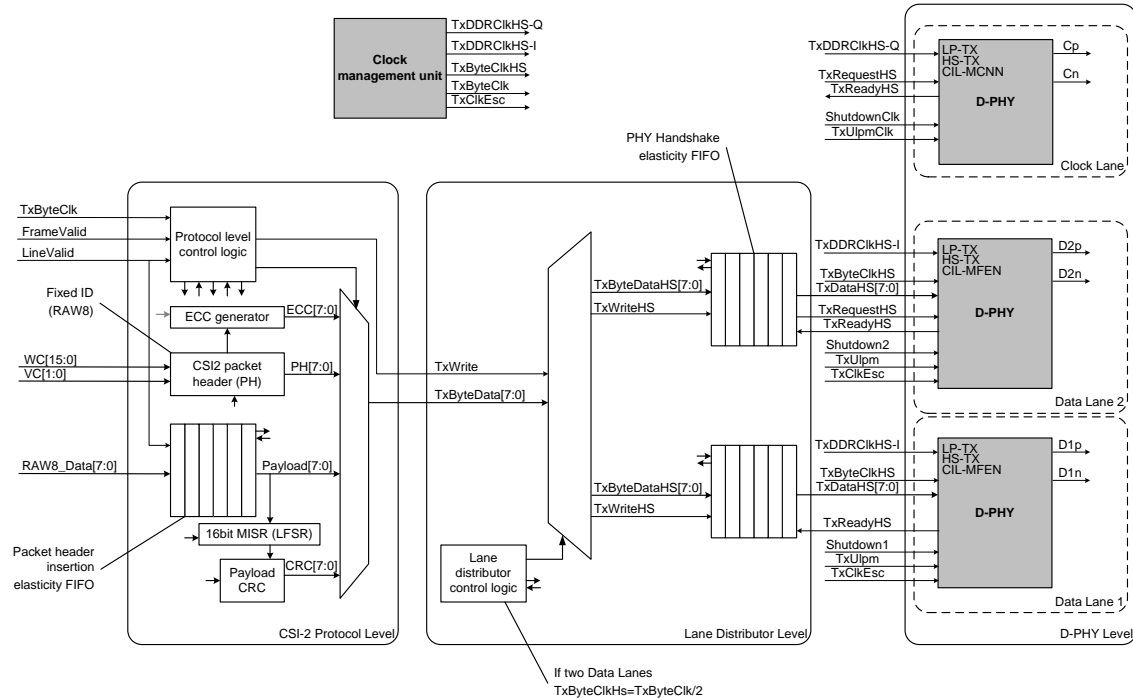


Figure 133 CSI-2 Transmitter Block Diagram

### B.3 CSI-2 Receiver Detailed Block Diagram

Using the layered structure described in the overview, the CSI-2 receiver could have the block diagram in Figure 134.

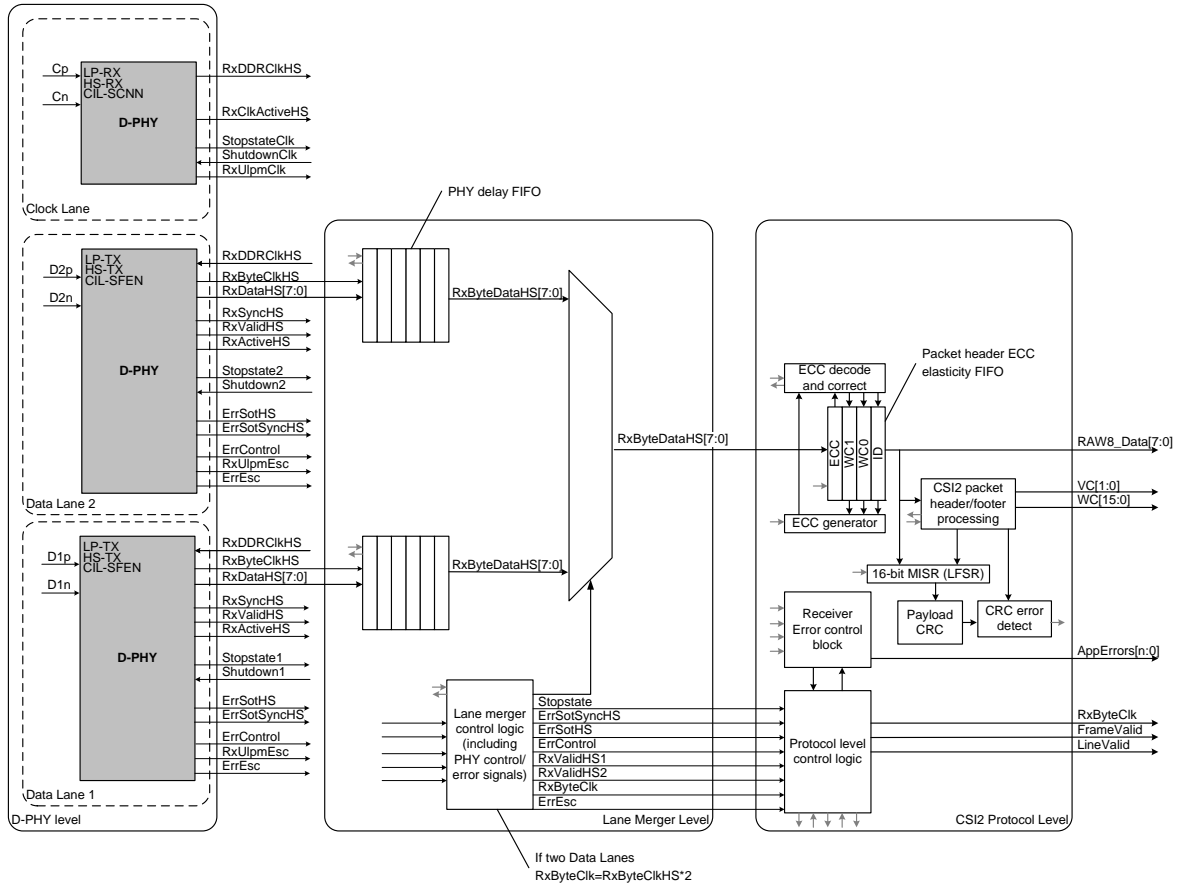


Figure 134 CSI-2 Receiver Block Diagram

## B.4 Details on the D-PHY implementation

The PHY level of implementation has the top level structure as seen in Figure 135.

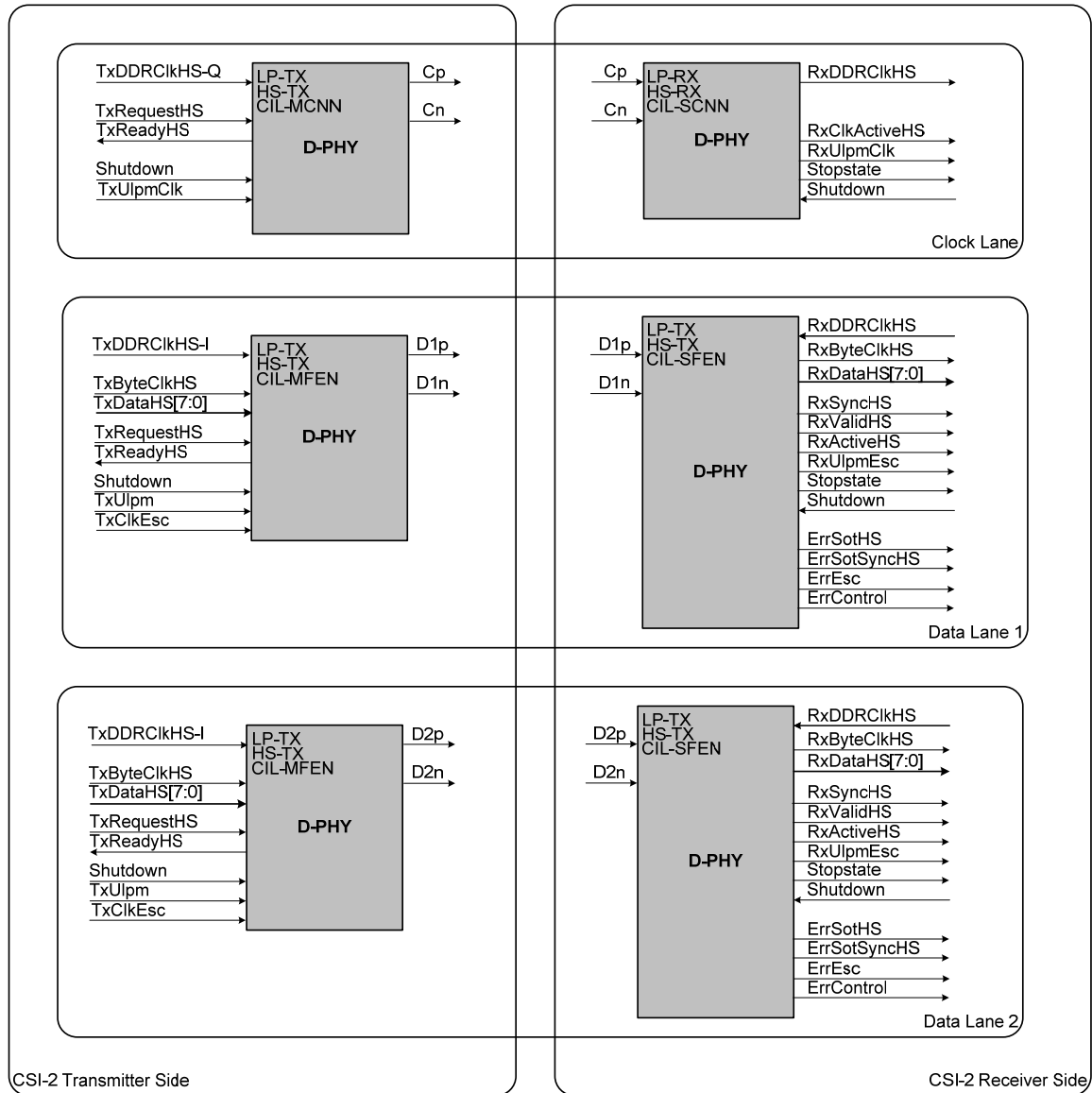


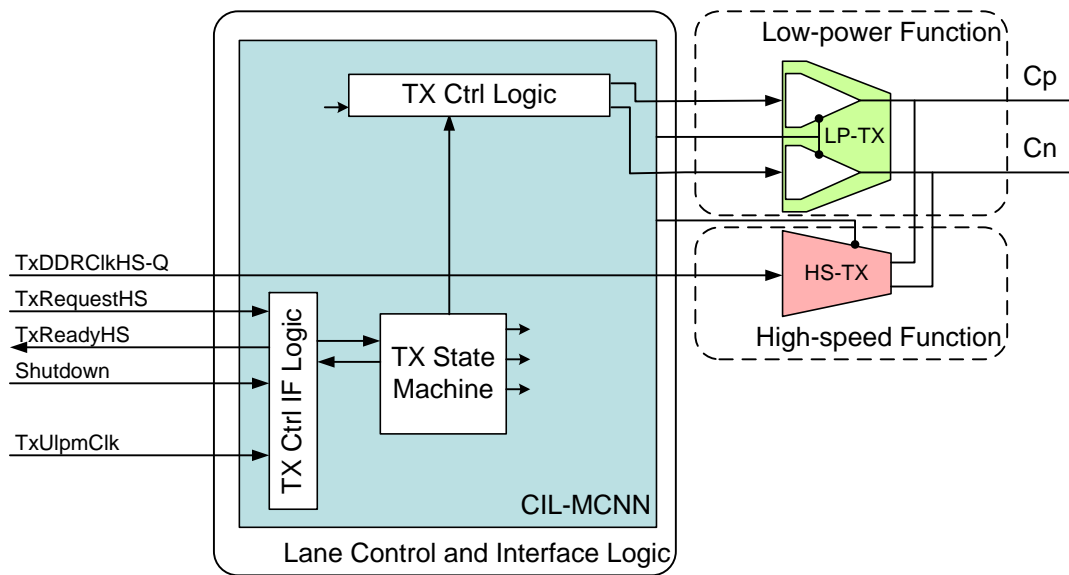
Figure 135 D-PHY Level Block Diagram

The components can be categorized as:

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

### B.4.1 CSI-2 Clock Lane Transmitter

The suggested implementation can be seen in Figure 136.



**Figure 136 CSI-2 Clock Lane Transmitter**

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

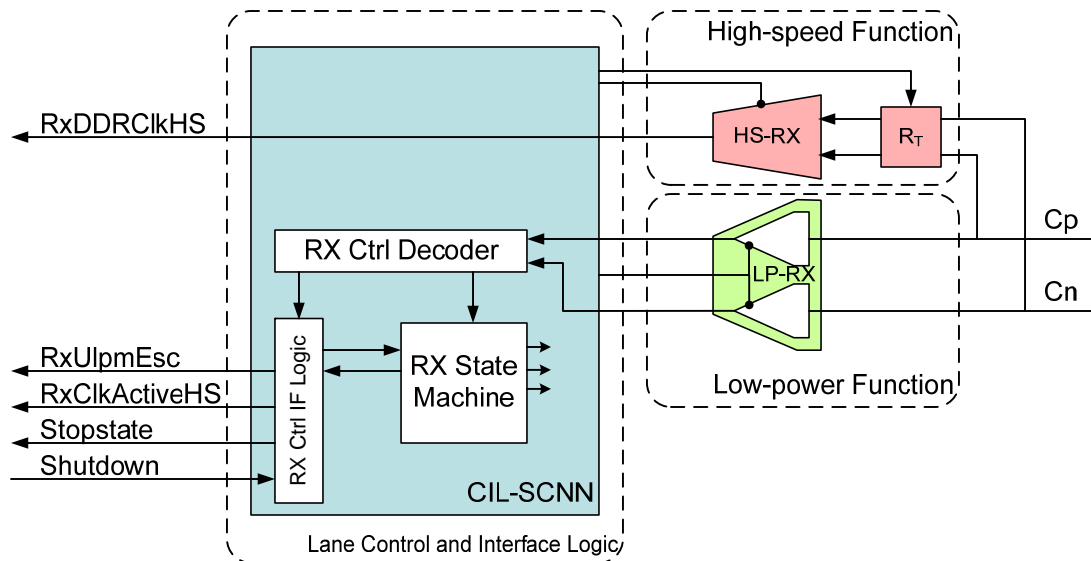
- **LP-TX** for the Low-power function
- **HS-TX** for the High-speed function
- **CIL-MCNN** for the Lane control and interface logic

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

- **TxDDRCIkHS-Q** (Input): High-Speed Transmit DDR Clock (Quadrature).
- **TxRequestHS** (Input): High-Speed Transmit Request. This active high signal causes the lane module to begin transmitting a high-speed clock.
- **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that the clock lane is transmitting HS clock.
- **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into “shutdown”, disabling all activity. All line drivers, including terminators, are turned off when Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on any clock.
- **TxUlpmClk** (Input): Transmit Ultra Low-Power mode on Clock Lane This active high signal is asserted to cause a Clock Lane module to enter the Ultra Low-Power mode. The lane module remains in this mode until TxUlpmClk is de-asserted.

### B.4.2 CSI-2 Clock Lane Receiver

The suggested implementation can be seen in Figure 137.



### Figure 137 CSI-2 Clock Lane Receiver

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

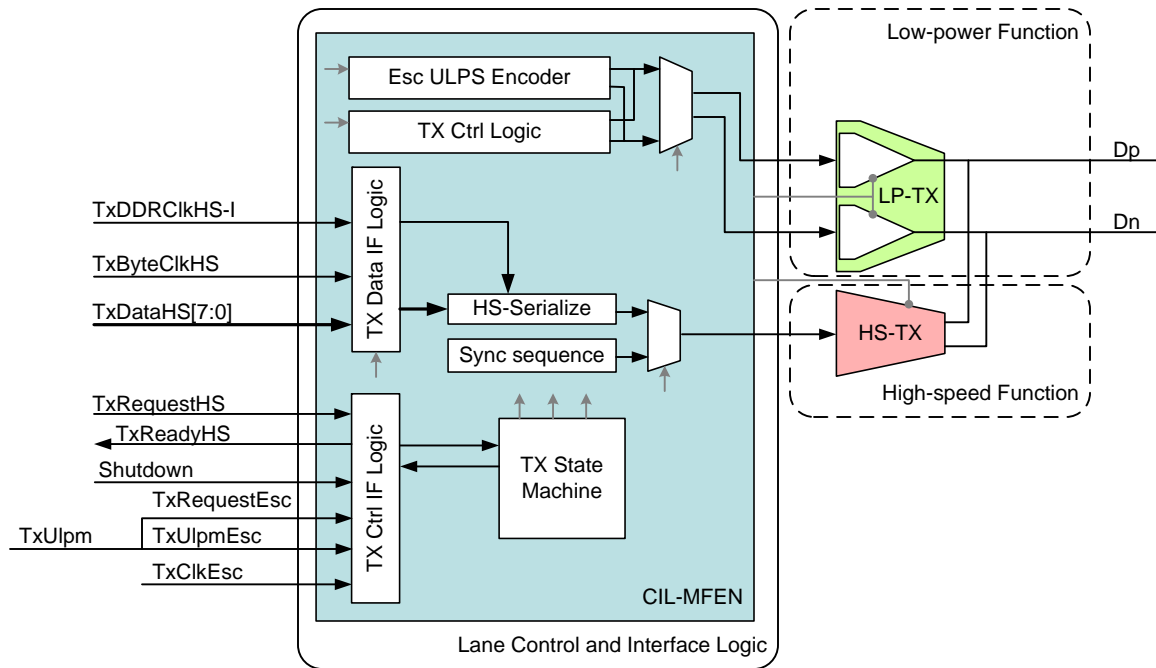
- **LP-RX** for the Low-power function
- **HS-RX** for the High-speed function
- **CIL-SCNN** for the Lane control and interface logic

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

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

### B.4.3 CSI-2 Data Lane Transmitter

The suggested implementation can be seen in Figure 138.



**Figure 138 CSI-2 Data Lane Transmitter**

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

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

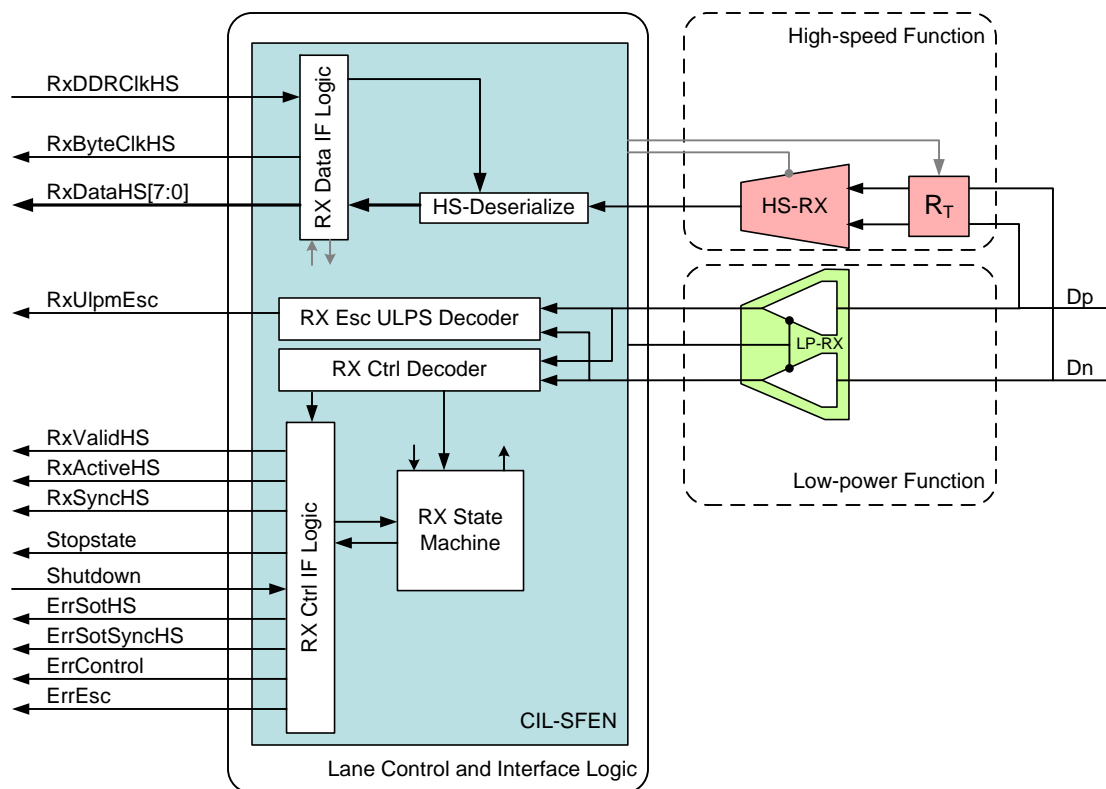
The PPI interface signals to the CSI-2 data lane transmitter are:

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

- **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into “shutdown”, disabling all activity. All line drivers, including terminators, are turned off when Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on any clock.
- **TxUlpmEsc** (Input): Escape mode Transmit Ultra Low Power. This active high signal is asserted with TxRequestEsc to cause the lane module to enter the ultra low power mode. The lane module remains in this mode until TxRequestEsc is de-asserted.
- **TxRequestEsc** (Input): This active high signal, asserted together with TxUlpmEsc is used to request entry into escape mode. Once in escape mode, the lane stays in escape mode until TxRequestEsc is de-asserted. TxRequestEsc is only asserted by the protocol while TxRequestHS is low.
- **TxCclkEsc** (Input): Escape mode Transmit Clock. This clock is directly used to generate escape sequences. The period of this clock determines the symbol time for low power signals. It is therefore constrained by the normative part of the [MIPI01].

#### B.4.4 CSI-2 Data Lane Receiver

The suggested implementation can be seen in Figure 139.



**Figure 139 CSI-2 Data Lane Receiver**

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

- **LP-RX** for the Low-power function
- **HS-RX** for the High-speed function



- **CIL-SFEN** for the Lane control and interface logic

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

- **RxDDRCIkHS** (Input): High-Speed Receive DDR Clock used to sample the data in all data lanes. This signal is supplied by the CSI-2 clock lane receiver.
- **RxByteClkHS** (Output): High-Speed Receive Byte Clock. This signal is used to synchronize signals in the high-speed receive clock domain. The RxByteClkHS is generated by dividing the received RxDDRCIkHS.
- **RXDataHS[7:0]** (Output): High-Speed Receive Data. Eight bit high-speed data received by the lane module. The signal connected to RxDataHS[0] was received first. Data is transferred on rising edges of RxByteClkHS.
- **RxValidHS** (Output): High-Speed Receive Data Valid. This active high signal indicates that the lane module is driving valid data to the protocol on the RxDataHS output. There is no “RxReadyHS” signal, and the protocol is expected to capture RxDataHS on every rising edge of RxByteClkHS where RxValidHS is asserted. There is no provision for the protocol to slow down (“throttle”) the receive data.
- **RxActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the lane module is actively receiving a high-speed transmission from the lane interconnect.
- **RxSyncHS** (Output): Receiver Synchronization Observed. This active high signal indicates that the lane module has seen an appropriate synchronization event. In a typical high-speed transmission, RxSyncHS is high for one cycle of RxByteClkHS at the beginning of a high-speed transmission when RxActiveHS is first asserted. This signal missing is signaled using ErrSotSyncHS.
- **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is asserted to indicate that the lane module has entered the ultra low power mode. The lane module remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane interconnect.
- **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is currently in Stop state. This signal is asynchronous.
- **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into “shutdown”, disabling all activity. All line drivers including terminators, are turned off when Shutdown is asserted. When Shutdown is high, all PPI outputs are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on any clock.
- **ErrSotHS** (Output): Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is corrupted, but in such a way that proper synchronization can still be achieved, this error signal is asserted for one cycle of RxByteClkHS. This is considered to be a “soft error” in the leader sequence and confidence in the payload data is reduced.
- **ErrSotSyncHS** (Output): Start-of-Transmission Synchronization Error. If the high-speed SoT leader sequence is corrupted in a way that proper synchronization cannot be expected, this error is asserted for one cycle of RxByteClkHS.
- **ErrControl** (Output): Control Error. This signal is asserted when an incorrect line state sequence is detected.
- **ErrEsc** (Output): Escape Entry Error. If an unrecognized escape entry command is received, this signal is asserted and remains high until the next change in line state. The only escape entry command supported by the receiver is the ULPS.

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

### C.1 Overview

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

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

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

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

## C.2 D-PHY Level Error

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

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

The recommended receiver error behavior for this level is:

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

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

## C.3 Packet Level Error

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

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

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

1986 Also, computing the CRC code over the whole payload of the received packet could generate the following  
 1987 errors:

- 1988 • **ErrCrc:** Asserted when the computed CRC code is different than the received CRC code.
- 1989 • **ErrID:** Asserted when a Packet Header is decoded with an unrecognized or unimplemented data  
 1990 ID.

1991 The recommended receiver error behavior for this level is:

- 1992 • **ErrEccDouble** should be passed to the Application Layer since assertion of this signal proves that  
 1993 the Packet Header information is corrupt, and therefore the WC is not usable, and thus the packet  
 1994 end cannot be estimated. Commonly, this type of error will be accompanied with an ErrCrc. This  
 1995 type of error should also be passed to the Protocol Decoding Level, since the whole transmission  
 1996 until D-PHY Stop state should be ignored.
- 1997 • **ErrEccCorrected** should be passed to the Application Layer since the application should be  
 1998 informed that an error had occurred but was corrected, so the received Packet Header was  
 1999 unaffected, although the confidence in the data integrity is reduced.
- 2000 • **ErrEccNoError** can be passed to the Protocol Decoding Level to signal the validity of the current  
 2001 Packet Header.
- 2002 • **ErrCrc** should be passed to the Protocol Decoding Level to indicate that the packet's payload data  
 2003 might be corrupt.
- 2004 • **ErrID** should be passed to the Application Layer to indicate that the data packet is unidentified  
 2005 and cannot be unpacked by the receiver. This signal should be asserted after the ID has been  
 2006 identified and de-asserted on the first Frame End (FE) on same virtual channel.

#### 2007 C.4 Protocol Decoding Level Error

2008 The recommended behavior for this error level covers errors caused by decoding the Packet Header  
 2009 information and detecting a sequence that is not allowed by the CSI-2 protocol or a sequence of detected  
 2010 errors by the previous layers. CSI-2 implementers will commonly choose to implement this level of error  
 2011 handling using a state machine that should be paired with the corresponding virtual channel. The state  
 2012 machine should generate at least the following error signals:

- 2013 • **ErrFrameSync:** Asserted when a Frame End (FE) is not paired with a Frame Start (FS) on the  
 2014 same virtual channel. A ErrSotSyncHS should also generate this error signal.
- 2015 • **ErrFrameData:** Asserted after a FE when the data payload received between FS and FE contains  
 2016 errors.

2017 The recommended receiver error behavior for this level is:

- 2018 • **ErrFrameSync** should be passed to the Application Layer with the corresponding virtual channel,  
 2019 since the frame could not be successfully identified. Several error cases on the same virtual  
 2020 channel can be identified for this type of error.
- 2021 • If a FS is followed by a second FS on the same virtual channel, the frame corresponding to the  
 2022 first FS is considered in error.
- 2023 • If a Packet Level ErrEccDouble was signaled from the Protocol Layer, the whole transmission  
 2024 until the first D-PHY Stop-state should be ignored since it contains no information that can be  
 2025 safely decoded and cannot be qualified with a data valid signal.
- 2026 • If a FE is followed by a second FE on the same virtual channel, the frame corresponding to  
 2027 the second FE is considered in error.

- 2028
- 2029
- 2030
- If an ErrSotSyncHS was signaled from the PHY Layer, the whole transmission until the first D-PHY Stop state should be ignored since it contains no information that can be safely decoded and cannot be qualified with a data valid signal.
- 2031
- 2032
- **ErrFrameData:** should be passed to the Application Layer to indicate that the frame contains data errors. This signal should be asserted on any ErrCrc and de-asserted on the first FE.
- 2033

## Annex D CSI-2 Sleep Mode (informative)

### D.1 Overview

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

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

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

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

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

### D.2 SLM Command Phase

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

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

### D.3 SLM Entry Phase

For the second phase, consider one option:

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

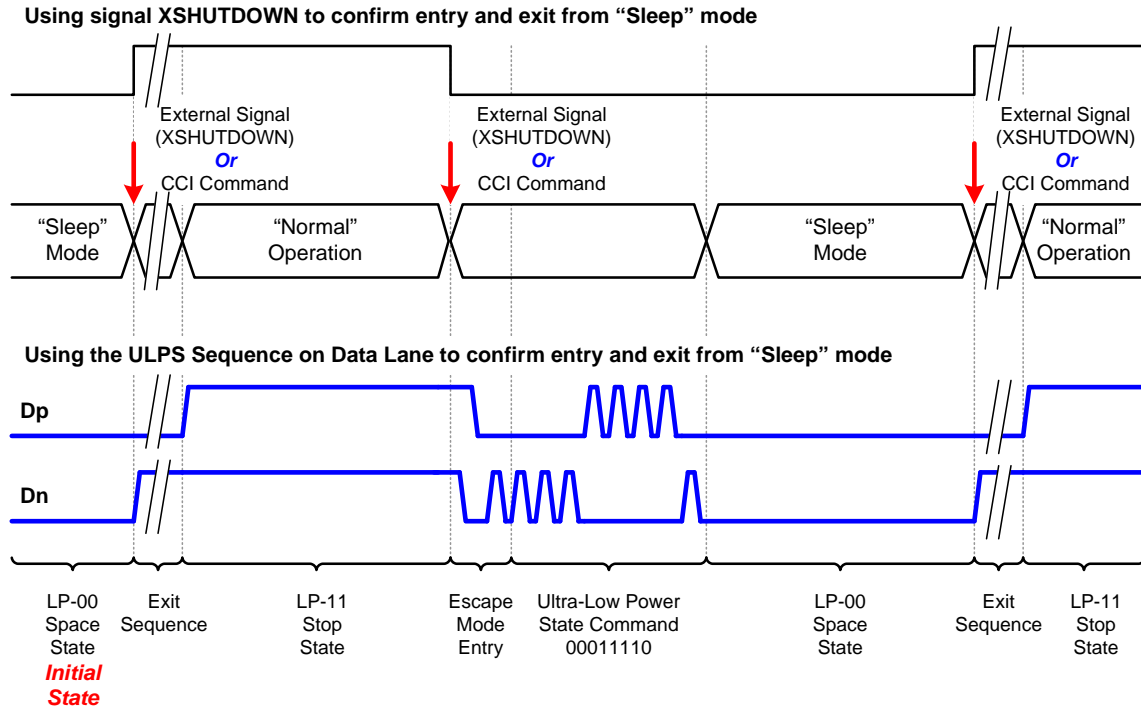


Figure 140 SLM Synchronization

#### 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 have been in 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.

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

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

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

The following data compression schemes are defined:

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

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

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

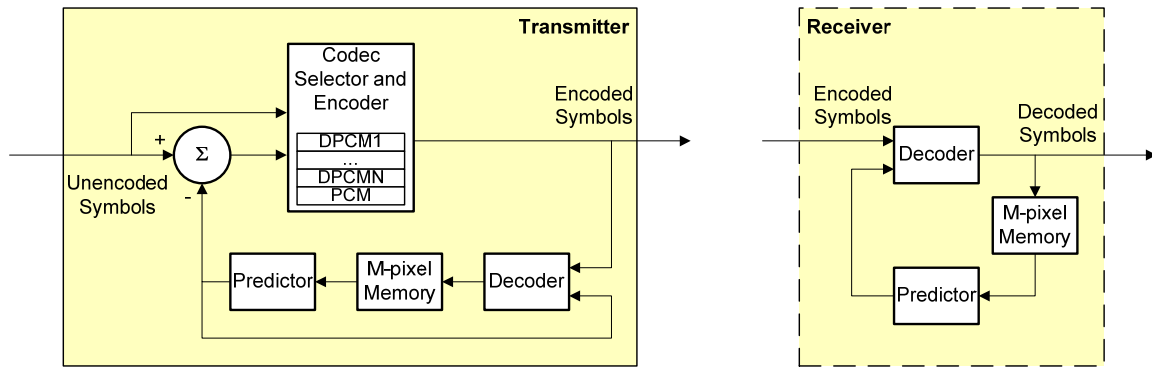
The data compression schemes in this annex are lossy and designed to encode each line independent of the other lines in the image.

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

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

The data compression system consists of encoder, decoder and predictor blocks as shown in Figure 141.





**Figure 141 Data Compression System Block Diagram**

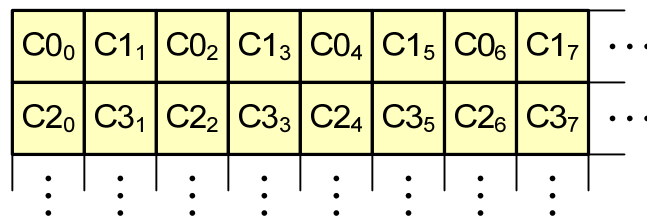
The encoder uses a simple algorithm to encode the pixel values. A fixed number of pixel values at the beginning of each line are encoded without using prediction. These first few values are used to initialize the predictor block. The remaining pixel values on the line are encoded using prediction.

If the predicted value of the pixel,  $X_{pred}$ , is close enough to the original value of the pixel,  $X_{orig}$ , ( $abs(X_{orig} - X_{pred}) < \text{difference limit}$ ) its difference value,  $X_{diff}$ , is quantized using a DPCM codec. Otherwise,  $X_{orig}$  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,  $X_{enco}$ .

## E.1 Predictors

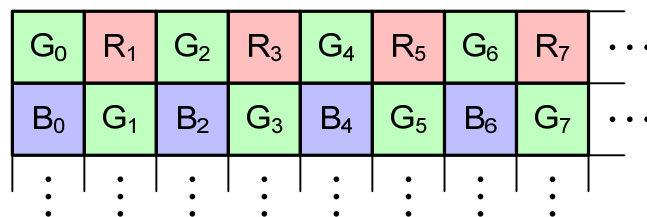
In order to have meaningful data transfer, both the transmitter and the receiver need to use the same predictor block.

The order of pixels in a raw image is shown in Figure 142.



**Figure 142 Pixel Order of the Original Image**

Figure 143 shows an example of the pixel order with RGB data.



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

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

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

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

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

#### 2146 **E.1.1 Predictor1**

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

The first two pixels ( $C0_0$ ,  $C1_1$  /  $C2_0$ ,  $C3_1$  or as in example  $G_0$ ,  $R_1$  /  $B_0$ ,  $G_1$ ) in a line are encoded without prediction.

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

$$2153 \quad \mathbf{Xpred(n)} = \mathbf{Xdeco(n-2)}$$

#### 2154 **E.1.2 Predictor2**

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

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

The first pixel ( $C0_0$  /  $C2_0$ , or as in example  $G_0$  /  $B_0$ ) in a line is coded without prediction.

The second pixel ( $C1_1$  /  $C3_1$  or as in example  $R_1$  /  $G_1$ ) in a line is predicted using the previous decoded different color value as a prediction value. The predictor equation for the second pixel is shown below:

$$2163 \quad \mathbf{Xpred(n)} = \mathbf{Xdeco(n-1)}$$

The third pixel ( $C0_2$  /  $C2_2$  or as in example  $G_2$  /  $B_2$ ) in a line is predicted using the previous decoded same color value as a prediction value. The predictor equation for the third pixel is shown below:

$$2166 \quad \mathbf{Xpred(n)} = \mathbf{Xdeco(n-2)}$$

The fourth pixel ( $C1_3$  /  $C3_3$  or as in example  $R_3$  /  $G_3$ ) in a line is predicted using the following equation:

```

2168     if ((Xdeco(n-1) <= Xdeco(n-2) AND Xdeco(n-2) <= Xdeco(n-3)) OR
2169         (Xdeco(n-1) >= Xdeco(n-2) AND Xdeco(n-2) >= Xdeco(n-3))) then
2170         Xpred(n) = Xdeco(n-1)
2171     else
2172         Xpred(n) = Xdeco(n-2)
2173     endif

```

2174 Other pixels in all lines are predicted using the equation:

```

2175     if ((Xdeco( n-1 ) <= Xdeco( n-2 ) AND Xdeco( n-2 ) <= Xdeco( n-3 )) OR
2176         (Xdeco( n-1 ) >= Xdeco( n-2 ) AND Xdeco( n-2 ) >= Xdeco( n-3 ))) then
2177         Xpred( n ) = Xdeco( n-1 )
2178     else if ((Xdeco( n-1 ) <= Xdeco( n-3 ) AND Xdeco( n-2 ) <= Xdeco( n-4 )) OR
2179         (Xdeco( n-1 ) >= Xdeco( n-3 ) AND Xdeco( n-2 ) >= Xdeco( n-4 ))) then
2180         Xpred( n ) = Xdeco( n-2 )
2181     else
2182         Xpred( n ) = (Xdeco( n-2 ) + Xdeco( n-4 ) + 1) / 2
2183     endif

```

## 2184 E.2 Encoders

2185 There are six different encoders available, one for each data compression scheme.

2186 For all encoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula  
2187 for predicted pixels.

### 2188 E.2.1 Coder for 10–8–10 Data Compression

2189 The 10–8–10 coder offers a 20% bit rate reduction with very high image quality.

2190 Pixels without prediction are encoded using the following formula:

2191 
$$\mathbf{Xenco( n ) = Xorig( n ) / 4}$$

2192 To avoid a full-zero encoded value, the following check is performed:

```

2193     if (Xenco( n ) == 0) then
2194         Xenco( n ) = 1
2195     endif

```

2196 Pixels with prediction are encoded using the following formula:

```

2197     if (abs(Xdiff( n )) < 32) then
2198         use DPCM1
2199     else if (abs(Xdiff( n )) < 64) then
2200         use DPCM2
2201     else if (abs(Xdiff( n )) < 128) then
2202         use DPCM3
2203     else
2204         use PCM
2205     endif

```

#### 2206 E.2.1.1 DPCM1 for 10–8–10 Coder

2207 **Xenco( n )** has the following format:

2208 
$$\mathbf{Xenco( n ) = "00 s xxxxx"}$$

2209           where,

2210           “00” is the code word

2211           “s” is the **sign** bit

2212           “xxxxx” is the five bit **value** field

2213   The coder equation is described as follows:

2214           if (**Xdiff**( n ) <= 0) then

2215               **sign** = 1

2216           else

2217               **sign** = 0

2218           endif

2219           **value** = **abs**(**Xdiff**( n ))

2220   Note: Zero code has been avoided (0 is sent as -0).

#### 2221   **E.2.1.2       DPCM2 for 10–8–10 Coder**

2222   **Xenco**( n ) has the following format:

2223           **Xenco**( n ) = “010 s xxxx”

2224           where,

2225           “010” is the code word

2226           “s” is the **sign** bit

2227           “xxxx” is the four bit **value** field

2228   The coder equation is described as follows:

2229           if (**Xdiff**( n ) < 0) then

2230               **sign** = 1

2231           else

2232               **sign** = 0

2233           endif

2234           **value** = (**abs**(**Xdiff**( n )) - 32) / 2

2235

#### 2236   **E.2.1.3       DPCM3 for 10–8–10 Coder**

2237   **Xenco**( n ) has the following format:

2238           **Xenco**( n ) = “011 s xxxx”

2239           where,

2240           “011” is the code word

2241           “s” is the **sign** bit

2242           “xxxx” is the four bit **value** field

2243 The coder equation is described as follows:

```

2244     if (Xdiff( n ) < 0) then
2245         sign = 1
2246     else
2247         sign = 0
2248     endif
2249     value = (abs(Xdiff( n )) - 64) / 4

```

#### 2250 **E.2.1.4 PCM for 10–8–10 Coder**

2251 **Xenco**( n ) has the following format:

2252 **Xenco**( n ) = “1 xxxxxxx”

2253 where,

```

2254     “1” is the code word
2255     the sign bit is not used
2256     “xxxxxxx” is the seven bit value field

```

2257 The coder equation is described as follows:

2258 **value** = **Xorig**( n ) / 8

#### 2259 **E.2.2 Coder for 10–7–10 Data Compression**

2260 The 10–7–10 coder offers 30% bit rate reduction with high image quality.

2261 Pixels without prediction are encoded using the following formula:

2262 **Xenco**( n ) = **Xorig**( n ) / 8

2263 To avoid a full-zero encoded value, the following check is performed:

```

2264     if (Xenco( n ) == 0) then
2265         Xenco( n ) = 1

```

2266 Pixels with prediction are encoded using the following formula:

```

2267     if (abs(Xdiff( n )) < 8) then
2268         use DPCM1
2269     else if (abs(Xdiff( n )) < 16) then
2270         use DPCM2
2271     else if (abs(Xdiff( n )) < 32) then
2272         use DPCM3
2273     else if (abs(Xdiff( n )) < 160) then
2274         use DPCM4
2275     else
2276         use PCM
2277     endif

```

2278 **E.2.2.1 DPCM1 for 10–7–10 Coder**

2279 **Xenco( n )** has the following format:

2280 **Xenco( n )** = “000 s xxx”

2281 where,

2282 “000” is the code word

2283 “s” is the **sign** bit

2284 “xxx” is the three bit **value** field

2285 The coder equation is described as follows:

2286 if (**Xdiff( n )** <= 0) then

2287 **sign** = 1

2288 else

2289 **sign** = 0

2290 endif

2291 **value** = abs(**Xdiff( n )**)

2292 Note: Zero code has been avoided (0 is sent as -0).

2293 **E.2.2.2 DPCM2 for 10–7–10 Coder**

2294 **Xenco( n )** has the following format:

2295 **Xenco( n )** = “0010 s xx”

2296 where,

2297 “0010” is the code word

2298 “s” is the **sign** bit

2299 “xx” is the two bit **value** field

2300 The coder equation is described as follows:

2301 if (**Xdiff( n )** < 0) then

2302 **sign** = 1

2303 else

2304 **sign** = 0

2305 endif

2306 **value** = (abs(**Xdiff( n )**) - 8) / 2

2307 **E.2.2.3 DPCM3 for 10–7–10 Coder**

2308 **Xenco( n )** has the following format:

2309 **Xenco( n )** = “0011 s xx”

2310           where,  
 2311           “0011” is the code word  
 2312           “s” is the **sign** bit  
 2313           “xx” is the two bit **value** field

2314   The coder equation is described as follows:

2315           if ( $X_{diff}(n) < 0$ ) then  
 2316           **sign** = 1  
 2317           else  
 2318           **sign** = 0  
 2319           endif  
 2320           **value** =  $(abs(X_{diff}(n)) - 16) / 4$

#### 2321   **E.2.2.4       DPCM4 for 10–7–10 Coder**

2322   **Xenco( n )** has the following format:

2323           **Xenco( n )** = “01 s xxxx”

2324           where,

2325           “01” is the code word  
 2326           “s” is the **sign** bit  
 2327           “xxxx” is the four bit **value** field

2328   The coder equation is described as follows:

2329           if ( $X_{diff}(n) < 0$ ) then  
 2330           **sign** = 1  
 2331           else  
 2332           **sign** = 0  
 2333           endif  
 2334           **value** =  $(abs(X_{diff}(n)) - 32) / 8$

#### 2335   **E.2.2.5       PCM for 10–7–10 Coder**

2336   **Xenco( n )** has the following format:

2337           **Xenco( n )** = “1 xxxxxx”

2338           where,

2339           “1” is the code word  
 2340           the **sign** bit is not used  
 2341           “xxxxxx” is the six bit **value** field

2342   The coder equation is described as follows:

2343           **value** =  $X_{orig}(n) / 16$

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

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

Pixels without prediction are encoded using the following formula:

$$\mathbf{Xenco( n ) = Xorig( n ) / 16}$$

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

```

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

```

Pixels with prediction are encoded using the following formula:

```

if (abs(Xdiff( n )) < 1) then
    use DPCM1
else if (abs(Xdiff( n )) < 3) then
    use DPCM2
else if (abs(Xdiff( n )) < 11) then
    use DPCM3
else if (abs(Xdiff( n )) < 43) then
    use DPCM4
else if (abs(Xdiff( n )) < 171) then
    use DPCM5
else
    use PCM
endif

```

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

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

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

where,

```

"00000" is the code word
"s" is the sign bit
the value field is not used

```

The coder equation is described as follows:

$$\mathbf{sign = 1}$$

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

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

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

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



2379           where,

2380           “00001” is the code word

2381           “s” is the **sign** bit

2382           the **value** field is not used

2383   The coder equation is described as follows:

2384           if ( $X_{diff}(n) < 0$ ) then

2385           **sign** = 1

2386           else

2387           **sign** = 0

2388           endif

#### 2389   **E.2.3.3       DPCM3 for 10–6–10 Coder**

2390   **Xenco( n )** has the following format:

2391           **Xenco( n )** = “0001 s x”

2392           where,

2393           “0001” is the code word

2394           “s” is the **sign** bit

2395           “x” is the one bit **value** field

2396   The coder equation is described as follows:

2397           if ( $X_{diff}(n) < 0$ ) then

2398           **sign** = 1

2399           else

2400           **sign** = 0

2401           **value** = ( $\text{abs}(X_{diff}(n)) - 3$ ) / 4

2402           endif

#### 2403   **E.2.3.4       DPCM4 for 10–6–10 Coder**

2404   **Xenco( n )** has the following format:

2405           **Xenco( n )** = “001 s xx”

2406           where,

2407           “001” is the code word

2408           “s” is the **sign** bit

2409           “xx” is the two bit **value** field

2410   The coder equation is described as follows:

2411           if ( $X_{diff}(n) < 0$ ) then

2412           **sign** = 1

2413           else

2414           **sign** = 0

2415           endif

2416            **value** = (abs(**Xdiff**( **n** )) - 11) / 8

#### 2417    **E.2.3.5        DPCM5 for 10–6–10 Coder**

2418    **Xenco**( **n** ) has the following format:

2419            **Xenco**( **n** ) = “01 s xxx”

2420            where,

2421            “01” is the code word

2422            “s” is the **sign** bit

2423            “xxx” is the three bit **value** field

2424    The coder equation is described as follows:

2425            if (**Xdiff**( **n** ) < 0) then

2426                **sign** = 1

2427            else

2428                **sign** = 0

2429            endif

2430            **value** = (abs(**Xdiff**( **n** )) - 43) / 16

#### 2431    **E.2.3.6        PCM for 10–6–10 Coder**

2432    **Xenco**( **n** ) has the following format:

2433            **Xenco**( **n** ) = “1 xxxxx”

2434            where,

2435            “1” is the code word

2436            the **sign** bit is not used

2437            “xxxxx” is the five bit **value** field

2438    The coder equation is described as follows:

2439            **value** = **Xorig**( **n** ) / 32

#### 2440    **E.2.4        Coder for 12–8–12 Data Compression**

2441    The 12–8–12 coder offers 33% bit rate reduction with very high image quality.

2442    Pixels without prediction are encoded using the following formula:

2443            **Xenco**( **n** ) = **Xorig**( **n** ) / 16

2444    To avoid a full-zero encoded value, the following check is performed:

2445            if (**Xenco**( **n** ) == 0) then

2446                **Xenco**( **n** ) = 1

2447            endif

2448 Pixels with prediction are encoded using the following formula:

```

2449     if (abs(Xdiff( n )) < 8) then
2450         use DPCM1
2451     else if (abs(Xdiff( n )) < 40) then
2452         use DPCM2
2453     else if (abs(Xdiff( n )) < 104) then
2454         use DPCM3
2455     else if (abs(Xdiff( n )) < 232) then
2456         use DPCM4
2457     else if (abs(Xdiff( n )) < 360) then
2458         use DPCM5
2459     else
2460         use PCM

```

#### 2461 **E.2.4.1 DPCM1 for 12–8–12 Coder**

2462 **Xenco**( n ) has the following format:

2463 **Xenco**( n ) = “0000 s xxx”

2464 where,

```

2465     “0000” is the code word
2466     “s” is the sign bit
2467     “xxx” is the three bit value field

```

2468 The coder equation is described as follows:

```

2469     if (Xdiff( n ) <= 0) then
2470         sign = 1
2471     else
2472         sign = 0
2473     endif
2474     value = abs(Xdiff( n ))

```

2475 Note: Zero code has been avoided (0 is sent as -0).

#### 2476 **E.2.4.2 DPCM2 for 12–8–12 Coder**

2477 **Xenco**( n ) has the following format:

2478 **Xenco**( n ) = “011 s xxxx”

2479 where,

```

2480     “011” is the code word
2481     “s” is the sign bit
2482     “xxxx” is the four bit value field

```

2483 The coder equation is described as follows:

```

2484     if (Xdiff( n ) < 0) then
2485         sign = 1
2486     else
2487         sign = 0
2488     endif
2489     value = (abs(Xdiff( n )) - 8) / 2

```

#### 2490 **E.2.4.3 DPCM3 for 12–8–12 Coder**

2491 **Xenco**( n ) has the following format:

2492 **Xenco**( n ) = “010 s xxxx”

2493 where,

```

2494     “010” is the code word
2495     “s” is the sign bit
2496     “xxxx” is the four bit value field

```

2497 The coder equation is described as follows:

```

2498     if (Xdiff( n ) < 0) then
2499         sign = 1
2500     else
2501         sign = 0
2502     endif
2503     value = (abs(Xdiff( n )) - 40) / 4

```

#### 2504 **E.2.4.4 DPCM4 for 12–8–12 Coder**

2505 **Xenco**( n ) has the following format:

2506 **Xenco**( n ) = “001 s xxxx”

2507 where,

```

2508     “001” is the code word
2509     “s” is the sign bit
2510     “xxxx” is the four bit value field

```

2511 The coder equation is described as follows:

```

2512     if (Xdiff( n ) < 0) then
2513         sign = 1
2514     else
2515         sign = 0
2516     endif
2517     value = (abs(Xdiff( n )) - 104) / 8

```

2518 **E.2.4.5 DPCM5 for 12–8–12 Coder**

2519 **Xenco( n )** has the following format:

2520 **Xenco( n )** = “0001 s xxx”

2521 where,

2522 “0001” is the code word

2523 “s” is the **sign** bit

2524 “xxx” is the three bit **value** field

2525 The coder equation is described as follows:

2526 if (**Xdiff( n )** < 0) then

2527 **sign** = 1

2528 else

2529 **sign** = 0

2530 endif

2531 **value** = (abs(**Xdiff( n )**) - 232) / 16

2532 **E.2.4.6 PCM for 12–8–12 Coder**

2533 **Xenco( n )** has the following format:

2534 **Xenco( n )** = “1 xxxxxxx”

2535 where,

2536 “1” is the code word

2537 the **sign** bit is not used

2538 “xxxxxxx” is the seven bit **value** field

2539 The coder equation is described as follows:

2540 **value** = **Xorig( n )** / 32

2541 **E.2.5 Coder for 12–7–12 Data Compression**

2542 The 12–7–12 coder offers 42% bit rate reduction with high image quality.

2543 Pixels without prediction are encoded using the following formula:

2544 **Xenco( n )** = **Xorig( n )** / 32

2545 To avoid a full-zero encoded value, the following check is performed:

2546 if (**Xenco( n )** == 0) then

2547 **Xenco( n )** = 1

2548 endif

2549 Pixels with prediction are encoded using the following formula:

```

2550     if (abs(Xdiff( n )) < 4) then
2551         use DPCM1
2552     else if (abs(Xdiff( n )) < 12) then
2553         use DPCM2
2554     else if (abs(Xdiff( n )) < 28) then
2555         use DPCM3
2556     else if (abs(Xdiff( n )) < 92) then
2557         use DPCM4
2558     else if (abs(Xdiff( n )) < 220) then
2559         use DPCM5
2560     else if (abs(Xdiff( n )) < 348) then
2561         use DPCM6
2562     else
2563         use PCM
2564     endif

```

#### 2565 **E.2.5.1 DPCM1 for 12–7–12 Coder**

2566 **Xenco**( n ) has the following format:

2567 **Xenco**( n ) = “0000 s xx”

2568 where,

2569 “0000” is the code word  
 2570 “s” is the **sign** bit  
 2571 “xx” is the two bit **value** field

2572 The coder equation is described as follows:

```

2573     if (Xdiff( n ) <= 0) then
2574         sign = 1
2575     else
2576         sign = 0
2577     endif
2578     value = abs(Xdiff( n ))

```

2579 Note: Zero code has been avoided (0 is sent as -0).

#### 2580 **E.2.5.2 DPCM2 for 12–7–12 Coder**

2581 **Xenco**( n ) has the following format:

2582 **Xenco**( n ) = “0001 s xx”

2583 where,

2584 “0001” is the code word  
 2585 “s” is the **sign** bit  
 2586 “xx” is the two bit **value** field

2587 The coder equation is described as follows:

```

2588     if (Xdiff( n ) < 0) then
2589         sign = 1
2590     else
2591         sign = 0
2592     endif
2593     value = (abs(Xdiff( n )) - 4) / 2

```

#### 2594 **E.2.5.3 DPCM3 for 12–7–12 Coder**

2595 **Xenco**( n ) has the following format:

2596 **Xenco**( n ) = “0010 s xx”

2597 where,

```

2598     “0010” is the code word
2599     “s” is the sign bit
2600     “xx” is the two bit value field

```

2601 The coder equation is described as follows:

```

2602     if (Xdiff( n ) < 0) then
2603         sign = 1
2604     else
2605         sign = 0
2606     endif
2607     value = (abs(Xdiff( n )) - 12) / 4

```

#### 2608 **E.2.5.4 DPCM4 for 12–7–12 Coder**

2609 **Xenco**( n ) has the following format:

2610 **Xenco**( n ) = “010 s xxx”

2611 where,

```

2612     “010” is the code word
2613     “s” is the sign bit
2614     “xxx” is the three bit value field

```

2615 The coder equation is described as follows:

```

2616     if (Xdiff( n ) < 0) then
2617         sign = 1
2618     else
2619         sign = 0
2620     endif
2621     value = (abs(Xdiff( n )) - 28) / 8

```

2622 **E.2.5.5 DPCM5 for 12–7–12 Coder**

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

2624 **Xenco( n )** = “011 s xxx”

2625 where,

2626 “011” is the code word

2627 “s” is the **sign** bit

2628 “xxx” is the three bit **value** field

2629 The coder equation is described as follows:

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

2631 **sign** = 1

2632 else

2633 **sign** = 0

2634 endif

2635 **value** = (abs(**Xdiff( n )**) - 92) / 16

2636 **E.2.5.6 DPCM6 for 12–7–12 Coder**

2637 **Xenco( n )** has the following format:

2638 **Xenco( n )** = “0011 s xx”

2639 where,

2640 “0011” is the code word

2641 “s” is the **sign** bit

2642 “xx” is the two bit **value** field

2643 The coder equation is described as follows:

2644 if (**Xdiff( n )** < 0) then

2645 **sign** = 1

2646 else

2647 **sign** = 0

2648 endif

2649 **value** = (abs(**Xdiff( n )**) - 220) / 32

2650 **E.2.5.7 PCM for 12–7–12 Coder**

2651 **Xenco( n )** has the following format:

2652 **Xenco( n )** = “1 xxxxxx”

2653 where,

2654 “1” is the code word

2655 the **sign** bit is not used

2656 “xxxxxx” is the six bit **value** field



2657 The coder equation is described as follows:

2658       **value** = **Xorig**( **n** ) / 64

## 2659 **E.2.6      Coder for 12–6–12 Data Compression**

2660 The 12–6–12 coder offers 50% bit rate reduction with acceptable image quality.

2661 Pixels without prediction are encoded using the following formula:

2662       **Xenco**( **n** ) = **Xorig**( **n** ) / 64

2663 To avoid a full-zero encoded value, the following check is performed:

2664       if (**Xenco**( **n** ) == 0) then

2665       **Xenco**( **n** ) = 1

2666       endif

2667 Pixels with prediction are encoded using the following formula:

2668       if (abs(**Xdiff**( **n** )) < 2) then

2669       use **DPCM1**

2670       else if (abs(**Xdiff**( **n** )) < 10) then

2671       use **DPCM3**

2672       else if (abs(**Xdiff**( **n** )) < 42) then

2673       use **DPCM4**

2674       else if (abs(**Xdiff**( **n** )) < 74) then

2675       use **DPCM5**

2676       else if (abs(**Xdiff**( **n** )) < 202) then

2677       use **DPCM6**

2678       else if (abs(**Xdiff**( **n** )) < 330) then

2679       use **DPCM7**

2680       else

2681       use **PCM**

2682       endif

2683 Note: **DPCM2** is not used.

### 2684 **E.2.6.1      DPCM1 for 12–6–12 Coder**

2685 **Xenco**( **n** ) has the following format:

2686       **Xenco**( **n** ) = “0000 s x”

2687       where,

2688       “0000” is the code word

2689       “s” is the **sign** bit

2690       “x” is the one bit **value** field

2691 The coder equation is described as follows:

```

2692     if (Xdiff( n ) <= 0) then
2693         sign = 1
2694     else
2695         sign = 0
2696     endif
2697     value = abs(Xdiff( n ))

```

2698 Note: Zero code has been avoided (0 is sent as -0).

#### 2699 **E.2.6.2 DPCM3 for 12–6–12 Coder**

2700 **Xenco( n )** has the following format:

2701 **Xenco( n )** = “0001 s x”

2702 where,

```

2703     “0001” is the code word
2704     “s” is the sign bit
2705     “x” is the one bit value field

```

2706 The coder equation is described as follows:

```

2707     if (Xdiff( n ) < 0) then
2708         sign = 1
2709     else
2710         sign = 0
2711     endif
2712     value = (abs(Xdiff( n )) - 2) / 4

```

#### 2713 **E.2.6.3 DPCM4 for 12–6–12 Coder**

2714 **Xenco( n )** has the following format:

2715 **Xenco( n )** = “010 s xx”

2716 where,

```

2717     “010” is the code word
2718     “s” is the sign bit
2719     “xx” is the two bit value field

```

2720 The coder equation is described as follows:

```

2721     if (Xdiff( n ) < 0) then
2722         sign = 1
2723     else
2724         sign = 0
2725     endif
2726     value = (abs(Xdiff( n )) - 10) / 8

```

2727 **E.2.6.4 DPCM5 for 12–6–12 Coder**

2728 **Xenco( n )** has the following format:

2729 **Xenco( n )** = “0010 s x”

2730 where,

2731 “0010” is the code word

2732 “s” is the **sign** bit

2733 “x” is the one bit **value** field

2734 The coder equation is described as follows:

2735 if (**Xdiff( n )** < 0) then

2736 **sign** = 1

2737 else

2738 **sign** = 0

2739 endif

2740 **value** = (abs(**Xdiff( n )**) - 42) / 16

2741 **E.2.6.5 DPCM6 for 12–6–12 Coder**

2742 **Xenco( n )** has the following format:

2743 **Xenco( n )** = “011 s xx”

2744 where,

2745 “011” is the code word

2746 “s” is the **sign** bit

2747 “xx” is the two bit **value** field

2748 The coder equation is described as follows:

2749 if (**Xdiff( n )** < 0) then

2750 **sign** = 1

2751 else

2752 **sign** = 0

2753 endif

2754 **value** = (abs(**Xdiff( n )**) - 74) / 32

2755 **E.2.6.6 DPCM7 for 12–6–12 Coder**

2756 **Xenco( n )** has the following format:

2757 **Xenco( n )** = “0011 s x”

2758 where,

2759 “0011” is the code word

2760 “s” is the **sign** bit

2761 “x” is the one bit **value** field

2762 The coder equation is described as follows:

```

2763     if (Xdiff( n ) < 0) then
2764         sign = 1
2765     else
2766         sign = 0
2767     endif
2768     value = (abs(Xdiff( n )) - 202) / 64

```

#### 2769 **E.2.6.7 PCM for 12–6–12 Coder**

2770 **Xenco**( n ) has the following format:

2771 **Xenco**( n ) = “1 xxxxx”

2772 where,

```

2773     “1” is the code word
2774     the sign bit is not used
2775     “xxxxx” is the five bit value field

```

2776 The coder equation is described as follows:

2777 **value** = **Xorig**( n ) / 128

### 2778 **E.3 Decoders**

2779 There are six different decoders available, one for each data compression scheme.

2780 For all decoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula  
 2781 for predicted pixels.

#### 2782 **E.3.1 Decoder for 10–8–10 Data Compression**

2783 Pixels without prediction are decoded using the following formula:

2784 **Xdeco**( n ) = 4 \* **Xenco**( n ) + 2

2785 Pixels with prediction are decoded using the following formula:

```

2786     if (Xenco( n ) & 0xc0 == 0x00) then
2787         use DPCM1
2788     else if (Xenco( n ) & 0xe0 == 0x40) then
2789         use DPCM2
2790     else if (Xenco( n ) & 0xe0 == 0x60) then
2791         use DPCM3
2792     else
2793         use PCM
2794     endif
2795

```

2796 **E.3.1.1 DPCM1 for 10–8–10 Decoder**

2797 **Xenco( n )** has the following format:

2798 **Xenco( n )** = “00 s xxxxx”

2799 where,

2800 “00” is the code word

2801 “s” is the **sign** bit

2802 “xxxxx” is the five bit **value** field

2803 The decoder equation is described as follows:

2804 **sign** = **Xenco( n )** & 0x20

2805 **value** = **Xenco( n )** & 0x1f

2806 if (**sign** > 0) then

2807 **Xdeco( n )** = **Xpred( n )** - **value**

2808 else

2809 **Xdeco( n )** = **Xpred( n )** + **value**

2810 endif

2811 **E.3.1.2 DPCM2 for 10–8–10 Decoder**

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

2813 **Xenco( n )** = “010 s xxxx”

2814 where,

2815 “010” is the code word

2816 “s” is the **sign** bit

2817 “xxxx” is the four bit **value** field

2818 The decoder equation is described as follows:

2819 **sign** = **Xenco( n )** & 0x10

2820 **value** = 2 \* (**Xenco( n )** & 0xf) + 32

2821 if (**sign** > 0) then

2822 **Xdeco( n )** = **Xpred( n )** - **value**

2823 else

2824 **Xdeco( n )** = **Xpred( n )** + **value**

2825 endif

2826 **E.3.1.3 DPCM3 for 10–8–10 Decoder**

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

2828 **Xenco( n )** = “011 s xxxx”

2829           where,

2830           “011” is the code word

2831           “s” is the **sign** bit

2832           “xxxx” is the four bit **value** field

2833   The decoder equation is described as follows:

```

2834       sign = Xenco( n ) & 0x10
2835       value = 4 * (Xenco( n ) & 0xf) + 64 + 1
2836       if (sign > 0) then
2837           Xdeco( n ) = Xpred( n ) - value
2838           if (Xdeco( n ) < 0) then
2839               Xdeco( n ) = 0
2840           endif
2841       else
2842           Xdeco( n ) = Xpred( n ) + value
2843           if (Xdeco( n ) > 1023) then
2844               Xdeco( n ) = 1023
2845           endif
2846       endif

```

#### 2847   **E.3.1.4       PCM for 10–8–10 Decoder**

2848   **Xenco**( **n** ) has the following format:

2849       **Xenco**( **n** ) = “1 xxxxxxxx”

2850           where,

2851           “1” is the code word

2852           the **sign** bit is not used

2853           “xxxxxxx” is the seven bit **value** field

2854   The codec equation is described as follows:

```

2855       value = 8 * (Xenco( n ) & 0x7f)
2856       if (value > Xpred( n )) then
2857           Xdeco( n ) = value + 3
2858       endif
2859       else
2860           Xdeco( n ) = value + 4
2861       endif

```

#### 2862   **E.3.2       Decoder for 10–7–10 Data Compression**

2863   Pixels without prediction are decoded using the following formula:

2864       **Xdeco**( **n** ) = 8 \* **Xenco**( **n** ) + 4

2865 Pixels with prediction are decoded using the following formula:

```

2866     if (Xenco( n ) & 0x70 == 0x00) then
2867         use DPCM1
2868     else if (Xenco( n ) & 0x78 == 0x10) then
2869         use DPCM2
2870     else if (Xenco( n ) & 0x78 == 0x18) then
2871         use DPCM3
2872     else if (Xenco( n ) & 0x60 == 0x20) then
2873         use DPCM4
2874     else
2875         use PCM
2876     endif

```

#### 2877 **E.3.2.1 DPCM1 for 10–7–10 Decoder**

2878 **Xenco**( n ) has the following format:

2879 **Xenco**( n ) = “000 s xxx”

2880 where,

2881 “000” is the code word  
2882 “s” is the **sign** bit  
2883 “xxx” is the three bit **value** field

2884 The codec equation is described as follows:

```

2885     sign = Xenco( n ) & 0x8
2886     value = Xenco( n ) & 0x7
2887     if (sign > 0) then
2888         Xdeco( n ) = Xpred( n ) - value
2889     else
2890         Xdeco( n ) = Xpred( n ) + value
2891     endif

```

#### 2892 **E.3.2.2 DPCM2 for 10–7–10 Decoder**

2893 **Xenco**( n ) has the following format:

2894 **Xenco**( n ) = “0010 s xx”

2895 where,

2896 “0010” is the code word  
2897 “s” is the **sign** bit  
2898 “xx” is the two bit **value** field

2899 The codec equation is described as follows:

```

2900     sign = Xenco( n ) & 0x4
2901     value = 2 * (Xenco( n ) & 0x3) + 8

```

```

2902         if (sign > 0) then
2903             Xdeco( n ) = Xpred( n ) - value
2904         else
2905             Xdeco( n ) = Xpred( n ) + value
2906         endif

```

### 2907 **E.3.2.3 DPCM3 for 10–7–10 Decoder**

2908 **Xenco**( **n** ) has the following format:

2909 **Xenco**( **n** ) = “0011 s xx”

2910 where,

2911 “0011” is the code word  
 2912 “s” is the **sign** bit  
 2913 “xx” is the two bit **value** field

2914 The codec equation is described as follows:

```

2915     sign = Xenco( n ) & 0x4
2916     value = 4 * (Xenco( n ) & 0x3) + 16 + 1
2917     if (sign > 0) then
2918         Xdeco( n ) = Xpred( n ) - value
2919         if (Xdeco( n ) < 0) then
2920             Xdeco( n ) = 0
2921         endif
2922     else
2923         Xdeco( n ) = Xpred( n ) + value
2924         if (Xdeco( n ) > 1023) then
2925             Xdeco( n ) = 1023
2926         endif
2927     endif

```

### 2928 **E.3.2.4 DPCM4 for 10–7–10 Decoder**

2929 **Xenco**( **n** ) has the following format:

2930 **Xenco**( **n** ) = “01 s xxxx”

2931 where,

2932 “01” is the code word  
 2933 “s” is the **sign** bit  
 2934 “xxxx” is the four bit **value** field

2935 The codec equation is described as follows:

```

2936     sign = Xenco( n ) & 0x10
2937     value = 8 * (Xenco( n ) & 0xf) + 32 + 3

```



```

2938     if (sign > 0) then
2939         Xdeco( n ) = Xpred( n ) - value
2940         if (Xdeco( n ) < 0) then
2941             Xdeco( n ) = 0
2942         endif
2943     else
2944         Xdeco( n ) = Xpred( n ) + value
2945         if (Xdeco( n ) > 1023) then
2946             Xdeco( n ) = 1023
2947         endif
2948     endif

```

#### 2949 **E.3.2.5 PCM for 10–7–10 Decoder**

2950 **Xenco( n )** has the following format:

2951 **Xenco( n )** = “1 xxxxxx”

2952 where,

2953 “1” is the code word  
 2954 the **sign** bit is not used  
 2955 “xxxxxx” is the six bit **value** field

2956 The codec equation is described as follows:

```

2957     value = 16 * (Xenco( n ) & 0x3f)
2958     if (value > Xpred( n )) then
2959         Xdeco( n ) = value + 7
2960     else
2961         Xdeco( n ) = value + 8
2962     endif

```

#### 2963 **E.3.3 Decoder for 10–6–10 Data Compression**

2964 Pixels without prediction are decoded using the following formula:

2965 **Xdeco( n ) = 16 \* Xenco( n ) + 8**

2966 Pixels with prediction are decoded using the following formula:

```

2967     if (Xenco( n ) & 0x3e == 0x00) then
2968         use DPCM1
2969     else if (Xenco( n ) & 0x3e == 0x02) then
2970         use DPCM2
2971     else if (Xenco( n ) & 0x3c == 0x04) then
2972         use DPCM3
2973     else if (Xenco( n ) & 0x38 == 0x08) then
2974         use DPCM4
2975     else if (Xenco( n ) & 0x30 == 0x10) then
2976         use DPCM5
2977     else
2978         use PCM
2979     endif

```

2980 **E.3.3.1 DPCM1 for 10–6–10 Decoder**

2981 **Xenco( n )** has the following format:

2982 **Xenco( n )** = “00000 s”

2983 where,

2984 “00000” is the code word

2985 “s” is the **sign** bit

2986 the **value** field is not used

2987 The codec equation is described as follows:

2988 **Xdeco( n ) = Xpred( n )**

2989 **E.3.3.2 DPCM2 for 10–6–10 Decoder**

2990 **Xenco( n )** has the following format:

2991 **Xenco( n )** = “00001 s”

2992 where,

2993 “00001” is the code word

2994 “s” is the **sign** bit

2995 the **value** field is not used

2996 The codec equation is described as follows:

2997 **sign = Xenco( n ) & 0x1**

2998 **value = 1**

2999 if (**sign** > 0) then

3000 **Xdeco( n ) = Xpred( n ) - value**

3001 else

3002 **Xdeco( n ) = Xpred( n ) + value**

3003 endif

3004 **E.3.3.3 DPCM3 for 10–6–10 Decoder**

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

3006 **Xenco( n )** = “0001 s x”

3007 where,

3008 “0001” is the code word

3009 “s” is the **sign** bit

3010 “x” is the one bit **value** field

3011 The codec equation is described as follows:

3012 **sign = Xenco( n ) & 0x2**

3013 **value = 4 \* (Xenco( n ) & 0x1) + 3 + 1**

```

3014         if (sign > 0) then
3015             Xdeco( n ) = Xpred( n ) - value
3016             if (Xdeco( n ) < 0) then
3017                 Xdeco( n ) = 0
3018             endif
3019         else
3020             Xdeco( n ) = Xpred( n ) + value
3021             if (Xdeco( n ) > 1023) then
3022                 Xdeco( n ) = 1023
3023             endif
3024         endif

```

#### 3025 **E.3.3.4 DPCM4 for 10–6–10 Decoder**

3026 **Xenco( n )** has the following format:

3027 **Xenco( n )** = “001 s xx”

3028 where,

3029 “001” is the code word  
3030 “s” is the **sign** bit  
3031 “xx” is the two bit **value** field

3032 The codec equation is described as follows:

```

3033     sign = Xenco( n ) & 0x4
3034     value = 8 * (Xenco( n ) & 0x3) + 11 + 3
3035     if (sign > 0) then
3036         Xdeco( n ) = Xpred( n ) - value
3037         if (Xdeco( n ) < 0) then
3038             Xdeco( n ) = 0
3039         endif
3040     else
3041         Xdeco( n ) = Xpred( n ) + value
3042         if (Xdeco( n ) > 1023) then
3043             Xdeco( n ) = 1023
3044         endif
3045     endif

```

#### 3046 **E.3.3.5 DPCM5 for 10–6–10 Decoder**

3047 **Xenco( n )** has the following format:

3048 **Xenco( n )** = “01 s xxx”

3049 where,

3050 “01” is the code word  
3051 “s” is the **sign** bit  
3052 “xxx” is the three bit **value** field

3053 The codec equation is described as follows:

```

3054     sign = Xenco( n ) & 0x8
3055     value = 16 * (Xenco( n ) & 0x7) + 43 + 7
3056     if (sign > 0) then
3057         Xdeco( n ) = Xpred( n ) - value
3058         if (Xdeco( n ) < 0) then
3059             Xdeco( n ) = 0
3060         endif
3061     else
3062         Xdeco( n ) = Xpred( n ) + value
3063         if (Xdeco( n ) > 1023) then
3064             Xdeco( n ) = 1023
3065         endif
3066     endif

```

#### 3067 **E.3.3.6 PCM for 10–6–10 Decoder**

3068 **Xenco**( **n** ) has the following format:

3069 **Xenco**( **n** ) = “1 xxxxx”

3070 where,

3071 “1” is the code word  
3072 the **sign** bit is not used  
3073 “xxxxx” is the five bit **value** field

3074 The codec equation is described as follows:

```

3075     value = 32 * (Xenco( n ) & 0x1f)
3076     if (value > Xpred( n )) then
3077         Xdeco( n ) = value + 15
3078     else
3079         Xdeco( n ) = value + 16
3080     endif

```

#### 3081 **E.3.4 Decoder for 12–8–12 Data Compression**

3082 Pixels without prediction are decoded using the following formula:

3083 **Xdeco**( **n** ) = 16 \* **Xenco**( **n** ) + 8

3084 Pixels with prediction are decoded using the following formula:

```

3085     if (Xenco( n ) & 0xf0 == 0x00) then
3086         use DPCM1
3087     else if (Xenco( n ) & 0xe0 == 0x60) then
3088         use DPCM2
3089     else if (Xenco( n ) & 0xe0 == 0x40) then
3090         use DPCM3
3091     else if (Xenco( n ) & 0xe0 == 0x20) then
3092         use DPCM4
3093     else if (Xenco( n ) & 0xf0 == 0x10) then

```

```

3094         use DPCM5
3095     else
3096         use PCM
3097     endif

```

#### 3098 **E.3.4.1 DPCM1 for 12–8–12 Decoder**

3099 **Xenco( n )** has the following format:

3100 **Xenco( n )** = “0000 s xxx”

3101 where,

3102 “0000” is the code word  
3103 “s” is the **sign** bit  
3104 “xxx” is the three bit **value** field

3105 The codec equation is described as follows:

```

3106     sign = Xenco( n ) & 0x8
3107     value = Xenco( n ) & 0x7
3108     if (sign > 0) then
3109         Xdeco( n ) = Xpred( n ) - value
3110     else
3111         Xdeco( n ) = Xpred( n ) + value
3112     endif

```

#### 3113 **E.3.4.2 DPCM2 for 12–8–12 Decoder**

3114 **Xenco( n )** has the following format:

3115 **Xenco( n )** = “011 s xxxx”

3116 where,

3117 “011” is the code word  
3118 “s” is the **sign** bit  
3119 “xxxx” is the four bit **value** field

3120 The codec equation is described as follows:

```

3121     sign = Xenco( n ) & 0x10
3122     value = 2 * (Xenco( n ) & 0xf) + 8
3123     if (sign > 0) then
3124         Xdeco( n ) = Xpred( n ) - value
3125     else
3126         Xdeco( n ) = Xpred( n ) + value
3127     endif

```

3128 **E.3.4.3 DPCM3 for 12–8–12 Decoder**3129 **Xenco( n )** has the following format:3130 **Xenco( n )** = “010 s xxxx”

3131 where,

3132 “010” is the code word

3133 “s” is the **sign** bit3134 “xxxx” is the four bit **value** field

3135 The codec equation is described as follows:

```

3136 sign = Xenco( n ) & 0x10
3137 value = 4 * (Xenco( n ) & 0xf) + 40 + 1
3138 if (sign > 0) then
3139     Xdeco( n ) = Xpred( n ) - value
3140     if (Xdeco( n ) < 0) then
3141         Xdeco( n ) = 0
3142     endif
3143 else
3144     Xdeco( n ) = Xpred( n ) + value
3145     if (Xdeco( n ) > 4095) then
3146         Xdeco( n ) = 4095
3147     endif
3148 endif

```

3149 **E.3.4.4 DPCM4 for 12–8–12 Decoder**3150 **Xenco( n )** has the following format:3151 **Xenco( n )** = “001 s xxxx”

3152 where,

3153 “001” is the code word

3154 “s” is the **sign** bit3155 “xxxx” is the four bit **value** field

3156 The codec equation is described as follows:

```

3157 sign = Xenco( n ) & 0x10
3158 value = 8 * (Xenco( n ) & 0xf) + 104 + 3
3159 if (sign > 0) then
3160     Xdeco( n ) = Xpred( n ) - value
3161     if (Xdeco( n ) < 0) then
3162         Xdeco( n ) = 0
3163     endif
3164 else
3165     Xdeco( n ) = Xpred( n ) + value
3166     if (Xdeco( n ) > 4095)
3167         Xdeco( n ) = 4095
3168     endif
3169 endif

```

### 3170 **E.3.4.5 DPCM5 for 12–8–12 Decoder**

3171 **Xenco( n )** has the following format:

3172 **Xenco( n )** = “0001 s xxx”

3173 where,

3174 “0001” is the code word

3175 “s” is the **sign** bit

3176 “xxx” is the three bit **value** field

3177 The codec equation is described as follows:

```

3178 sign = Xenco( n ) & 0x8
3179 value = 16 * (Xenco( n ) & 0x7) + 232 + 7
3180 if (sign > 0) then
3181     Xdeco( n ) = Xpred( n ) - value
3182     if (Xdeco( n ) < 0) then
3183         Xdeco( n ) = 0
3184     endif
3185 else
3186     Xdeco( n ) = Xpred( n ) + value
3187     if (Xdeco( n ) > 4095) then
3188         Xdeco( n ) = 4095
3189     endif
3190 endif

```

### 3191 **E.3.4.6 PCM for 12–8–12 Decoder**

3192 **Xenco( n )** has the following format:

3193 **Xenco( n )** = “1 xxxxxxx”

3194 where,

3195 “1” is the code word

3196 the **sign** bit is not used

3197 “xxxxxxx” is the seven bit **value** field

3198 The codec equation is described as follows:

```

3199 value = 32 * (Xenco( n ) & 0x7f)
3200 if (value > Xpred( n )) then
3201     Xdeco( n ) = value + 15
3202 else
3203     Xdeco( n ) = value + 16
3204 endif

```

### 3205 **E.3.5 Decoder for 12–7–12 Data Compression**

3206 Pixels without prediction are decoded using the following formula:

3207 **Xdeco( n )** = 32 \* **Xenco( n )** + 16

3208 Pixels with prediction are decoded using the following formula:

```

3209     if (Xenco( n ) & 0x78 == 0x00) then
3210         use DPCM1
3211     else if (Xenco( n ) & 0x78 == 0x08) then
3212         use DPCM2
3213     else if (Xenco( n ) & 0x78 == 0x10) then
3214         use DPCM3
3215     else if (Xenco( n ) & 0x70 == 0x20) then
3216         use DPCM4
3217     else if (Xenco( n ) & 0x70 == 0x30) then
3218         use DPCM5
3219     else if (Xenco( n ) & 0x78 == 0x18) then
3220         use DPCM6
3221     else
3222         use PCM
3223     endif

```

#### 3224 **E.3.5.1 DPCM1 for 12–7–12 Decoder**

3225 **Xenco**( n ) has the following format:

3226 **Xenco**( n ) = “0000 s xx”

3227 where,

3228 “0000” is the code word  
3229 “s” is the **sign** bit  
3230 “xx” is the two bit **value** field

3231 The codec equation is described as follows:

```

3232     sign = Xenco( n ) & 0x4
3233     value = Xenco( n ) & 0x3
3234     if (sign > 0) then
3235         Xdeco( n ) = Xpred( n ) - value
3236     else
3237         Xdeco( n ) = Xpred( n ) + value
3238     endif

```

#### 3239 **E.3.5.2 DPCM2 for 12–7–12 Decoder**

3240 **Xenco**( n ) has the following format:

3241 **Xenco**( n ) = “0001 s xx”

3242 where,

3243 “0001” is the code word  
3244 “s” is the **sign** bit  
3245 “xx” is the two bit **value** field



3246 The codec equation is described as follows:

```

3247     sign = Xenco( n ) & 0x4
3248     value = 2 * (Xenco( n ) & 0x3) + 4
3249     if (sign > 0) then
3250         Xdeco( n ) = Xpred( n ) - value
3251     else
3252         Xdeco( n ) = Xpred( n ) + value
3253     endif

```

### 3254 E.3.5.3 DPCM3 for 12–7–12 Decoder

3255 **Xenco**( **n** ) has the following format:

3256 **Xenco**( **n** ) = “0010 s xx”

3257 where,

3258 “0010” is the code word  
3259 “s” is the **sign** bit  
3260 “xx” is the two bit **value** field

3261 The codec equation is described as follows:

```

3262     sign = Xenco( n ) & 0x4
3263     value = 4 * (Xenco( n ) & 0x3) + 12 + 1
3264     if (sign > 0) then
3265         Xdeco( n ) = Xpred( n ) - value
3266         if (Xdeco( n ) < 0) then
3267             Xdeco( n ) = 0
3268         endif
3269     else
3270         Xdeco( n ) = Xpred( n ) + value
3271         if (Xdeco( n ) > 4095) then
3272             Xdeco( n ) = 4095
3273         endif
3274     endif

```

### 3275 E.3.5.4 DPCM4 for 12–7–12 Decoder

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

3277 **Xenco**( **n** ) = “010 s xxx”

3278 where,

3279 “010” is the code word  
3280 “s” is the **sign** bit  
3281 “xxx” is the three bit **value** field

3282 The codec equation is described as follows:

```

3283     sign = Xenco( n ) & 0x8
3284     value = 8 * (Xenco( n ) & 0x7) + 28 + 3

```

```

3285     if (sign > 0) then
3286         Xdeco( n ) = Xpred( n ) - value
3287         if (Xdeco( n ) < 0) then
3288             Xdeco( n ) = 0
3289         endif
3290     else
3291         Xdeco( n ) = Xpred( n ) + value
3292         if (Xdeco( n ) > 4095) then
3293             Xdeco( n ) = 4095
3294         endif
3295     endif

```

#### 3296 **E.3.5.5 DPCM5 for 12–7–12 Decoder**

3297 **Xenco( n )** has the following format:

3298 **Xenco( n )** = “011 s xxx”

3299 where,

3300 “011” is the code word  
3301 “s” is the **sign** bit  
3302 “xxx” is the three bit **value** field

3303 The codec equation is described as follows:

```

3304     sign = Xenco( n ) & 0x8
3305     value = 16 * (Xenco( n ) & 0x7) + 92 + 7
3306     if (sign > 0) then
3307         Xdeco( n ) = Xpred( n ) - value
3308         if (Xdeco( n ) < 0) then
3309             Xdeco( n ) = 0
3310         endif
3311     else
3312         Xdeco( n ) = Xpred( n ) + value
3313         if (Xdeco( n ) > 4095) then
3314             Xdeco( n ) = 4095
3315         endif
3316     endif

```

#### 3317 **E.3.5.6 DPCM6 for 12–7–12 Decoder**

3318 **Xenco( n )** has the following format:

3319 **Xenco( n )** = “0011 s xx”

3320 where,

3321 “0011” is the code word  
3322 “s” is the **sign** bit  
3323 “xx” is the two bit **value** field

3324 The codec equation is described as follows:

```

3325     sign = Xenco( n ) & 0x4
3326     value = 32 * (Xenco( n ) & 0x3) + 220 + 15
3327     if (sign > 0) then
3328         Xdeco( n ) = Xpred( n ) - value
3329         if (Xdeco( n ) < 0) then
3330             Xdeco( n ) = 0
3331         endif
3332     else
3333         Xdeco( n ) = Xpred( n ) + value
3334         if (Xdeco( n ) > 4095) then
3335             Xdeco( n ) = 4095
3336         endif
3337     endif

```

### 3338 E.3.5.7 PCM for 12–7–12 Decoder

3339 **Xenco**( **n** ) has the following format:

3340 **Xenco**( **n** ) = “1 xxxxxx”

3341 where,

3342 “1” is the code word  
 3343 the **sign** bit is not used  
 3344 “xxxxxx” is the six bit **value** field

3345 The codec equation is described as follows:

```

3346     value = 64 * (Xenco( n ) & 0x3f)
3347     if (value > Xpred( n )) then
3348         Xdeco( n ) = value + 31
3349     else
3350         Xdeco( n ) = value + 32
3351     endif

```

### 3352 E.3.6 Decoder for 12–6–12 Data Compression

3353 Pixels without prediction are decoded using the following formula:

3354 **Xdeco**( **n** ) = 64 \* **Xenco**( **n** ) + 32

3355 Pixels with prediction are decoded using the following formula:

```

3356     if (Xenco( n ) & 0x3c == 0x00) then
3357         use DPCM1
3358     else if (Xenco( n ) & 0x3c == 0x04) then
3359         use DPCM3
3360     else if (Xenco( n ) & 0x38 == 0x10) then
3361         use DPCM4
3362     else if (Xenco( n ) & 0x3c == 0x08) then
3363         use DPCM5
3364     else if (Xenco( n ) & 0x38 == 0x18) then

```

```

3365         use DPCM6
3366     else if (Xenco( n ) & 0x3c == 0x0c) then
3367         use DPCM7
3368     else
3369         use PCM
3370     endif

```

3371 Note: **DPCM2** is not used.

### 3372 **E.3.6.1 DPCM1 for 12–6–12 Decoder**

3373 **Xenco**( n ) has the following format:

3374 **Xenco**( n ) = “0000 s x”

3375 where,

3376 “0000” is the code word  
3377 “s” is the **sign** bit  
3378 “x” is the one bit **value** field

3379 The codec equation is described as follows:

```

3380     sign = Xenco( n ) & 0x2
3381     value = Xenco( n ) & 0x1
3382     if (sign > 0) then
3383         Xdeco( n ) = Xpred( n ) - value
3384     else
3385         Xdeco( n ) = Xpred( n ) + value
3386     endif

```

### 3387 **E.3.6.2 DPCM3 for 12–6–12 Decoder**

3388 **Xenco**( n ) has the following format:

3389 **Xenco**( n ) = “0001 s x”

3390 where,

3391 “0001” is the code word  
3392 “s” is the **sign** bit  
3393 “x” is the one bit **value** field

3394 The codec equation is described as follows:

```

3395     sign = Xenco( n ) & 0x2
3396     value = 4 * (Xenco( n ) & 0x1) + 2 + 1
3397     if (sign > 0) then
3398         Xdeco( n ) = Xpred( n ) - value
3399     if (Xdeco( n ) < 0) then
3400         Xdeco( n ) = 0
3401     endif
3402     else
3403         Xdeco( n ) = Xpred( n ) + value

```

```

3404         if (Xdeco( n ) > 4095) then
3405             Xdeco( n ) = 4095
3406         endif
3407     endif

```

### 3408 E.3.6.3 DPCM4 for 12–6–12 Decoder

3409 **Xenco( n )** has the following format:

3410 **Xenco( n )** = “010 s xx”

3411 where,

```

3412         “010” is the code word
3413         “s” is the sign bit
3414         “xx” is the two bit value field

```

3415 The codec equation is described as follows:

```

3416     sign = Xenco( n ) & 0x4
3417     value = 8 * (Xenco( n ) & 0x3) + 10 + 3
3418     if (sign > 0) then
3419         Xdeco( n ) = Xpred( n ) - value
3420         if (Xdeco( n ) < 0) then
3421             Xdeco( n ) = 0
3422         endif
3423     else
3424         Xdeco( n ) = Xpred( n ) + value
3425         if (Xdeco( n ) > 4095) then
3426             Xdeco( n ) = 4095
3427         endif
3428     endif

```

### 3429 E.3.6.4 DPCM5 for 12–6–12 Decoder

3430 **Xenco( n )** has the following format:

3431 **Xenco( n )** = “0010 s x”

3432 where,

```

3433         “0010” is the code word
3434         “s” is the sign bit
3435         “x” is the one bit value field

```

3436 The codec equation is described as follows:

```

3437     sign = Xenco( n ) & 0x2
3438     value = 16 * (Xenco( n ) & 0x1) + 42 + 7
3439     if (sign > 0) then
3440         Xdeco( n ) = Xpred( n ) - value
3441         if (Xdeco( n ) < 0) then
3442             Xdeco( n ) = 0
3443         endif

```

```

3444         else
3445             Xdeco( n ) = Xpred( n ) + value
3446             if (Xdeco( n ) > 4095) then
3447                 Xdeco( n ) = 4095
3448             endif
3449         endif

```

### 3450 **E.3.6.5 DPCM6 for 12–6–12 Decoder**

3451 **Xenco( n )** has the following format:

3452 **Xenco( n )** = “011 s xx”

3453 where,

3454 “011” is the code word  
 3455 “s” is the **sign** bit  
 3456 “xx” is the two bit **value** field

3457 The codec equation is described as follows:

```

3458     sign = Xenco( n ) & 0x4
3459     value = 32 * (Xenco( n ) & 0x3) + 74 + 15
3460     if (sign > 0) then
3461         Xdeco( n ) = Xpred( n ) - value
3462         if (Xdeco( n ) < 0) then
3463             Xdeco( n ) = 0
3464         endif
3465     else
3466         Xdeco( n ) = Xpred( n ) + value
3467         if (Xdeco( n ) > 4095) then
3468             Xdeco( n ) = 4095
3469         endif
3470     endif

```

### 3471 **E.3.6.6 DPCM7 for 12–6–12 Decoder**

3472 **Xenco( n )** has the following format:

3473 **Xenco( n )** = “0011 s x”

3474 where,

3475 “0011” is the code word  
 3476 “s” is the **sign** bit  
 3477 “x” is the one bit **value** field

3478 The codec equation is described as follows:

```

3479     sign = Xenco( n ) & 0x2
3480     value = 64 * (Xenco( n ) & 0x1) + 202 + 31
3481     if (sign > 0) then
3482         Xdeco( n ) = Xpred( n ) - value
3483         if (Xdeco( n ) < 0) then

```

```

3484         Xdeco( n ) = 0
3485     endif
3486 else
3487     Xdeco( n ) = Xpred( n ) + value
3488     if (Xdeco( n ) > 4095) then
3489         Xdeco( n ) = 4095
3490     endif
3491 endif

```

#### 3492 **E.3.6.7 PCM for 12–6–12 Decoder**

3493 **Xenco( n )** has the following format:

3494 **Xenco( n )** = “1 xxxxx”

3495 where,

3496 “1” is the code word  
 3497 the **sign** bit is not used  
 3498 “xxxxx” is the five bit **value** field

3499 The codec equation is described as follows:

```

3500     value = 128 * (Xenco( n ) & 0x1f)
3501     if (value > Xpred( n )) then
3502         Xdeco( n ) = value + 63
3503     else
3504         Xdeco( n ) = value + 64
3505     endif

```

3506

## Annex F JPEG Interleaving (informative)

This annex illustrates how the standard features of the CSI-2 protocol should be used to interleave (multiplex) JPEG image data with other types of image data, e.g. RGB565 or YUV422, without requiring a custom JPEG format such as JPEG8.

The Virtual Channel Identifier and Data Type value in the CSI-2 Packet Header provide simple methods of interleaving multiple data streams or image data types at the packet level. Interleaving at the packet level minimizes the amount of buffering required in the system.

The Data Type value in the CSI-2 Packet Header should be used to multiplex different image data types at the CSI-2 transmitter and de-multiplex the data types at the CSI-2 receiver.

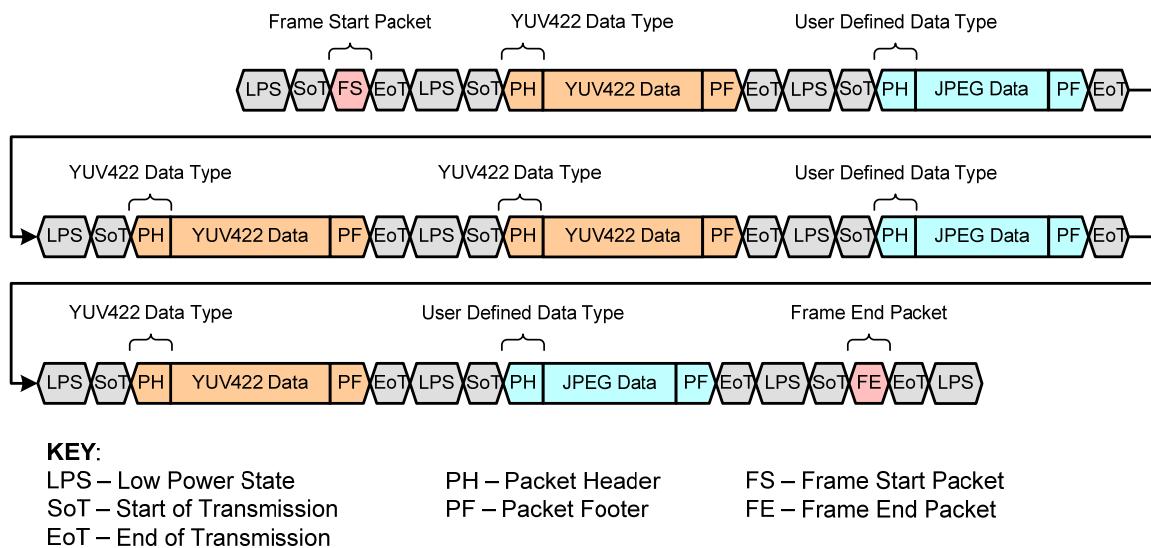
The Virtual Channel Identifier in the CSI-2 Packet Header should be used to multiplex different data streams (channels) at the CSI-2 transmitter and de-multiplex the streams at the CSI-2 receiver.

The main difference between the two interleaving methods is that images with different Data Type values within the same Virtual Channel use the same frame and line synchronization information, whereas multiple Virtual Channels (data streams) each have their own independent frame and line synchronization information and thus potentially each channel may have different frame rates.

Since the predefined Data Type values represent only YUV, RGB and RAW data types, one of the User Defined Data Type values should be used to represent JPEG image data.

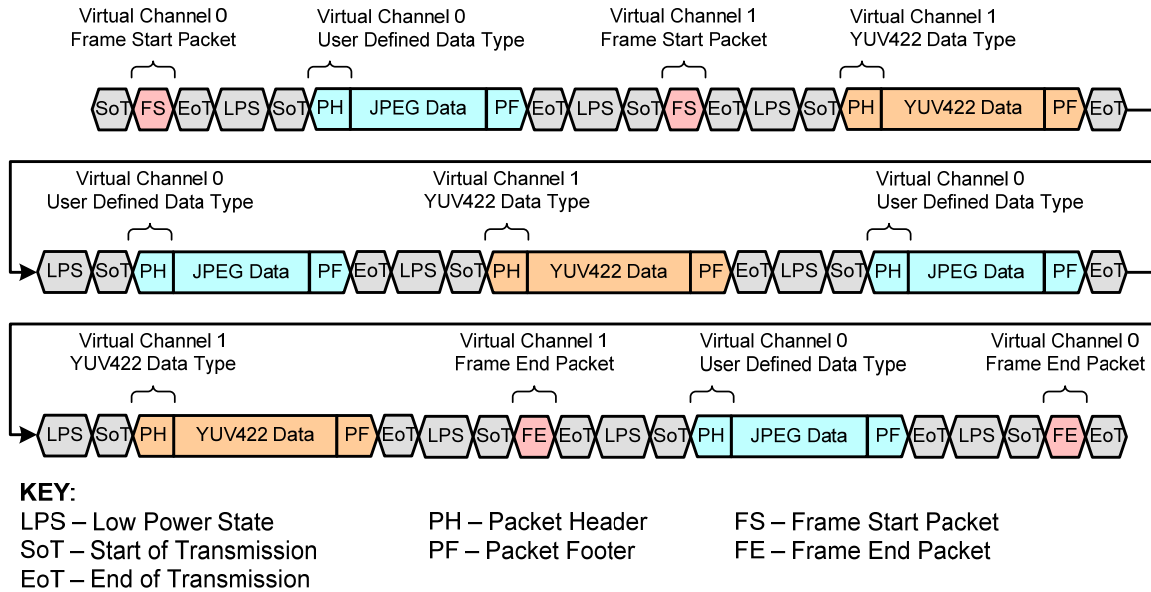
Figure 144 illustrates interleaving JPEG image data with YUV422 image data using Data Type values.

Figure 145 illustrates interleaving JPEG image data with YUV422 image data using both Data Type values and Virtual Channel Identifiers.



**Figure 144 Data Type Interleaving: Concurrent JPEG and YUV Image Data**





**Figure 145 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data**

Both Figure 144 and Figure 145 can be similarly extended to the interleaving of JPEG image data with any other type of image data, e.g. RGB565.

Figure 146 illustrates the use of Virtual Channels to support three different JPEG interleaving usage cases:

- Concurrent JPEG and YUV422 image data.
- Alternating JPEG and YUV422 output - one frame JPEG, then one frame YUV
- Streaming YUV22 with occasional JPEG for still capture

Again, these examples could also represent interleaving JPEG data with any other image data type.

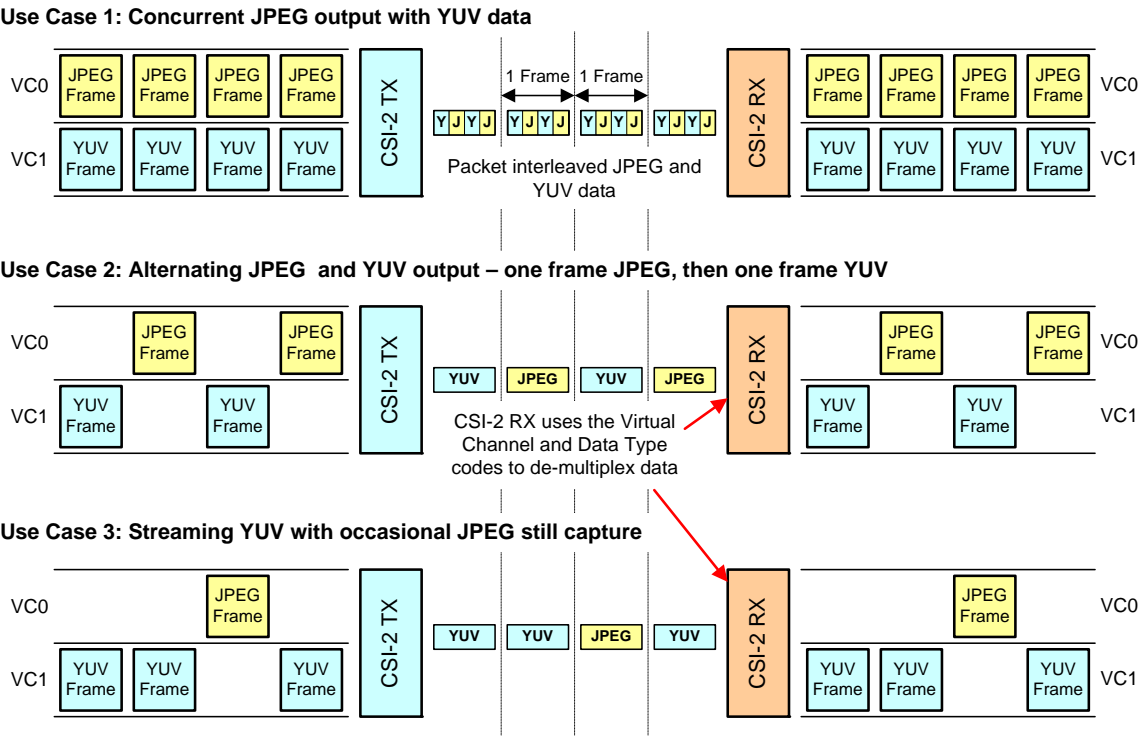


Figure 146 Example JPEG and YUV Interleaving Use Cases