

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

Version 1.2

22 January 2014

MIPI Board Adopted 10 September 2014

This document is subject to further editorial and technical development.

#### NOTICE OF DISCLAIMER

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

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

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

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

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

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

# Contents

3	Contents	iii
4	Tables	xii
5	Release History	xiii
6	5 1 Overview	1
7	1.1 Scope	1
8	1.2 Purpose	1
9	2 Terminology	2
10	2.1 Definitions	2
11	2.2 Abbreviations	2
12	2.3 Acronyms	3
13	3 References	5
14	4 Overview of CSI-2	6
15	5 CSI-2 Layer Definitions	7
16	6 Camera Control Interface (CCI)	9
17	6.1 Data Transfer Protocol	9
18	6.1.1 Message Type	9
19	6.1.2 Read/Write Operations	10
20	6.2 CCI Slave Addresses	13
21	6.3 CCI Multi-Byte Registers	13
22	6.3.1 Overview	13
23	6.3.2 The Transmission Byte Order for Multi-byte Regis	ster Values15
24	6.3.3 Multi-Byte Register Protocol	16
25	6.4 Electrical Specifications and Timing for I/O Stages	20
26	7 Physical Layer	23
27	8 Multi-Lane Distribution and Merging	24
28	8.1 Multi-Lane Interoperability	29
29	9 Low Level Protocol	32
30	9.1 Low Level Protocol Packet Format	32
31	9.1.1 Low Level Protocol Long Packet Format	32
32	9.1.2 Low Level Protocol Short Packet Format	34
33	9.2 Data Identifier (DI)	34
34	9.3 Virtual Channel Identifier	34
35	9.4 Data Type (DT)	35
36	9.5 Packet Header Error Correction Code	36
37	9.5.1 General Hamming Code Applied to Packet Header	r 37

	Version 22 Jan-2		e Specification for CSI-2
38	9.5.	.2 Hamming-Modified Code	37
39	9.5.	.3 ECC Generation on TX Side	40
40	9.5.	.4 Applying ECC on RX Side	41
41	9.6	Checksum Generation	42
42	9.7	Packet Spacing	44
43	9.8	Synchronization Short Packet Data Type Codes	45
44	9.8.	.1 Frame Synchronization Packets	45
45	9.8.	.2 Line Synchronization Packets	46
46	9.9	Generic Short Packet Data Type Codes	46
47	9.10	Packet Spacing Examples	47
48	9.11	Packet Data Payload Size Rules	49
49	9.12	Frame Format Examples	50
50	9.13	Data Interleaving	52
51	9.13	3.1 Data Type Interleaving	52
52	9.13	3.2 Virtual Channel Identifier Interleaving	55
53	10 C	Color Spaces	57
54	10.1	RGB Color Space Definition	57
55	10.2	YUV Color Space Definition	57
56	11 D	Data Formats	58
57	11.1	Generic 8-bit Long Packet Data Types	59
58	11.	1.1 Null and Blanking Data	59
59	11.	1.2 Embedded Information	59
60	11.2	YUV Image Data	60
61	11.2	2.1 Legacy YUV420 8-bit	61
62	11.2	2.2 YUV420 8-bit	63
63	11.2	2.3 YUV420 10-bit	65
64	11.2	2.4 YUV422 8-bit	67
65	11.2	2.5 YUV422 10-bit	69
66	11.3	RGB Image Data	70
67	11.3	3.1 RGB888	71
68	11.3	3.2 RGB666	72
69	11.3	3.3 RGB565	73
70	11.3	3.4 RGB555	74
71	11.3	3.5 RGB444	75
72	11.4	RAW Image Data	75

RAW7.......77

73

74

11.4.1

11.4.2

	Version 1 22 Jan-20		Alliance Specification for CSI-2
75	11.4	.3 RAW8	78
76	11.4	.4 RAW10	79
77	11.4	.5 RAW12	80
78	11.4	.6 RAW14	81
79	11.5	User Defined Data Formats	82
80	12 R	ecommended Memory Storage	84
81	12.1	General/Arbitrary Data Reception	84
82	12.2	RGB888 Data Reception	85
83	12.3	RGB666 Data Reception	85
84	12.4	RGB565 Data Reception	86
85	12.5	RGB555 Data Reception	86
86	12.6	RGB444 Data Reception	87
87	12.7	YUV422 8-bit Data Reception	87
88	12.8	YUV422 10-bit Data Reception	88
89	12.9	YUV420 8-bit (Legacy) Data Reception	88
90	12.10	YUV420 8-bit Data Reception	90
91	12.11	YUV420 10-bit Data Reception	91
92	12.12	RAW6 Data Reception	92
93	12.13	RAW7 Data Reception	92
94	12.14	RAW8 Data Reception	93
95	12.15	RAW10 Data Reception	93
96	12.16	RAW12 Data Reception	94
97	12.17	RAW14 Data Reception	94
98	Annex A	JPEG8 Data Format (informative)	95
99	A.1	Introduction	95
100	A.2	JPEG Data Definition	96
101	A.3	Image Status Information	97
102	A.4	Embedded Images	98
103	A.5	JPEG8 Non-standard Markers	99
104	A.6	JPEG8 Data Reception	100
105	Annex B	CSI-2 Implementation Example (informative)	101
106	B.1	Overview	101
107	B.2	CSI-2 Transmitter Detailed Block Diagram	101
108	B.3	CSI-2 Receiver Detailed Block Diagram	102

109

110

111

B.4

B.4.1

B.4.2

MIDI	Allianca	Specifica	ation	for	CCI 2
MHPI	Alliance	Specifica	ation	TOT	CSI-2

	Version 1.2 22 Jan-201		IIPI Alliance Specification for CSI-2
112	B.4.3	CSI-2 Data Lane Transmitter	107
113	B.4.4	CSI-2 Data Lane Receiver	108
114	Annex C	CSI-2 Recommended Receiver Error Behavior (informative	e)111
115	C.1 (	Overview	111
116	C.2 I	O-PHY Level Error	111
117	C.3 F	Packet Level Error	112
118	C.4 F	Protocol Decoding Level Error	113
119	Annex D	CSI-2 Sleep Mode (informative)	114
120	D.1 (	Overview	114
121	D.2 S	SLM Command Phase	114
122	D.3 S	SLM Entry Phase	114
123	D.4 S	SLM Exit Phase	115
124	Annex E	Data Compression for RAW Data Types (normative)	116
125	E.1 F	Predictors	117
126	E.1.1	Predictor1	118
127	E.1.2	Predictor2	118
128	E.2 E	Encoders	119
129	E.2.1	Coder for 10–8–10 Data Compression	119
130	E.2.2	Coder for 10–7–10 Data Compression	121
131	E.2.3	Coder for 10-6-10 Data Compression	123
132	E.2.4	Coder for 12–8–12 Data Compression	126
133	E.2.5	Coder for 12–7–12 Data Compression	129
134	E.2.6	Coder for 12–6–12 Data Compression	132
135	E.3 I	Decoders	135
136	E.3.1	Decoder for 10–8–10 Data Compression	136
137	E.3.2	Decoder for 10–7–10 Data Compression	138
138	E.3.3	Decoder for 10–6–10 Data Compression	140
139	E.3.4	Decoder for 12–8–12 Data Compression	143
140	E.3.5	Decoder for 12–7–12 Data Compression	146
141	E.3.6	Decoder for 12–6–12 Data Compression	150
142	Annex F	JPEG Interleaving (informative)	

# **Figures**

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

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

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

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

	Version 1.2 22 Jan-2014	MIPI Alliance Specification for CSI-2
290	Figure 147 Example JPEG and YUV Interleaving Use Cases	157

# **Tables**

291

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

# Release History

Date	Release	Description
2005-11-29	v1.00	Initial Board-approved release.
2010-11-09	v1.01.00	Board-approved release.
2013-01-22	v1.1	Board approved release.
2014-09-10	v1.2	Board approved release.

#### 1 Overview

#### 1.1 Scope

322

323

- 324 The Camera Serial Interface 2 Specification defines an interface between a peripheral device (camera) and a
- 325 host processor (baseband, application engine). The purpose of this document is to specify a standard interface
- between a camera and a host processor for mobile applications.
- 327 This Revision of the Camera Serial Interface 2 Specification leverages the improved skew tolerance and
- 328 higher data rate of the [MIPI01] D-PHY 1.2 Specification. These enhancements enable higher interface
- 329 bandwidth and more flexibility in channel layout.
- 330 A host processor in this document refers to the hardware and software that performs essential core functions
- 331 for telecommunication or application tasks. The engine of a mobile terminal includes hardware and the
- 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
- 334 software.

335

#### 1.2 Purpose

- 336 Demand for increasingly higher image resolutions is pushing the bandwidth capacity of existing host
- 337 processor-to-camera sensor interfaces. Common parallel interfaces are difficult to expand, require many
- 338 interconnects and consume relatively large amounts of power. Emerging serial interfaces address many of
- 339 the shortcomings of parallel interfaces while introducing their own problems. Incompatible, proprietary
- 340 interfaces prevent devices from different manufacturers from working together. This can raise system costs
- 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.
- 343 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 to*).
- The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).
- All sections are normative, unless they are explicitly indicated to be informative.

#### 364 **2.1 Definitions**

- 365 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.
- 369 **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.
- 371 **Sleep Mode:** Sleep mode (SLM) is a leakage level only power consumption mode.
- 372 **Transmission:** The time during which high-speed serial data is actively traversing the bus. A transmission
- 373 is comprised of one or more packets. A transmission is bounded by SoT (Start of Transmission) and EoT
- 374 (End of Transmission) at beginning and end, respectively.
- 375 Virtual Channel: Multiple independent data streams for up to four peripherals are supported by this
- 376 Specification. The data stream for each peripheral is a Virtual Channel. These data streams may be
- 377 interleaved and sent as sequential packets, with each packet dedicated to a particular peripheral or channel.
- Packet protocol includes information that links each packet to its intended peripheral.

#### 2.2 Abbreviations

379

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

Version	1.2
22 Jan-2	014

381	i.e.	That is (Latin: id est)
382	2.3 Ac	ronyms
383	BER	Bit Error Rate
384	CCI	Camera Control Interface
385	CIL	Control and Interface Logic
386	CRC	Cyclic Redundancy Check
387	CSI	Camera Serial Interface
388	CSPS	Chroma Sample Pixel Shifted
389	DDR	Dual Data Rate
390	DI	Data Identifier
391	DT	Data Type
392	ECC	Error Correction Code
393	ЕоТ	End of Transmission
394	EXIF	Exchangeable Image File Format
395	FE	Frame End
396	FS	Frame Start
397	HS	High Speed; identifier for operation mode
398	HS-RX	High-Speed Receiver (Low-Swing Differential)
399	HS-TX	High-Speed Transmitter (Low-Swing Differential)
400	I2C	Inter-Integrated Circuit
401	JFIF	JPEG File Interchange Format
402	JPEG	Joint Photographic Expert Group
403	LE	Line End
404	LLP	Low Level Protocol
405	LS	Line Start
406	LSB	Least Significant Bit

Low-Power; identifier for operation mode

407

LP

	Version 1.2 22 Jan-2014		MIPI Alliance Specification for CSI-2
408	LP-RX	Low-Power Receiver (Large-Swing Single Ended)	
409	LP-TX	Low-Power Transmitter (Large-Swing Single Ended)	
410	MSB	Most Significant Bit	
411	PF	Packet Footer	
412	PH	Packet Header	
413	PI	Packet Identifier	
414	PT	Packet Type	
415	PHY	Physical Layer	
416	PPI	PHY Protocol Interface	
417	RGB	Color representation (Red, Green, Blue)	
418	RX	Receiver	
419	SCL	Serial Clock (for CCI)	
420	SDA	Serial Data (for CCI)	
421	SLM	Sleep Mode	
422	SoT	Start of Transmission	
423	TX	Transmitter	
424	ULPS	Ultra-low Power State	
425	VGA	Video Graphics Array	

Color representation (Y for luminance, U & V for chrominance)

426

YUV

427	3 Referei	nces
428 429	[NXP01]	UM10204, <i>I2C-bus specification and user manual</i> , Revision 03, NXP B.V., 19 June 2007.
430 431	[MIPI01]	MIPI Alliance Specification for D-PHY, version 1.2, MIPI Alliance, Inc., 10 September 2014.

#### 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.

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

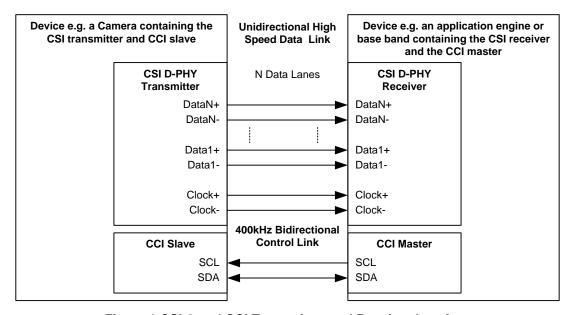


Figure 1 CSI-2 and CCI Transmitter and Receiver Interface

442

443

444

445

446 447

448

449

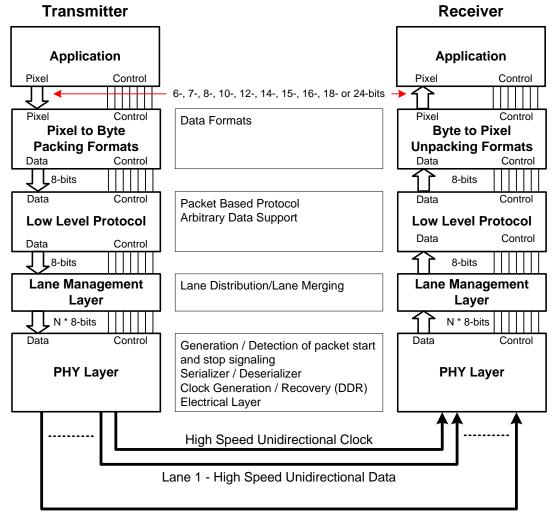
450 451

452

453

454

# 5 CSI-2 Layer Definitions



Lane N - High Speed Unidirectional Data

Figure 2 CSI-2 Layer Definitions

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

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

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

The PHY layer is described in [MIPI01].

- 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 bitlevel 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 chosen 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.

# 486 6 Camera Control Interface (CCI)

- 487 CCI is a two-wire, bi-directional, half duplex, serial interface for controlling the transmitter. CCI is
- 488 compatible with the fast mode variant of the I2C interface. CCI shall support 400kHz operation and 7-bit
- 489 Slave Addressing.
- 490 A CSI-2 receiver shall be configured as a master and a CSI-2 transmitter shall be configured as a slave on
- 491 the CCI bus. CCI is capable of handling multiple slaves on the bus. However, multi-master mode is not
- supported by CCI. Any I2C commands that are not described in this section shall be ignored and shall not
- 493 cause unintended device operation. Note that the terms master and slave, when referring to CCI, should not
- be confused with similar terminology used for D-PHY's operation; they are not related.
- 495 Typically, there is a dedicated CCI interface between the transmitter and the receiver.
- 496 CCI is a subset of the 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
- 498 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
- 505 control messages to the CCI slave.
- 506 The CCI defines an additional data protocol layer on top of I2C. The data protocol is presented in the
- following sections.

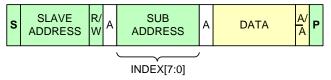
#### 508 6.1 Data Transfer Protocol

- 509 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^2C Specification* [NXP01].

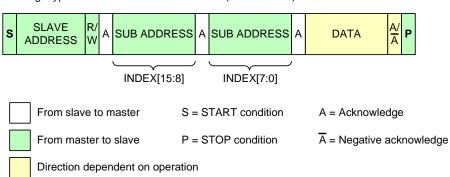
# **511 6.1.1 Message Type**

- 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,
- 514 in write operation data byte from master, acknowledge/negative acknowledge from slave and STOP
- 515 condition. In read operation data byte comes from slave and acknowledge/negative acknowledge from
- 516 master. This is illustrated in Figure 3.
- 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)



520521

522

523

524

525

529

530

531

532

533

Figure 3 CCI Message Types

### 6.1.2 Read/Write Operations

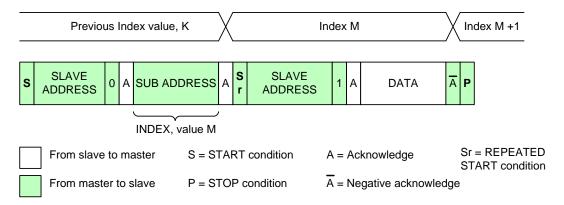
The CCI compatible device shall be able to support four different read operations and two different write operations; single read from random location, sequential read from random location, single read from current location, sequential read from current location, single write to random location and sequential 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.



534 535

Figure 4 CCI Single Read from Random Location

537

538539

540

541

542

543

544

545

546

547

548 549

550

551

#### 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 responses 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.

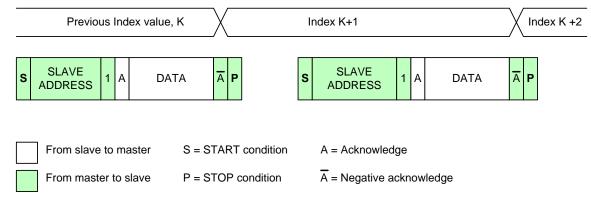


Figure 5 CCI Single Read from Current Location

# 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.

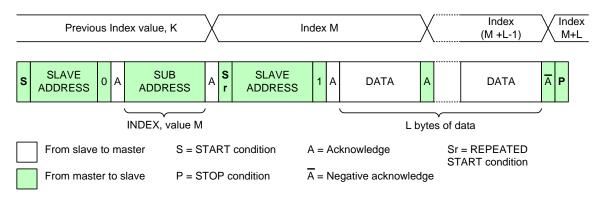
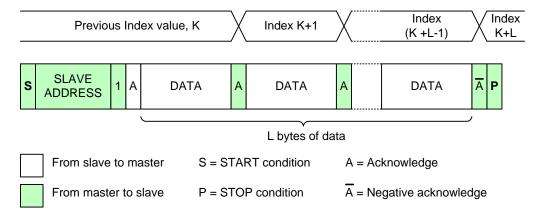


Figure 6 CCI Sequential Read Starting from a Random Location

### 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.



557

558

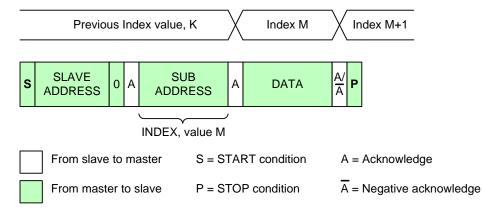
559

560

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.



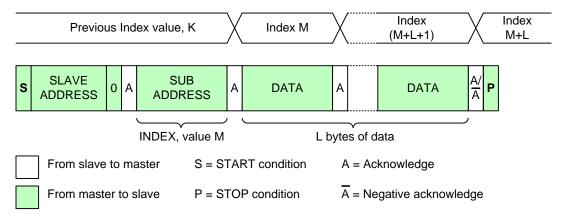
561562

563

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.



568

571

572

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-574 2 Specification supports the following register widths:
- 8-bit generic setup registers
- 16-bit parameters like line-length, frame-length and exposure values
- 32-bit high precision setup values
- 64-bit for needs of future sensors
- In general, the byte oriented access protocols described in the previous sections provide an efficient means
- to access multi-byte registers. However, the registers should reside in a byte-oriented address space, and the
- address of a multi-byte register should be the address of its first byte. Thus, addresses of contiguous multi-
- 582 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
- 585 the Current Location protocol.
- The motivation for a general multi-byte protocol rather than fixing the registers at 16-bits width is flexibility.
- The protocol described in the following paragraphs provides a way of transferring 16-bit, 32-bit or 64-bit
- values over a 16-bit index, 8-bit data, two-wire serial link while ensuring that the bytes of data transferred
- for a multi-byte register value are always consistent (temporally coherent).
- Using this protocol a single CCI message can contain one, two or all of the different register widths used
- within a device.
- The MS byte of a multi-byte register shall be located at the lowest address and the LS byte at the highest
- 593 address.

594 The address of the first byte of a multi-byte register may, or may not be, aligned to the size of the register; 595

i.e., a multiple of the number of register bytes. The register alignment is an implementation choice between

596 processing optimized and bandwidth optimized organizations. There are no restrictions on the number or mix 597

of multi-byte registers within the available 64K by 8-bit index space, with the exception that rules for the

valid locations for the MS bytes and LS bytes of registers are followed. 598

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

601 accessed second, etc.

602

603

604

605

606

609 610 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
- 607 Section 6.3.3 describes example behavior for the re-timing of multi-byte register accesses.
- 608 Without re-timing, data may be corrupted as illustrated in Figure 10 and Figure 11.

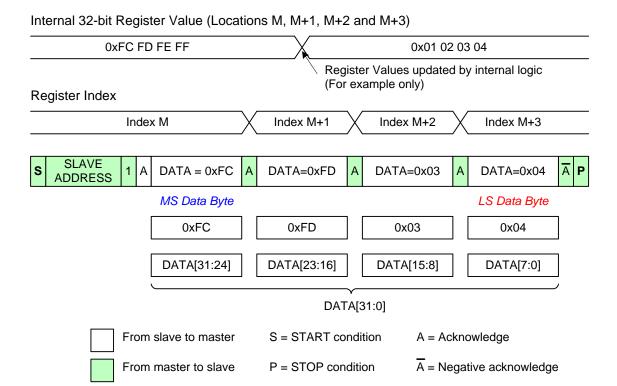


Figure 10 Corruption of a 32-bit Wide Register during a Read Message

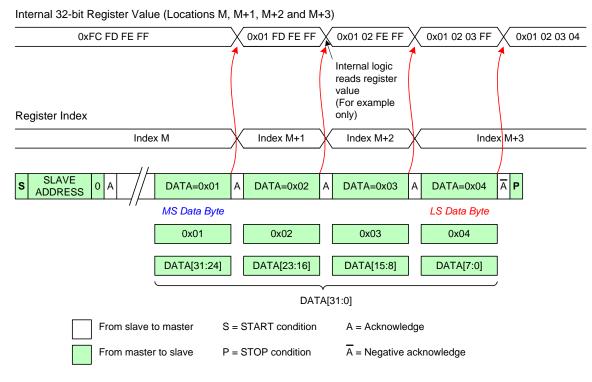


Figure 11 Corruption of a 32-bit Wide Register during a Write Message

# 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

616 LS byte.

611 612

613

617

618

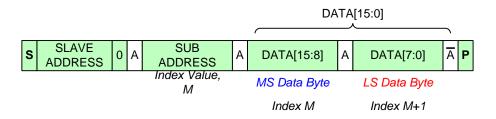


Figure 12 Example 16-bit Register Write

621 622

623

624

627

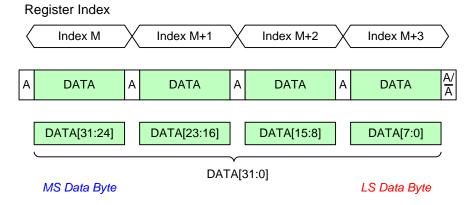


Figure 13 Example 32-bit Register Write (address not shown)

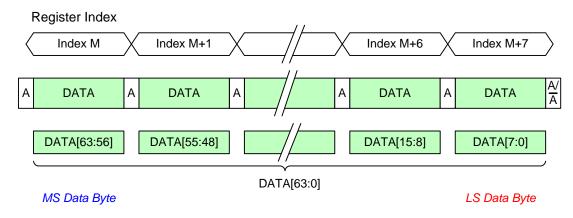


Figure 14 Example 64-bit Register Write (address not shown)

## 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
- 633 register.

635

636

637 638

639

640

641

643

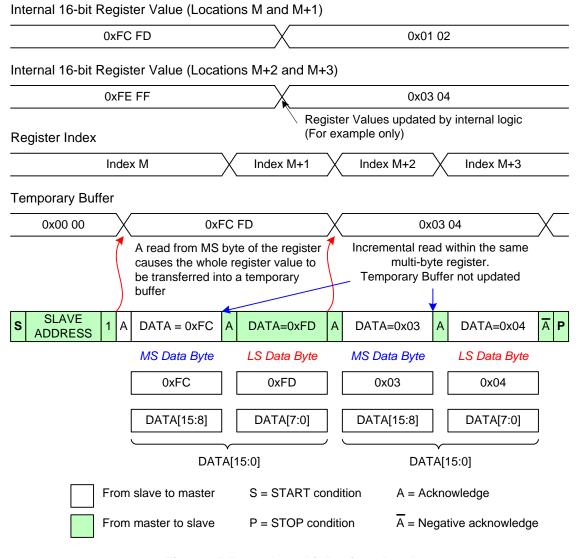
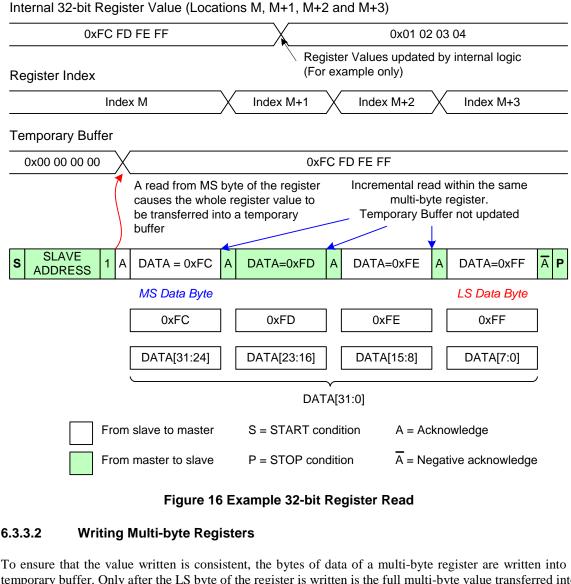


Figure 15 Example 16-bit Register Read

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.



648

649

650

651 652

- 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.
- 653 Figure 17 and Figure 18 illustrate multi-byte register write operations.
- 654 CCI messages that only write to the LS or MS byte of a multi-byte register are not allowed. Single byte writes 655 to a multi-byte register addresses may cause undesirable behavior in the device.

658

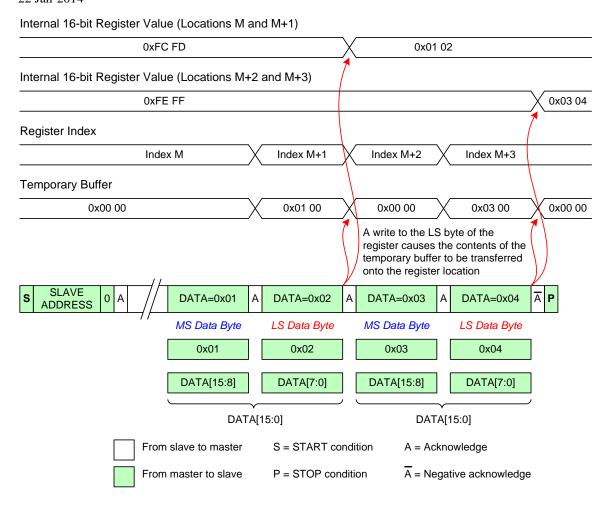


Figure 17 Example 16-bit Register Write

660

661

662 663

664

Internal 32-bit Register Value (Locations M, M+1, M+2 and M+3) 0xFC FD FE FF 0x01 02 03 04 Register Index Index M Index M+1 Index M+2 Index M+3 **Temporary Buffer** 0x00 00 00 00 0x01 00 00 00 0x01 02 00 00 0x01 02 03 00 0x00 00 00 00 A write to the LS byte of the register causes the contents of the temporary buffer to be transferred onto the register location SLAVE DATA=0x01 DATA=0x02 DATA=0x03 DATA=0x04 **ADDRESS** MS Data Byte LS Data Byte 0x01 0x02 0x03 0x04 DATA[31:24] DATA[23:16] DATA[15:8] DATA[7:0] DATA[31:0] From slave to master S = START condition A = Acknowledge  $\overline{A}$  = Negative acknowledge P = STOP condition From master to slave

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	VIL	-0.5	0.3V <sub>DD</sub>	-0.5	0.3 V <sub>DD</sub>	V
HIGH level input voltage	ViH	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>H</sub> ys	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	Vol1 Vol3	O N/A	0.4 N/A	0	0.4 0.2V <sub>DD</sub>	V

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

#### 665 Notes:

668

669

666 1.  $Maximum VIH = V_{DDmax} + 0.5V$ 

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

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

# **Table 2 CCI Timing Specification**

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

Version 1.2 22 Jan-2014

Parameter	Symbol	Standard-mode		Fast-mode		Unit
		Min.	Max.	Min.	Max.	
Noise margin at the LOW level for each connected device (including hysteresis)	V <sub>nL</sub>	0.1V <sub>DD</sub>	-	0.1V <sub>DD</sub>	-	V
Noise margin at the HIGH level for each connected device (including hysteresis)	V <sub>nH</sub>	0.2V <sub>DD</sub>	-	0.2V <sub>DD</sub>	-	V

#### 670 Notes:

671

672

673

674

675

676

677

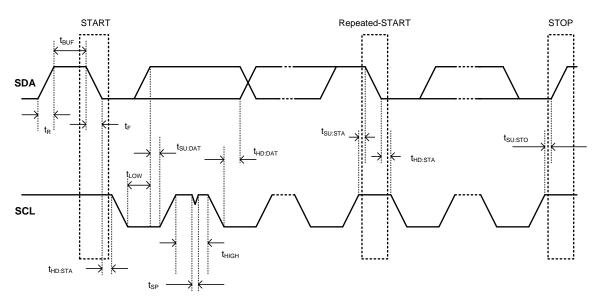
678

679

680

681

- 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<sub>IHmin</sub> of the SCL signal) to bridge the undefined region of the falling edge of SCL
- The maximum t<sub>HD:DAT</sub> has only to be met if the device does not the LOW period (t<sub>LOW</sub>) 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<sub>SU:DAT</sub> ≥ 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<sub>fMAX</sub> + t<sub>SU:DAT</sub> = 1000 + 250 = 1250 ns (according to the Standard-mode I2C bus specification) before the SCL line is released.
- 5. CB = total capacitance of one bus line in pF.
- The CCI timing is illustrated in Figure 19.



**Figure 19 CCI Timing** 

683 684 685

Copyright © 2005-2014 MIPI Alliance, Inc.

# 686 7 Physical Layer

- 687 CSI-2 uses the physical layer described in [MIPI01].
- The physical layer for a CSI-2 implementation is composed of a number of unidirectional data Lanes and
- one clock Lane. All CSI-2 transmitters and receivers shall support continuous clock behavior on the Clock
- Lane, and optionally may support non-continuous clock behavior.
- 691 For continuous clock behavior the Clock Lane remains in high-speed mode, generating active clock signals
- between the transmission of data packets.
- 693 For non-continuous clock behavior the Clock Lane enters the LP-11 state between the transmission of data
- 694 packets.
- 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.
- To enable higher data rates and higher number of lanes the physical layer described in [MIPI01] includes an
- 703 independent deskew mechanism in the Receive Data Lane Module. To facilitate deskew calibration at the
- receiver the transmitter Data Lane Module provides a deskew sequence pattern.
- 705 Since deskew calibration is only valid at a given transmit frequency:
- For initial calibration sequence the Transmitter shall be programmed with the desired frequency for calibration. It will then transmit the deskew calibration pattern and the Receiver will autonomously detect this pattern and tune the deskew function to achieve optimum performance.
- For any transmitter frequency changes the deskew calibration shall be rerun.
- Some transmitters and/or receiver may require deskew calibration to be rerun periodically and it is suggested that it can be optimally done within vertical or frame blanking periods.
- 712 For low transmit frequencies or when a receiver described in [MIPI01] is paired with a previous version
- transmitter not supporting the deskew calibration pattern the receiver may be instructed to bypass the deskew
- 714 mechanism.

716

717

718

719

720

721

722

723

724

725

# 8 Multi-Lane Distribution and Merging

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

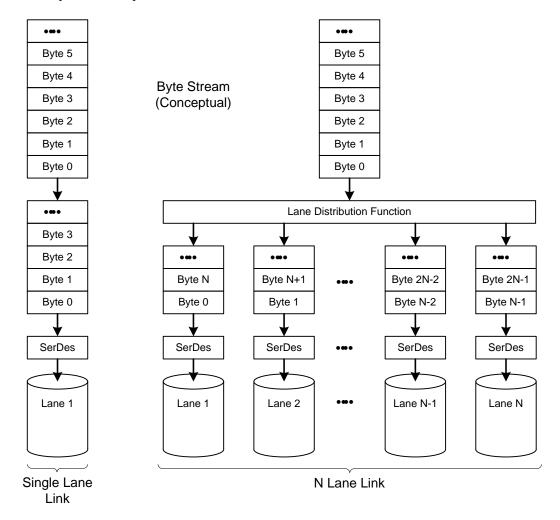
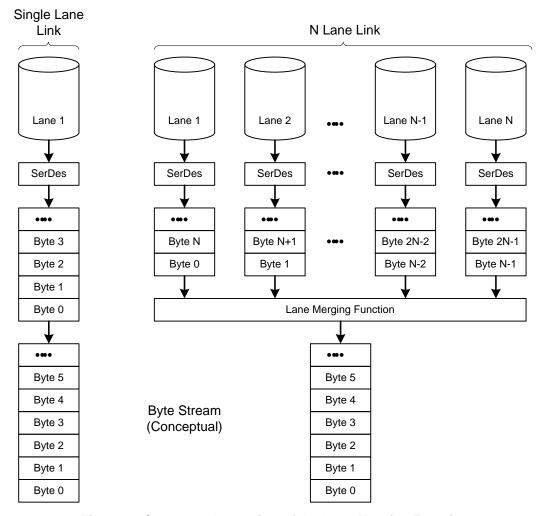


Figure 20 Conceptual Overview of the Lane Distributor Function



730

731

732

733

734

735

736

737738

739740

741742

743

744

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.
- N-Lane system (Figure 24): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte N-1 goes to Lane N, byte N goes to Lane 1, byte N+1 goes to Lane 2, and so on.
- N-lane system (Figure 25) with N>4 short packet (4 bytes) transmission: byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 goes to Lane 3, byte 3 goes to Lane 4, and Lanes 5 to N do not receive bytes and stay in LPS state.

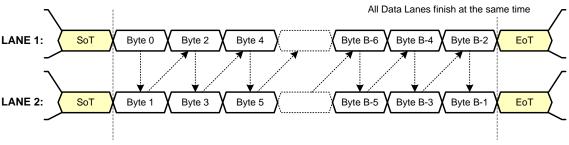
EoT – End of Transmission

Version 1.2 22 Jan-2014

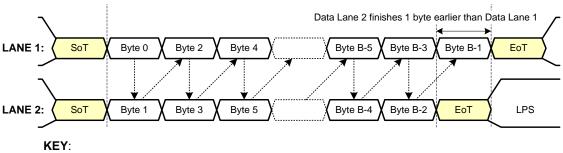
- At the end of the transmission, there may be "extra" bytes since the total byte count may not be an integer
- multiple of the number of Lanes, N. One or more Lanes may send their last bytes before the others. The Lane
- distributor, as it buffers up the final set of less-than-N bytes in parallel for sending to N data Lanes, de-asserts
- 748 its "valid data" signal into all Lanes for which there is no further data. For systems with more than 4 data
- Lanes sending a short packet constituted of 4 bytes the Lanes which do not receive a byte for transmission
- shall stay in LPS state.
- Each D-PHY data Lane operates autonomously.
- Although multiple Lanes all start simultaneously with parallel "start packet" codes, they may complete the
- transaction at different times, sending "end packet" codes one cycle (byte) apart.
- The N PHYs on the receiving end of the link collect bytes in parallel, and feed them into the Lane-merging
- 755 layer. This reconstitutes the original sequence of bytes in the transmission, which can then be partitioned into
- 756 individual packets for the packet decoder layer.

LPS - Low Power State

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



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

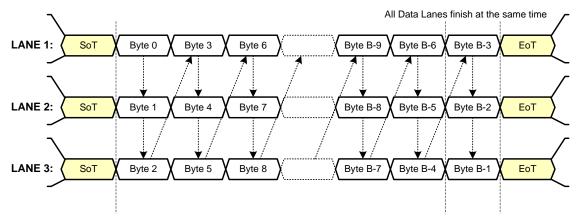


757758

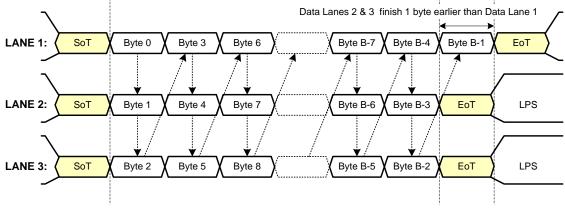
Figure 22 Two Lane Multi-Lane Example

SoT – Start of Transmission

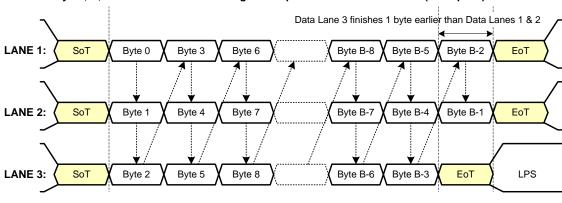
#### Number of Bytes, B, transmitted is an integer multiple of the number of lanes:



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



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



KEY:

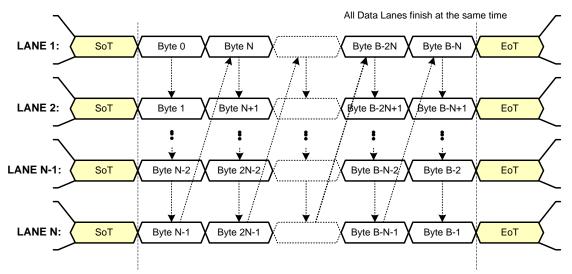
LPS – Low Power State

SoT - Start of Transmission

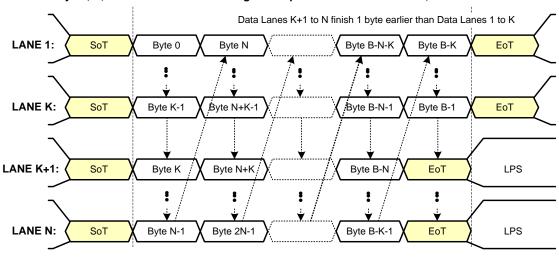
EoT – End of Transmission

Figure 23 Three Lane Multi-Lane Example

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



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



KEY:

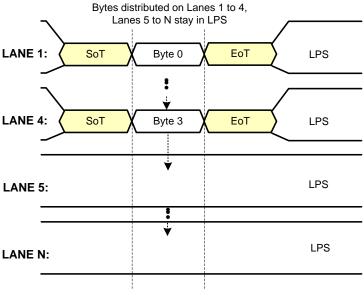
LPS - Low Power State

SoT - Start of Transmission

EoT – End of Transmission

Figure 24 N Lane Multi-Lane Example

#### Short packet of 4 bytes Transmitted on N lanes > 4



KEY:

LPS – Low Power State

SoT - Start of Transmission

EoT – End of Transmission

764765

766

767

768

769

770

771

772

773

775

776

777

778

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

# 8.1 Multi-Lane Interoperability

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

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

#### 774 Two cases:

- If M<=N then there is no loss of performance the receiver has sufficient data Lanes to match the transmitter (Figure 26 and Figure 27).
- If M> N then there may be a loss of performance (e.g. frame rate) as the receiver has fewer data Lanes than the transmitter (Figure 28 and Figure 29).

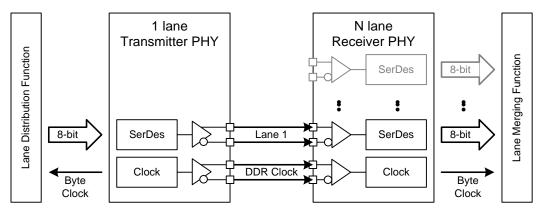
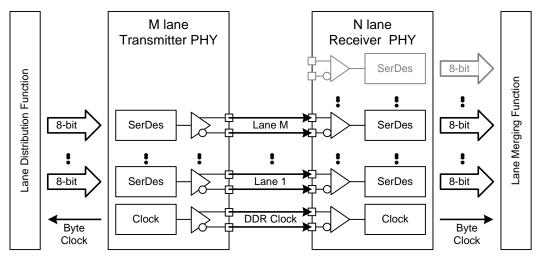


Figure 26 One Lane Transmitter and N Lane Receiver Example



781 782

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

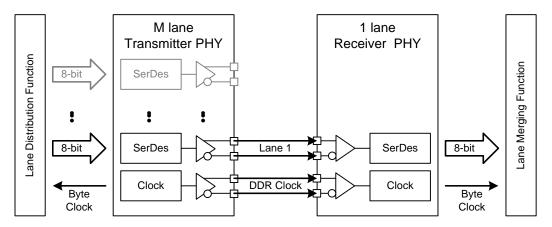


Figure 28 M Lane Transmitter and One Lane Receiver Example

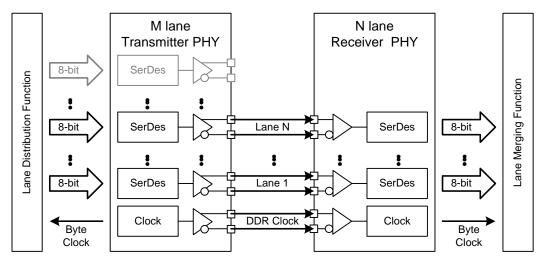
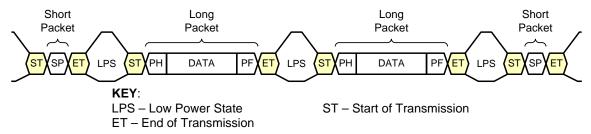


Figure 29 M Lane Transmitter and N Lane Receiver Example (N<M)

#### 787 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.
- 791 Low Level Protocol Features:
- Transport of arbitrary data (Payload independent)
- 793 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
- 797 16-bit Checksum Code for error detection.

#### DATA:



798 799

800

805

806

807

808

809

810

811

796

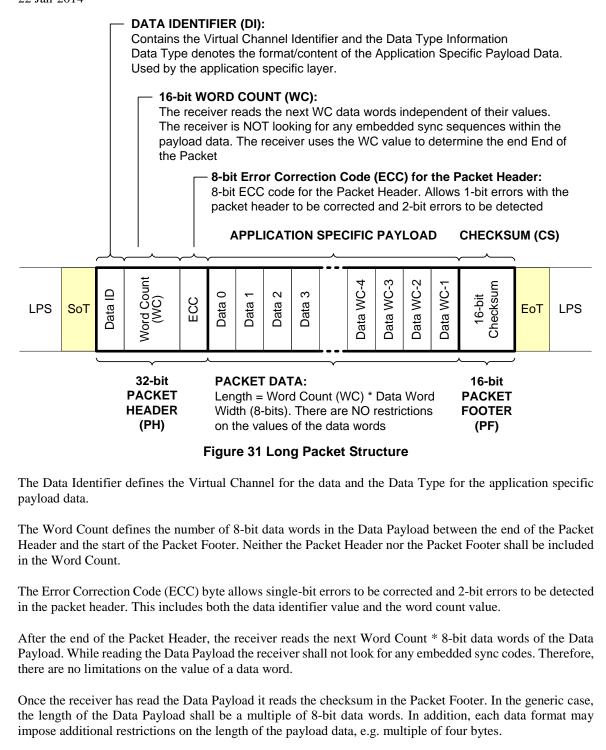
Figure 30 Low Level Protocol Packet Overview

#### 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 31 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 Section 9.2 through Section 9.5 for further descriptions of the packet elements.



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

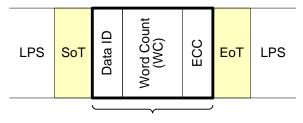
Short packet 16-bit Data Field shall be transmitted least significant byte first.

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

#### 9.1.2 Low Level Protocol Short Packet Format

- Figure 32 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.
- 836 For Frame Synchronization Data Types the Short Packet Data Field shall be the frame number. For Line
- 837 Synchronization Data Types the Short Packet Data Field shall be the line number. See Table 6 for a
- 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.



**32-bit SHORT PACKET (SH)** Data Type (DT) = 0x00 - 0x0F

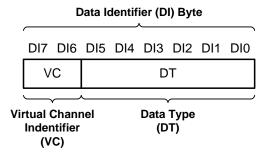
842843

844

Figure 32 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 33. 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.



848 849

850

Figure 33 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

855

856

857

858

859

860

861862

863

864

865

866

867 868 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 34.

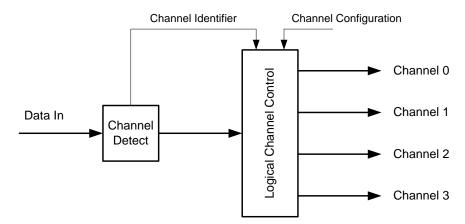


Figure 34 Logical Channel Block Diagram (Receiver)

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

EoT - End of Transmission

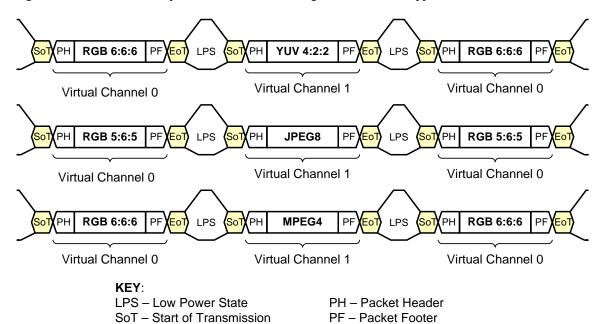


Figure 35 Interleaved Video Data Streams Examples

### 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.

870

871

872

873

874

875

876

877

878

879

880

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

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

### **Table 3 Data Type Classes**

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

#### 9.5 Packet Header Error Correction Code

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

The Data Identifier field DI[7:0] shall map to D[7:0] of the ECC input, the Word Count LS Byte (WC[7:0]) to D[15:8] and the Word Count MS Byte (WC[15:8]) to D[23:16]. This mapping is shown in Figure 36, which also serves as an ECC calculation example.

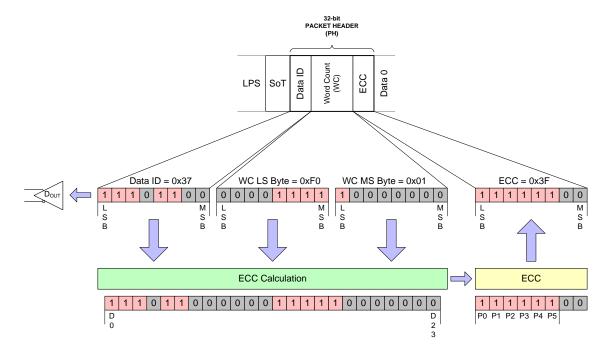


Figure 36 24-bit ECC Generation Example

881 882

Copyright © 2005-2014 MIPI Alliance, Inc. All rights reserved.

883

884

885

887

888

889 890

891 892

899

900

901

902

903

904

905

906

907

908

909

### 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:

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

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

- $\mathbf{G} = [\mathbf{I} \mid \mathbf{A}]$
- The packet header plus the ECC code can be obtained as: PH = p\*G where p represents the header (24 or 64 bits) and **G** is the corresponding generator matrix.
- Validating the received code word r, involves multiplying it by a parity check to form s, the syndrome or parity check vector:  $s = \mathbf{H}^*PH$  where PH is the received packet header and  $\mathbf{H}$  is the parity check matrix:
- $\mathbf{H} = [\mathbf{A}^{\mathsf{T}} \mid \mathbf{I}]$

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

# 9.5.2 Hamming-Modified Code

The error correcting code used is a 7+1 bits Hamming-modified code (72,64) and the subset of it is 5+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 allows the same 6-bit syndromes to correct the first 24-bits of a 64-bit sequence. To specify a compact 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
0b110	0x8C	0x91	0x92	0x94	0x98	0xA1	0xA2	0xA4

915

916

917

918

919

920

923

924

925

926

927

928

929

930

	d2d1d0							
d5d4d3	0b000	0b001	0b010	0b011	0b100	0b101	0b110	0b111
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.

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:

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

Table 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
11	0	0	1	0	0	1	0	1	0x25
12	0	0	1	0	0	1	1	0	0x26

Version 1.2 22 Jan-2014

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
13	0	0	1	0	1	0	0	1	0x29
14	0	0	1	0	1	0	1	0	0x2A
15	0	0	1	0	1	1	0	0	0x2C
16	0	0	1	1	0	0	0	1	0x31
17	0	0	1	1	0	0	1	0	0x32
18	0	0	1	1	0	1	0	0	0x34
19	0	0	1	1	1	0	0	0	0x38
20	0	0	0	1	1	1	1	1	0x1F
21	0	0	1	0	1	1	1	1	0x2F
22	0	0	1	1	0	1	1	1	0x37
23	0	0	1	1	1	0	1	1	0x3B
24	0	1	0	0	0	0	1	1	0x43
25	0	1	0	0	0	1	0	1	0x45
26	0	1	0	0	0	1	1	0	0x46
27	0	1	0	0	1	0	0	1	0x49
28	0	1	0	0	1	0	1	0	0x4A
29	0	1	0	0	1	1	0	0	0x4C
30	0	1	0	1	0	0	0	1	0x51
31	0	1	0	1	0	0	1	0	0x52
32	0	1	0	1	0	1	0	0	0x54
33	0	1	0	1	1	0	0	0	0x58
34	0	1	1	0	0	0	0	1	0x61
35	0	1	1	0	0	0	1	0	0x62
36	0	1	1	0	0	1	0	0	0x64
37	0	1	1	0	1	0	0	0	0x68
38	0	1	1	1	0	0	0	0	0x70
39	1	0	0	0	0	0	1	1	0x83
40	1	0	0	0	0	1	0	1	0x85
41	1	0	0	0	0	1	1	0	0x86
42	1	0	0	0	1	0	0	1	0x89
43	1	0	0	0	1	0	1	0	0x8A
44	0	0	1	1	1	1	0	1	0x3D
45	0	0	1	1	1	1	1	0	0x3E
46	0	1	0	0	1	1	1	1	0x4F
47	0	1	0	1	0	1	1	1	0x57
48	1	0	0	0	1	1	0	0	0x8C
49	1	0	0	1	0	0	0	1	0x91

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

# 9.5.3 ECC Generation on TX Side

932 This is an informative section.

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

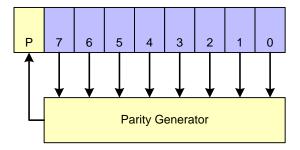


Figure 37 64-bit ECC Generation on TX Side

936 And Figure 38 for a 24-bit header:

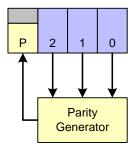


Figure 38 24-bit ECC Generation on TX Side

937938

934935

940

941

942

943 944

945 946

947

948

949

950

951 952

953

954

955

The parity generators are based on Table 5.

e.g. P3<sub>24-bit</sub> = D1^D2^D3^D7^D8^D9^D13^D14^D15^D19^D20^D21^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.

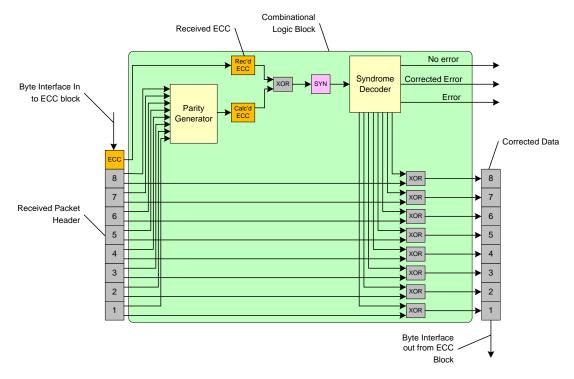
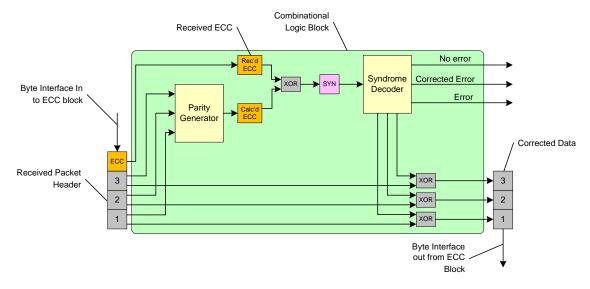


Figure 39 64-bit ECC on RX Side Including Error Correction

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, then a higher order error has occurred and the error flag will be set (the stream is corrupted and cannot be restored).
- Correcting the single error detected, as previously indicated.

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



960

961

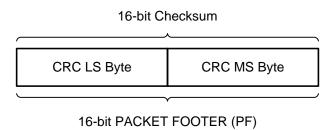
962

Figure 40 24-bit ECC on RX Side Including Error Correction

### 9.6 Checksum Generation

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

The transmission of the checksum is illustrated in Figure 41.



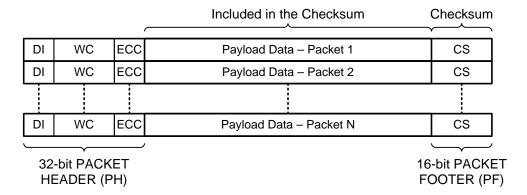
964 965

966

967

**Figure 41 Checksum Transmission** 

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



970

971

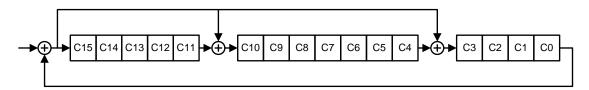
972

973

974 975

Figure 42 Checksum Generation for Packet Data

The definition of a serial CRC implementation is presented in Figure 43. The CRC implementation shall be functionally equivalent with the C code presented in Figure 44. 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 44 equals the final contents of the C[15:0] shift register shown in Figure 43. The checksum is then sent over CSI-2 bus to the receiver to verify that no errors have occurred in the transmission.



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

Figure 43 Definition of 16-bit CRC Shift Register

```
#define POLY 0x8408
                      /* 1021H bit reversed */
unsigned short crc16(char *data_p, unsigned short length)
   unsigned char i;
   unsigned int data;
   unsigned int crc = 0xffff;
   if (length == 0)
      return (unsigned short)(crc);
      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 44 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.

982

992

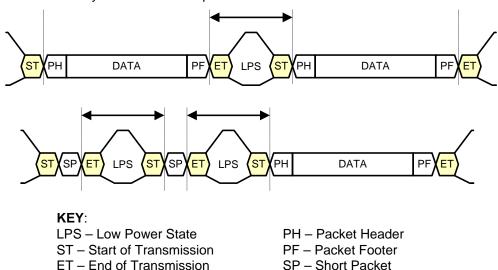
```
983
     Data:
984
     FF 00 00 02 B9 DC F3 72 BB D4 B8 5A C8 75 C2 7C 81 F8 05 DF FF 00 00 01
985
     Checksum LS byte and MS byte:
986
     F0 00
987
988
     Data:
989
     FF 00 00 00 1E F0 1E C7 4F 82 78 C5 82 E0 8C 70 D2 3C 78 E9 FF 00 00 01
990
     Checksum LS byte and MS byte:
991
   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

### 9.7 Packet Spacing

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



997 998

999

1000

1001

1002

1003

1013

Figure 45 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

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

1017

1026

1027

1028

10291030

1031

1032 1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

• Frame number increments by 1 for every FS packet with the same Virtual Channel and is periodically reset to one e.g. 1, 2, 1, 2, 1, 2 or 1, 2, 3, 4, 1, 2, 3, 4

1016 The frame number must be a non-zero value.

### 9.8.2 Line Synchronization Packets

- Line synchronization packets are optional.
- 1019 For Line Start (LS) and Line End (LE) synchronization packets the Short Packet Data Field shall contain a
- 1020 16-bit line number. This line number shall be the same for the LS and LE packets corresponding to a given
- line. Line numbers are logical line numbers and are not necessarily equal to the physical line numbers
- The 16-bit line number, when used, shall be non-zero to distinguish it from the case where line number is inoperative and remains set to zero.
- The behavior of the 16-bit line number shall be as one of the following:
- Line number is always zero line number is inoperative.
  - Line number increments by one for every LS packet within the same Virtual Channel and the same Data Type. The line number is periodically reset to one for the first LS packet after a FS packet. The intended usage is for progressive scan (non-interlaced) video data streams. The line number must be a non-zero value.
    - Line number increments by the same arbitrary step value greater than one for every LS packet within the same Virtual Channel and the same Data Type. The line number is periodically reset to a non-zero arbitrary start value for the first LS packet after a FS packet. The arbitrary start value may be different between successive frames. The intended usage is for interlaced video data streams.

### 9.9 Generic Short Packet Data Type Codes

Table 7 lists the Generic Short Packet Data Types.

#### **Table 7 Generic Short Packet Data Type Codes**

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

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

#### 9.10 **Packet Spacing Examples**

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

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

1046 respectively.

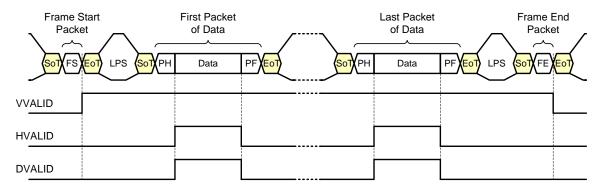
1043

1047

1048

Note that the VVALID, HVALID and DVALID signals in the figures in this section are only concepts to help illustrate the behavior of the frame start/end and line start/end packets. The VVALID, HVALID and

1049 DVALID signals do not form part of the Specification.



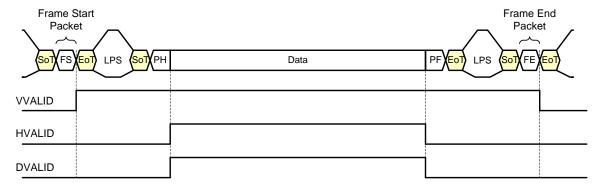
#### KEY:

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

PH - Packet Header PF - Packet Footer FS - Frame Start FE - Frame End LS - Line Start LE - Line End

1050 1051

### Figure 46 Multiple Packet Example

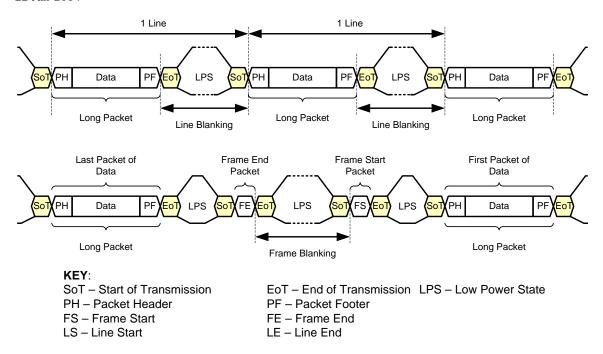


SoT - Start of Transmission EoT - End of Transmission LPS - Low Power State PH - Packet Header PF - Packet Footer FS - Frame Start FE - Frame End

LE - Line End

LS - Line Start

Figure 47 Single Packet Example



1056

1057

1060

1061

1062

1063

1064

1065

1066

1067

1068

1071

Figure 48 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.

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

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:

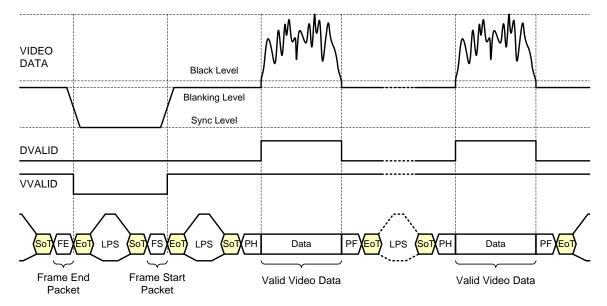
- The Frame Start packet to first data packet spacing should be as close as possible to the minimum packet spacing
- The last data packet to Frame End packet spacing should be as close as possible to the minimum packet spacing

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

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

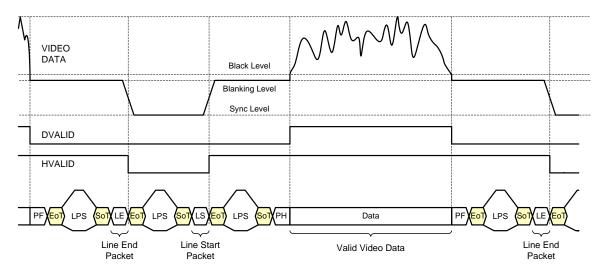
1074 Line Start and Line End packets shall be used for pixel level accurate horizontal synchronization timing 1075 information.

The positions of the Line Start and Line End packets, if present, can be varied within the Line Blanking Period in order to provide pixel accurate horizontal synchronization timing information. See Figure 50.



10781079

Figure 49 Vertical Sync Example



1080 1081

1082

1083

1084

1085

1086

Figure 50 Horizontal Sync Example

# 9.11 Packet Data Payload Size Rules

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

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

1096

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 51
- Digital Interlaced Video Example, Figure 52
- Digital Interlaced Video with accurate synchronization timing information, Figure 53

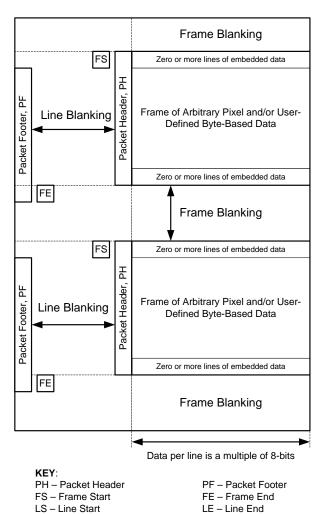
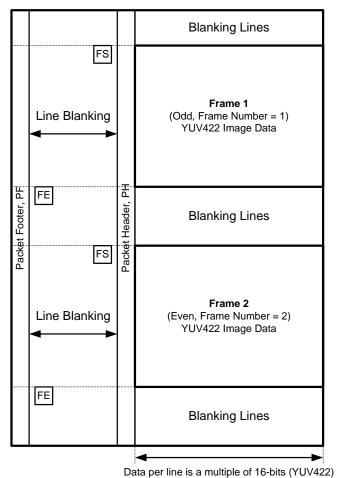


Figure 51 General Frame Format Example

1102



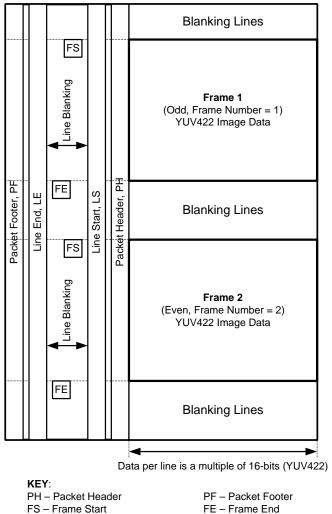
 KEY:

 PH – Packet Header
 PF – Packet Footer

 FS – Frame Start
 FE – Frame End

 LS – Line Start
 LE – Line End

Figure 52 Digital Interlaced Video Example



LS – Line Start LE – Line End

### Figure 53 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.
- 1111 There are two methods to interleave the transmission of different image data formats:
- Data Type

11061107

1108

1115

- Virtual Channel Identifier
- The preceding methods of interleaved data transmission can be combined in any manner.

# 9.13.1 Data Type Interleaving

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

1121

1122

1123

1125

1126

1127

1128

1129 1130

1131

1133 1134

1135

1136

1137 1138

- 1119 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
- 1124 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
- 1132 Data formats are defined further in Section 11.

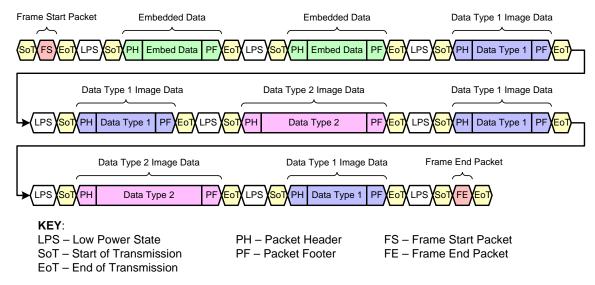
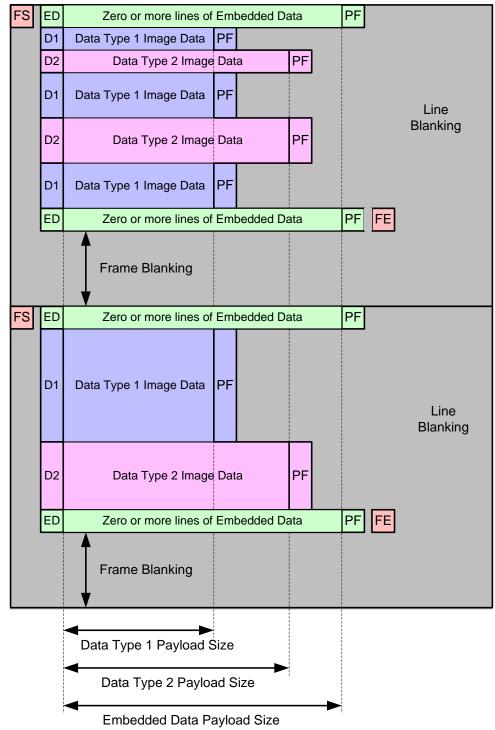


Figure 54 Interleaved Data Transmission using Data Type Value

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.

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



# KEY:

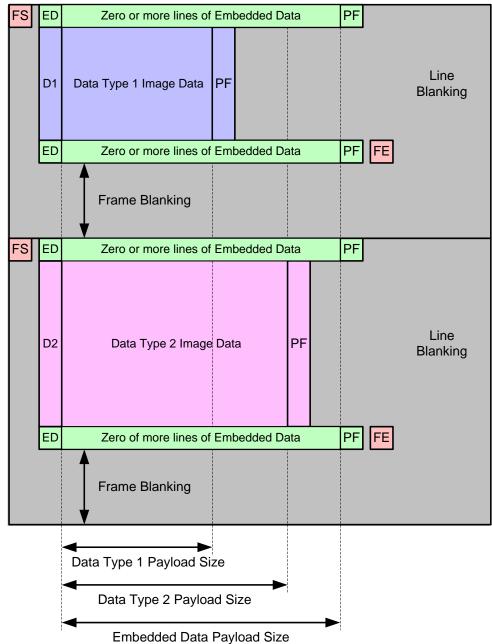
11411142

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

PF - Packet Footer

Figure 55 Packet Level Interleaved Data Transmission

Copyright © 2005-2014 MIPI Alliance, Inc. All rights reserved. Confidential



#### KEY:

1143

1144

1145

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

PF - Packet Footer

Figure 56 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 57 illustrates data interleaving using the Virtual Channel Identifier.

- Each virtual channel has its own Frame Start and Frame End packet. Therefore, it is possible for different virtual channels to have different frame rates, though the data rate for both channels would remain the same.
- In addition, Data Type value Interleaving can be used for each virtual channel, allowing different data types within a virtual channel and a second level of data interleaving.
- Therefore, receivers should be able to de-multiplex different data packets based on the combination of the Virtual Channel Identifier and the Data Type value. For example, data packets containing the same Data
- 1154 Type value but transmitted on different virtual channels are considered to belong to different frames (streams)
- of image data.

1156

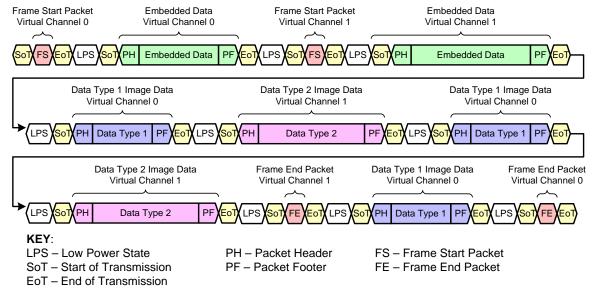


Figure 57 Interleaved Data Transmission using Virtual Channels

# 1158 10 Color Spaces

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

# 1162 10.1 RGB Color Space Definition

- In this Specification, the abbreviation RGB means the nonlinear sR'G'B' color space in 8-bit representation
- based on the definition of sRGB in IEC 61966.
- 1165 The 8-bit representation results as RGB888. The conversion to the more commonly used RGB565 format is
- achieved by scaling the 8-bit values to five bits (blue and red) and six bits (green). The scaling can be done
- either by simply dropping the LSBs or rounding.

# 1168 10.2 YUV Color Space Definition

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

#### 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
- 1175 configurations.

1171

1181

- 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	Р	
YUV422 10-bit		S
RGB888	Р	
RGB666		S
RGB565	Р	
RGB555		S
RGB444		S
RAW6		S
RAW7		S
RAW8	Р	
RAW10	Р	
RAW12		S
RAW14		S

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

1182 Notes:

1186

1188

1189

1194

- 1. Compressed image data should use the user defined, byte-based data type codes
- For clarity the Start of Transmission and End of Transmission sequences in the figures in this section have been omitted.

# 11.1 Generic 8-bit Long Packet Data Types

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

### Table 9 Generic 8-bit Long Packet Data Types

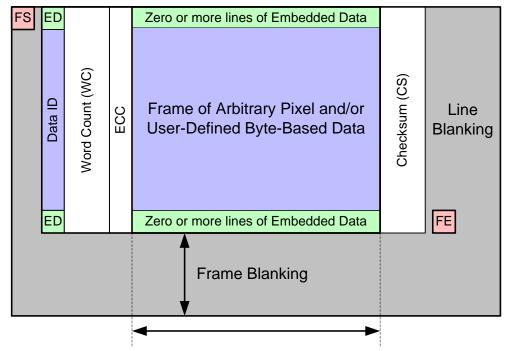
Data Type	Description
0x10	Null
0x11	Blanking Data
0x12	Embedded 8-bit non Image Data
0x13	Reserved
0x14	Reserved
0x15	Reserved
0x16	Reserved
0x17	Reserved

# 11.1.1 Null and Blanking Data

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

# 11.1.2 Embedded Information

- 1195 It is possible to embed extra lines containing additional information to the beginning and to the end of each
- 1196 picture frame as presented in the Figure 58. If embedded information exists, then the lines containing the
- embedded data must use the embedded data code in the data identifier.
- There may be zero or more lines of embedded data at the start of the frame. These lines are termed the frame
- 1199 header.
- 1200 There may be zero or more line of embedded data at the end of the frame. These lines are termed the frame
- 1201 footer.



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

V		/.
r	_	

1204

1205

1206

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

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

# 11.2 YUV Image Data

- Table 10 defines the data type codes for YUV data formats described in this section. The number of lines transmitted for the YUV420 data type shall be even.
- YUV420 data formats are divided into legacy and non-legacy data formats. The legacy YUV420 data format is for compatibility with existing systems. The non-legacy YUV420 data formats enable lower cost implementations.

# 1210 Table 10 YUV Image Data Types

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

# 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

1214 (2, 4, 6 ...). This sequence is illustrated in Figure 59.

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

the values in the table.

Table 11 Legacy YUV420 8-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

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

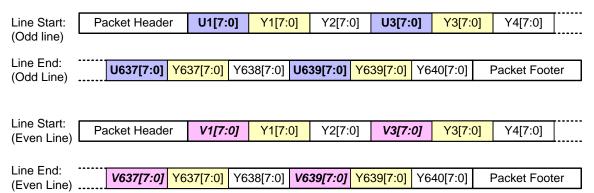


Figure 59 Legacy YUV420 8-bit Transmission

1220

1211

1224

1225

12261227

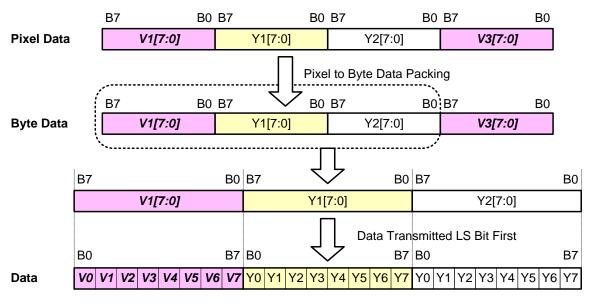


Figure 60 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

There is one spatial sampling option

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

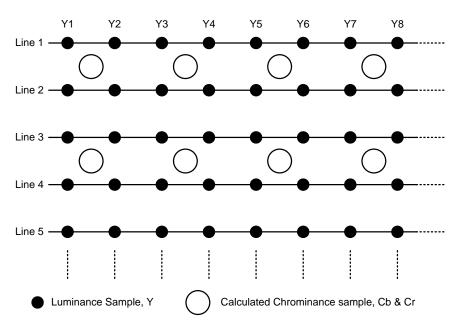
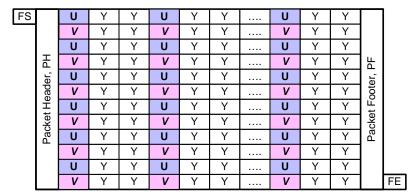


Figure 61 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1

Version 1.2 22 Jan-2014



12281229

1230

1231

1232

1233 1234

1235

1240

Figure 62 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 63.

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.

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

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

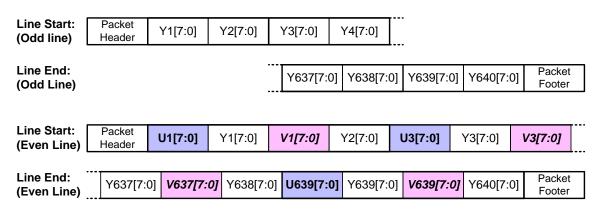
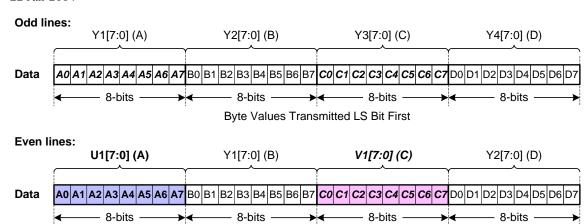


Figure 63 YUV420 8-bit Data Transmission Sequence



12481249

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

Byte Values Transmitted LS Bit First

1247 There are two spatial sampling options

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

Figure 67 shows the YUV420 frame format.

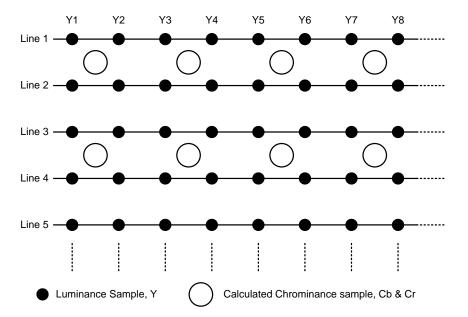


Figure 65 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1

12551256

1257

1258

1259 1260

1261

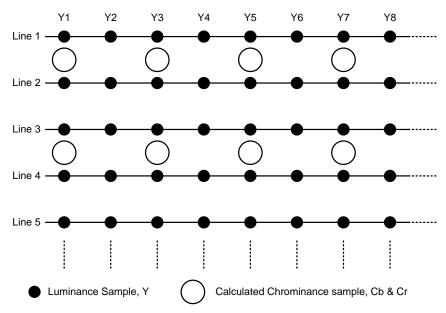


Figure 66 YUV420 Spatial Sampling for MPEG 2 and MPEG 4

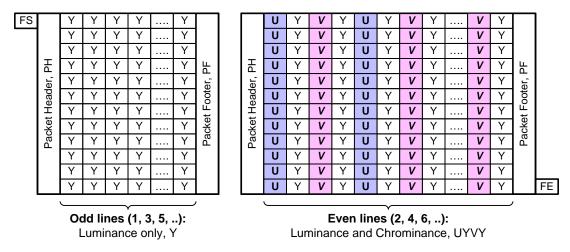


Figure 67 YUV420 8-bit Frame Format

### 11.2.3 YUV420 10-bit

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

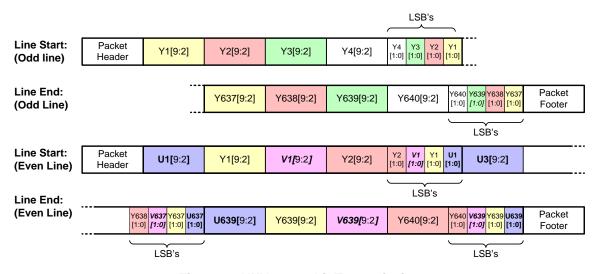
The payload data size, in bytes, for even lines (UYVY) is double the payload data size for odd lines (Y). This is exception to the general CSI-2 rule that each line shall have an equal length.

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

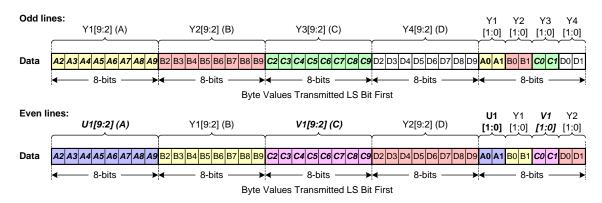
Odd Lines (1, 3, 5) Luminance Only, Y		Even Lines (2, 4, 6) Luminance and Chrominance, UYVY			
Pixels	Bytes	Bits	Pixels	Bytes	Bits
4	5	40	4	10	80

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



12691270

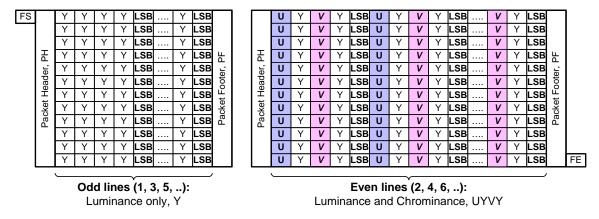
Figure 68 YUV420 10-bit Transmission



12711272

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

1273 The pixel spatial sampling options are the same as for the YUV420 8-bit data format.



1276

1279

12801281

1284

Figure 70 YUV420 10-bit Frame Format

# 11.2.4 YUV422 8-bit

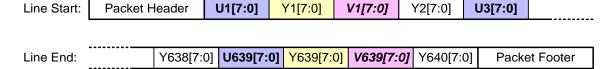
1277 YUV422 8-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in Figure 71.

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 72.



1285 Figure 71 YUV422 8-bit Transmission

12881289

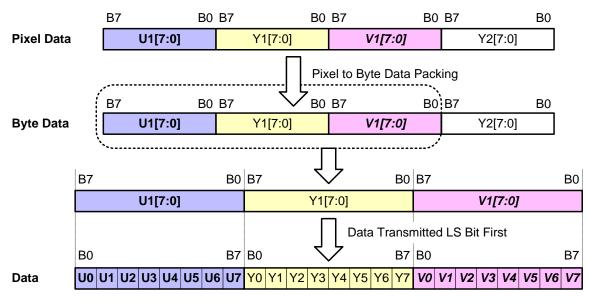


Figure 72 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration

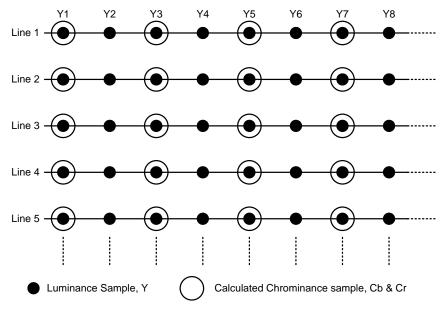
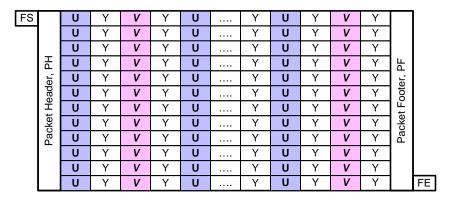


Figure 73 YUV422 Co-sited Spatial Sampling

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



1294

1299

Figure 74 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 75.

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 76.

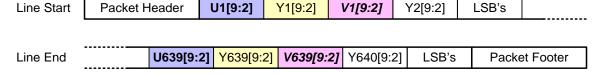
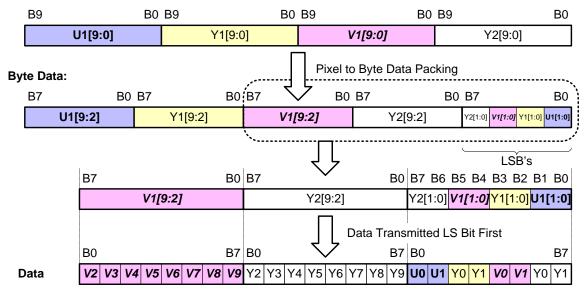


Figure 75 YUV422 10-bit Transmitted Bytes

#### Pixel Data:



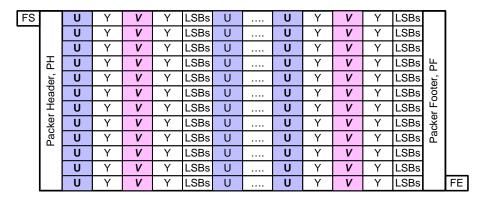
1304 1305

1306

1307

Figure 76 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration

The pixel spatial alignment is the same as in the YUV422 8-bit data case. The frame format for YUV422 is presented in the Figure 77.



13081309

1310

1311

Figure 77 YUV422 10-bit Frame Format

# 11.3 RGB Image Data

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

1313

1318

1321 1322

1323

1324

Data

# **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

1314 RGB888 data transmission is performed by transmitting a BGR byte sequence. This sequence is illustrated in Figure 78. The RGB888 frame format is illustrated in Figure 80.

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 79.

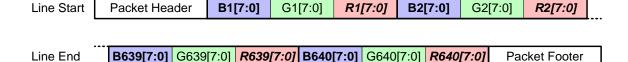


Figure 78 RGB888 Transmission

#### ·gano ro rro zooo rramonnoon

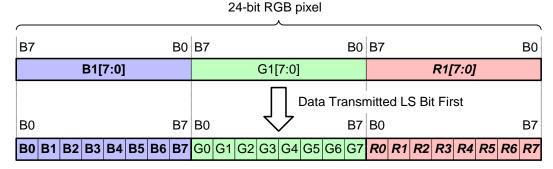


Figure 79 RGB888 Transmission in CSI-2 Bus Bitwise Illustration

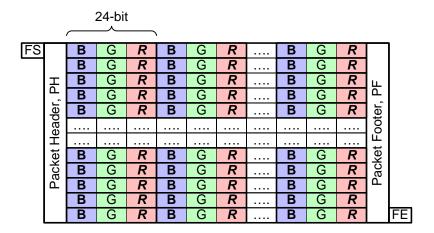


Figure 80 RGB888 Frame Format

1325

1327

1330

1331

1332

13361337

1338

1326

# 11.3.2 RGB666

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

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 82.

Line Start	Packet Header	BGR1[17:0]	BGR2[17:0]	BGR3[17:0]
Line End	BGR638[17:0	] BGR639[17:0]	BGR640[17:0	] Packet Footer

Figure 81 RGB666 Transmission with 18-bit BGR Words

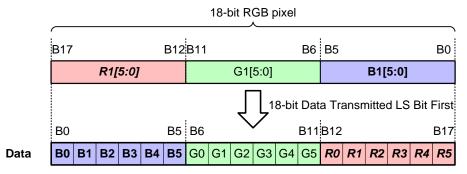
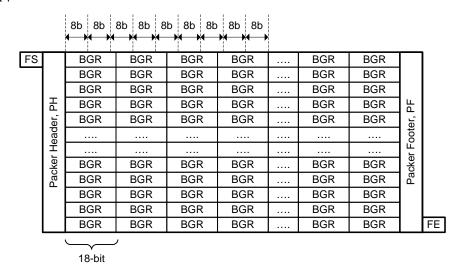


Figure 82 RGB666 Transmission on CSI-2 Bus Bitwise Illustration

1339 **Figure 8**2



1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

Figure 83 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 84. The frame format for RGB565 is presented in the Figure 86.

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 85.

Line Start	Packet Header	BGR1[15:0]	BGR2[15:0]	BGR3[15:0]
Line End	BGR638[15:0	] BGR639[15:0	] BGR640[15:0	Packet Footer

Figure 84 RGB565 Transmission with 16-bit BGR Words

13551356

1357

1358

1359

1360

1361

13651366

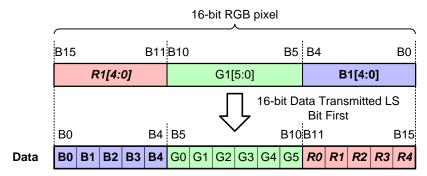


Figure 85 RGB565 Transmission on CSI-2 Bus Bitwise Illustration

		16-bit						
		$\overline{}$	•					
FS		BGR	BGR	BGR	 BGR	BGR		1
`		BGR	BGR	BGR	 BGR	BGR	1	
		BGR	BGR	BGR	 BGR	BGR	1	
	PH	BGR	BGR	BGR	 BGR	BGR	] ዜ	
	er,	BGR	BGR	BGR	 BGR	BGR		
	Header,				 		Footer,	
					 		] L	
	ķe	BGR	BGR	BGR	 BGR	BGR	Packer	
	Packer	BGR	BGR	BGR	 BGR	BGR	Pa	
		BGR	BGR	BGR	 BGR	BGR		
		BGR	BGR	BGR	 BGR	BGR		
		BGR	BGR	BGR	 BGR	BGR		FE

Figure 86 RGB565 Frame Format

### 11.3.4 RGB555

RGB555 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB555 data should be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs of the green color component as illustrated in Figure 87.

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

Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB555 case the length of one data word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words; i.e. instead of flipping each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 87.

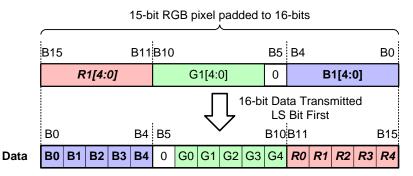


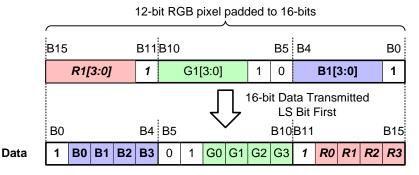
Figure 87 RGB555 Transmission on CSI-2 Bus Bitwise Illustration

Copyright © 2005-2014 MIPI Alliance, Inc. All rights reserved.

Version 1.2 22 Jan-2014

#### 11.3.5 **RGB444**

- 1368 RGB444 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB444 data should 1369 be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs of each
- 1370 color component as illustrated in Figure 88.
- 1371 Both the frame format and the package size constraints are the same as the RGB565 case.
- 1372 Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB444 case the length of one data 1373 word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words; i.e. instead of flipping each
- 1374 byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 88.



1375 1376

1377

1367

Figure 88 RGB444 Transmission on CSI-2 Bus Bitwise Illustration

#### 11.4 **RAW Image Data**

- 1378 The RAW 6/7/8/10/12/14 modes are used for transmitting Raw image data from the image sensor.
- 1379 The intent is that Raw image data is unprocessed image data (i.e. Raw Bayer data) or complementary color 1380 data, but RAW image data is not limited to these data types.
- 1381 It is possible to transmit e.g. light shielded pixels in addition to effective pixels. This leads to a situation
- where the line length is longer than sum of effective pixels per line. The line length, if not specified otherwise, 1382
- 1383 has to be a multiple of word (32 bits).
- 1384 Table 20 defines the data type codes for RAW data formats described in this section.

1386

1387

1388

1389

1390

1393 1394

1395 1396

# **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 done by transmitting the pixel data over CSI-2 bus. Each line is separated by line start / end synchronization codes. This sequence is illustrated in Figure 89 (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.

1391 Table 21 RAW6 Packet Data Size Constraints

Pixels	Bytes	Bits
4	3	24

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

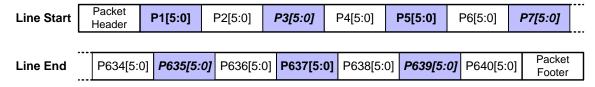


Figure 89 RAW6 Transmission

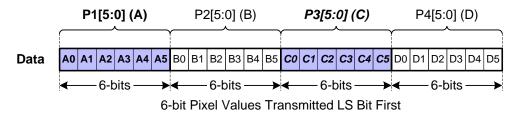
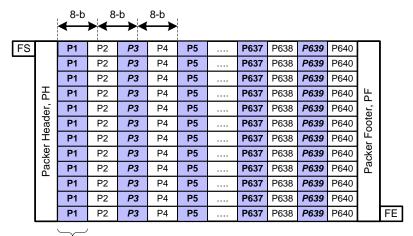


Figure 90 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration



6-bit Pixel Value

1397

1399

1400

1401

1402

1403

1404

1406 1407

1408 1409

1398

Figure 91 RAW6 Frame Format

#### 11.4.2 RAW7

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

**Table 22 RAW7 Packet Data Size Constraints** 

Pixels	Bytes	Bits
8	7	56

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

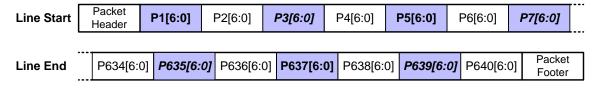
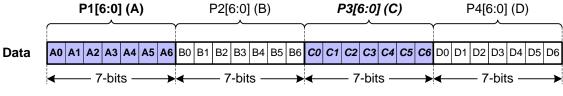
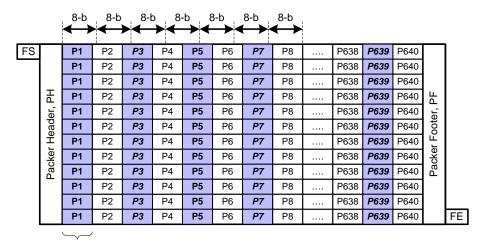


Figure 92 RAW7 Transmission



7-bit Pixel Values Transmitted LS Bit First

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



7-bit Pixel Value

14101411

1413

1414

1415

1416

1419

1420

14211422

Figure 94 RAW7 Frame Format

### 1412 **11.4.3 RAW8**

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

**Table 23 RAW8 Packet Data Size Constraints** 

Pixels	Bytes	Bits
1	1	8

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

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

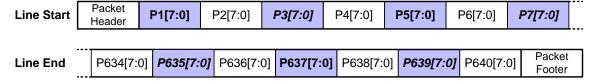


Figure 95 RAW8 Transmission

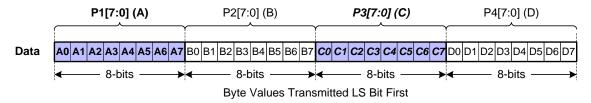


Figure 96 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration

Version 1.2 22 Jan-2014

FS		P1	P2	P3	P4	P5	 P637	P638	P639	P640			
		P1	P2	P3	P4	P5	 P637	P638	P639	P640	1		
		P1	P2	P3	P4	P5	 P637	P638	P639	P640	1		
	F	P1	P2	P3	P4	P5	 P637	P638	P639	P640	胎		
	er,	P1	P2	P3	P4	P5	 P637	P638	P639	P640			
	ader	P1	P2	P3	P4	P5	 P637	P638	P639	P640	ooter		
	r He	P1	P2	P3	P4	P5	 P637	P638	P639	P640	뜬		
	ackeı	P1	P2	P3	P4	P5	 P637	P638	P639	P640	Packe		
	Рас	P1	P2	P3	P4	P5	 P637	P638	P639	P640	Pa		
		P1	P2	P3	P4	P5	 P637	P638	P639	P640	1		
		P1	P2	P3	P4	P5	 P637	P638	P639	P640	l		
		P1	P2	P3	P4	P5	 P637	P638	P639	P640	1	FE	l

1423

1425

1426

1427 1428

1429

1424 Figure 97 RAW8 Frame Format

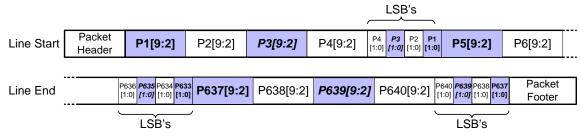
### 11.4.4 RAW10

The transmission of 10-bit Raw data is done by packing the 10-bit pixel data to look like 8-bit data format. Table 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

- 1430 This sequence is illustrated in Figure 98 (VGA case).
- Bit order in transmission follows the general CSI-2 rule, LSB first.



1433

1432

Figure 98 RAW10 Transmission

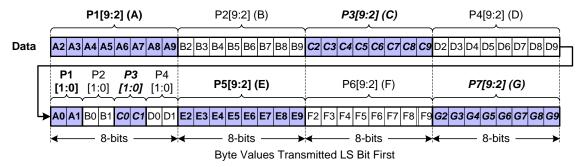


Figure 99 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration

Version 1.2 22 Jan-2014

FS		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		i
	1	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		i
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		İ
	표	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	H	i
	e,	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	_	l
	Header	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	ooter	l
	٠.	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	ᆫ	l
	ackeı	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	cke	l
	Рас	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	Ра	l
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		ĺ
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		l
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		FE

14361437

1438

1439

1440

1441

1442

Figure 100 RAW10 Frame Format

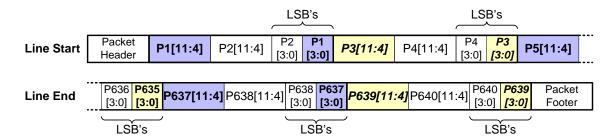
### 11.4.5 RAW12

The transmission of 12-bit Raw data is done by packing the 12-bit pixel data to look like 8-bit data format. Table 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

- 1443 This sequence is illustrated in Figure 101 (VGA case).
- Bit order in transmission follows the general CSI-2 rule, LSB first.



14451446

Figure 101 RAW12 Transmission

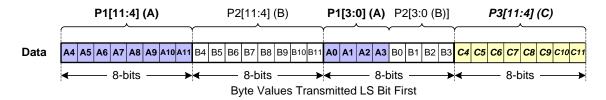


Figure 102 RAW12 Transmission on CSI-2 Bus Bitwise Illustration

FS		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs		1
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs		
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs		
	PH	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	H	
	er,	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	e,	
	Header	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	ooter	
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	Ē	
	acker	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	cke	
	Рас	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	Ра	
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs		
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs		
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs		F

1451

1452

1453

1454

1455

Figure 103 RAW12 Frame Format

### 11.4.6 RAW14

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

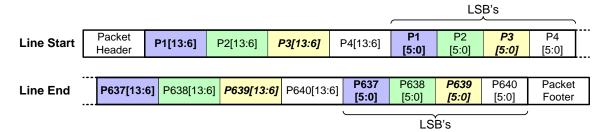
The length of each packet must be a multiple of the values in the table.

**Table 26 RAW14 Packet Data Size Constraints** 

Pixels	Bytes	Bits
4	7	56

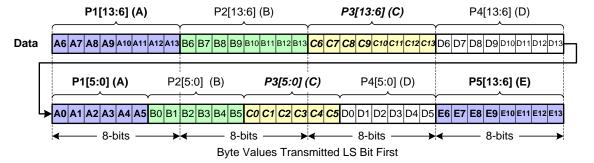
1456 The sequence is illustrated in Figure 104 (VGA case).

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



1460 1461

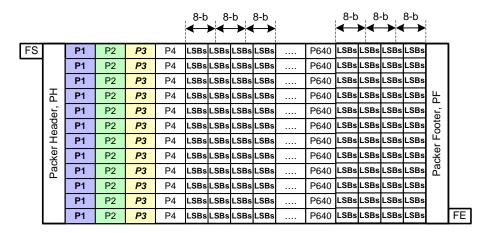
Figure 104 RAW14 Transmission



14621463

Figure 105 RAW14 Transmission on CSI-2 Bus Bitwise Illustration

Copyright © 2005-2014 MIPI Alliance, Inc. All rights reserved.



1465

1466

1467

1468 1469

14711472

14731474

1475

1477

# Figure 106 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.

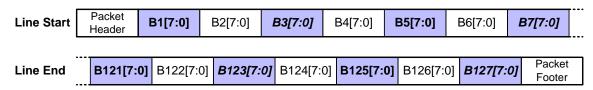


Figure 107 User Defined 8-bit Data (128 Byte Packet)

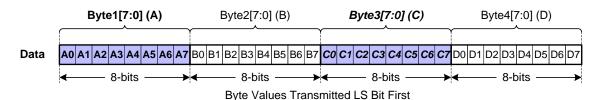
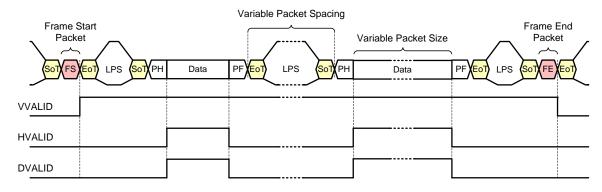


Figure 108 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration

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

1476 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.



KEY:

SoT – Start of Transmission

EoT – End of Transmission LPS – Low Power State PF – Packet Footer

PH – Packet Header FS – Frame Start

FE - Frame End

LE – Line End

1480 1481

1482

1483

# Figure 109 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
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

1490

1495 1496

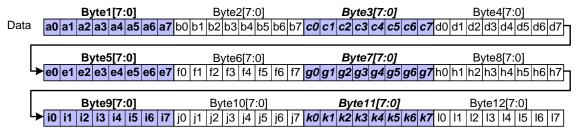
# 12 Recommended Memory Storage

- 1485 This section is informative.
- 1486 The CSI-2 data protocol requires certain behavior from the receiver connected to the CSI transmitter. The
- 1487 following sections describe how different data formats should be stored inside the receiver. While
- 1488 informative, this section is provided to ease application software development by suggesting a common data
- storage format among different receivers.

# 12.1 General/Arbitrary Data Reception

- In the generic case and for arbitrary data the first byte of payload data transmitted maps the LS byte of the
- 32-bit memory word and the fourth byte of payload data transmitted maps to the MS byte of the 32-bit
- memory word.
- Figure 110 shows the generic CSI-2 byte to 32-bit memory word mapping rule.

### Data on CSI-2 bus



Buffer	Data in receiver's buffer	r		
Addr	MSB Byte4[7:0]	Byte3[7:0]	Byte2[7:0]	Byte1[7:0] LSB
00h	d7 d6 d5 d4 d3 d2 d1 d0	c7 c6 c5 c4 c3 c2 c1 c0 b7	7 b6 b5 b4 b3 b2 b1 b0	a7 a6 a5 a4 a3 a2 a1 a0
	Byte8[7:0]	Byte7[7:0]	Byte6[7:0]	Byte5[7:0]
01h	h7 h6 h5 h4 h3 h2 h1 h0	g7 g6 g5 g4 g3 g2 g1 g0 f7	7 f6 f5 f4 f3 f2 f1 f0	e7 e6 e5 e4 e3 e2 e1 e0
	Byte12[7:0]	Byte11[7:0]	Byte10[7:0]	Byte9[7:0]
02h	17         16         15         14         13         12         11         10	k7 k6 k5 k4 k3 k2 k1 k0 j7	7 j6 j5 j4 j3 j2 j1 j0	i7   i6   i5   i4   i3   i2   i1   i0

32-bit standard memory width

1497 Figure 110 General/Arbitrary Data Reception

1500 1501

1502

15031504

# 12.2 RGB888 Data Reception

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

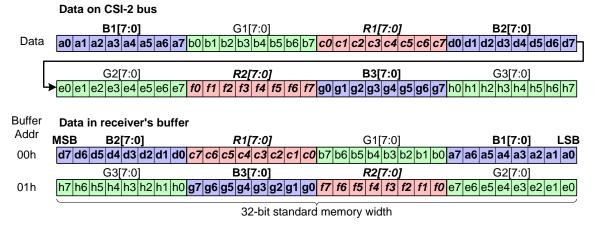


Figure 111 RGB888 Data Format Reception

# 12.3 RGB666 Data Reception

#### Data on CSI-2 bus B1[5:0] G1[5:0] R1[5:0] B2[5:0] G2[5:0] R2 Data a0 a1 a2 a3 a4 a5 b0 b1 b2 b3 b4 b5 c0 c1 c2 c3 c4 c5 d0 d1 d2 d3 d4 d5 e0 e1 e2 e3 e4 e5 f0 f1 G3[5:0] B3[5:0] R3[5:0] B4[5:0] G4 f2 f3 f4 f5 g0 g1 g2 g3 g4 g5 h0 h1 h2 h3 h4 h5 i0 i1 i2 i3 i4 i5 j0 j1 j2 j3 j4 j5 k0 k1 k2 k3 R4[5:0] G5[5:0] B5[5:0] R5[5:0] B6[5:0] k4 k5 10 11 12 13 14 15 m0 m1 m2 m3 m4 m5 n0 n1 n2 n3 n4 n5 00 01 02 03 04 05 p0 p1 p2 p3 p4 p5 Buffer Data in receiver's buffer Addr MSB R2 G2[5:0] B2[5:0] R1[5:0] B1[5:0] LSB G1[5:0] f1 f0 e5 e4 e3 e2 e1 e0 d5 d4 d3 d2 d1 d0 c5 c4 c3 c2 c1 c0 b5 b4 b3 b2 b1 b0 a5 a4 a3 a2 a1 a0 00h G4 B4[5:0] R3[5:0] G3[5:0] B3[5:0] R2 01h k3 k2 k1 k0 j5 j4 j3 j2 j1 j0 i5 i4 i3 i2 i1 i0 h5 h4 h3 h2 h1 h0 g5 g4 g3 g2 g1 g0 f5 f4 f3 f2 B6[5:0] R5[5:0] G5[5:0] B5[5:0] R4[5:0] G4 p5 p4 p3 p2 p1 p0 o5 o4 o3 o2 o1 o0 n5 n4 n3 n2 n1 n0 m5 m4 m3 m2 m1 m0 l5 l4 l3 l2 l1 l0 k5 k4 02h

Figure 112 RGB666 Data Format Reception

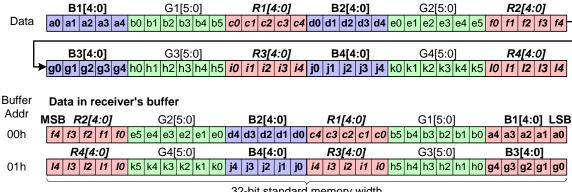
32-bit standard memory width

1508

1509 1510

# 1505 **12.4 RGB565 Data Reception**

#### Data on CSI-2 bus

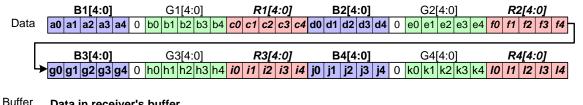


1506 32-bit standard memory width

Figure 113 RGB565 Data Format Reception

# 12.5 RGB555 Data Reception

#### Data on CSI-2 bus



Buller	Data in receiver's buffer																												
Addr	MSB R2[4:0]		G	G2[4:0]				B2[4:0]					R1[4:0]					G1[4:0]					B1[4:0] LS						
00h	f4 f	3 f2	f1	<b>f0</b> e <sup>2</sup>	1 e3	e2	e1	e0	0	d4	d3	d2	d1	d0	c4	c3	c2	c1	c0	4 b3	b2	b1	b0	0	a4	а3	a2 a	a1	a0
		<b>R4[4:0]</b> G4[4:0]						B4[4:0]						R	3[4.	:0]		G3[4:0]					B3[4:0]						
01h	14 I.	3 12	<i>I1</i>	<i>10</i> k4	1 k3	k2	k1	k0	0	j4	j3	j2	j1	j0	i4	i3	i2	i1	i0 h	14 h3	h2	h1	h0	0	g4	g3	g2 <u>g</u>	g1	g0
														$\overline{}$	=														=

32-bit standard memory width

Figure 114 RGB555 Data Format Reception

15141515

1516

1519 1520

1521

15221523

# 12.6 RGB444 Data Reception

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

#### Data on CSI-2 bus B1[3:0] G1[3:0] R1[3:0] B2[3:0] G2[3:0] R2[3:0] Data 1 | a0 | a1 | a2 | a3 | 0 | 1 | b0 | b1 | b2 | b3 | 1 | c0 | c1 | c2 | c3 1 **d0 d1 d2 d3** 0 e0 e1 e2 e3 f0 f1 f2 f3 R3[3:0] B4[3:0] B3[3:0] G3[3:0] G4[3:0] R4[3:0] 1 g0 g1 g2 g3 0 1 h0 h1 h2 h3 1 i0 i1 i2 i3 j0 j1 j2 j3 0 1 k0 k1 k2 k3 1 10 | I1 | I2 | I3 Buffer Data in receiver's buffer Addr **MSB** R2[3:0] G2[3:0] B2[3:0] G1[3:0] B1[3:0] LSB R1[3:0] X X X f3 f2 f1 f0 e3 e2 e1 e0 d3 d2 d1 d0 X X X X C3 C2 C1 C0 b3 b2 b1 b0 a3 a2 a1 a0 00h X R4[3:0] G4[3:0] B4[3:0] R3[3:0] G3[3:0] B3[3:0] 01h X | X | X | <mark>13 | 12 | 11 | 10 | k3 | k2 | k1 | k0 | *j*3 | *j*2 | *j*1 | *j*0 | X | X | X | X | X | **i3 | i2 | i1 | i0** | h3 | h2 | h1 | h0 | **g3 | g2 | g1 | g0**</mark>

32-bit standard memory width

Figure 115 RGB444 Data Format Reception

# 12.7 YUV422 8-bit Data Reception

Data on CSI-2 bus

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.

#### 

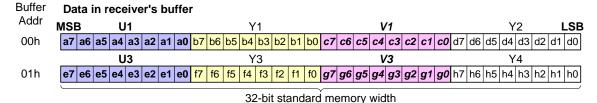


Figure 116 YUV422 8-bit Data Format Reception

# 1524 **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.

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

32-bit standard memory width

Figure 117 YUV422 10-bit Data Format Reception

<mark>| j9 | j8 | j7 | j6 | j5 | j4 | j3 | j2 | i9 | i8 | i7 | i6 | i5 | i4 | i3 | i2 |</mark> h1 | h0 | **g1 | g0 | f1 | f0 | e1 | e0 |** h9 | h8 | h7 | h6 | h5 | h4 | h3 | h2

# 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
- 1530 CSI-2 rule.

15261527

1528

02h

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

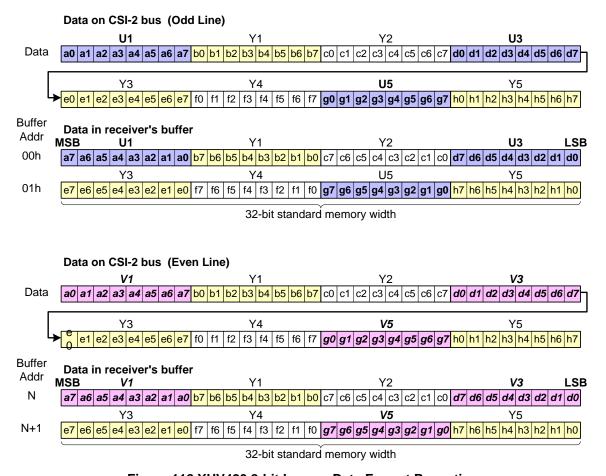


Figure 118 YUV420 8-bit Legacy Data Format Reception

15381539

# 12.10 YUV420 8-bit Data Reception

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

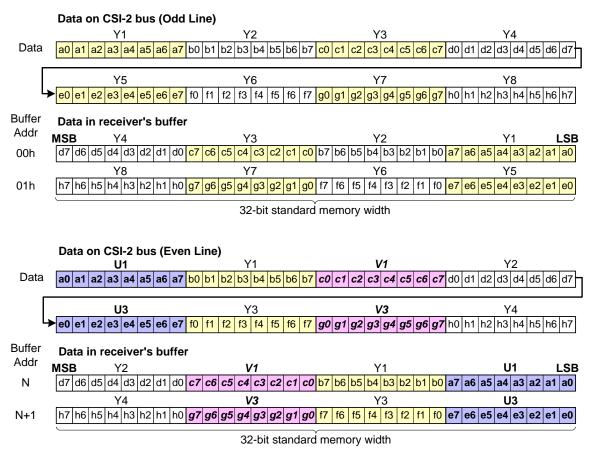


Figure 119 YUV420 8-bit Data Format Reception

15421543

### 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.

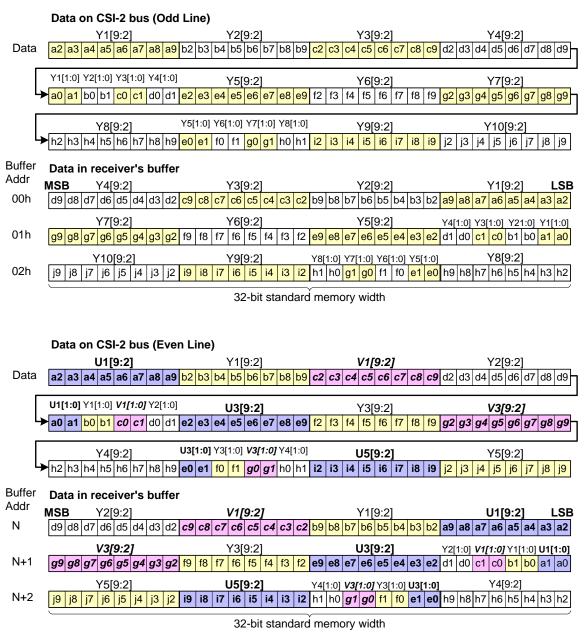
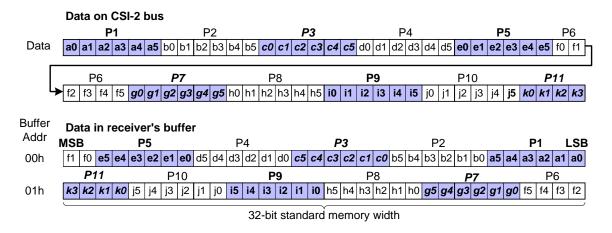


Figure 120 YUV420 10-bit Data Format Reception

# 1544 12.12 RAW6 Data Reception



1545 1546

1547

Figure 121 RAW6 Data Format Reception

# 1548 **12.13 RAW7 Data Reception**

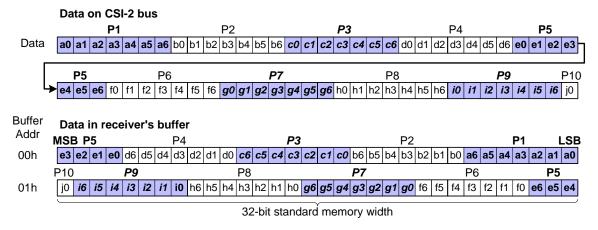


Figure 122 RAW7 Data Format Reception

15531554

1555

15571558

# 12.14 RAW8 Data Reception

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

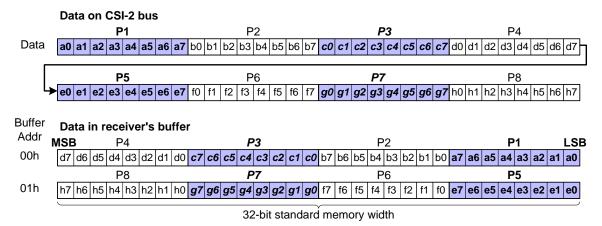


Figure 123 RAW8 Data Format Reception

# 12.15 RAW10 Data Reception

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

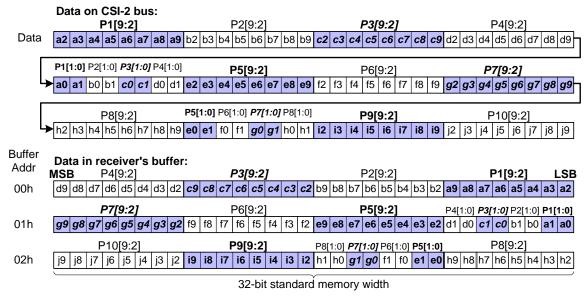


Figure 124 RAW10 Data Format Reception

1561

1562

1563

1564 1565

### 12.16 RAW12 Data Reception

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

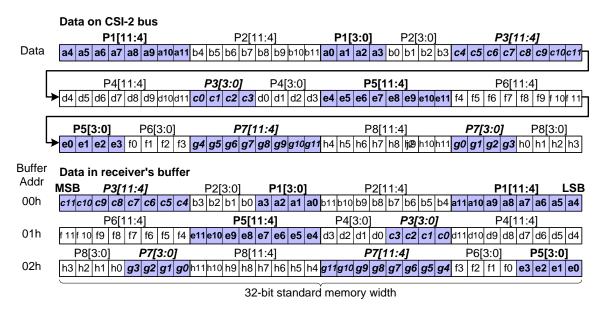


Figure 125 RAW12 Data Format Reception

# 12.17 RAW14 Data Reception

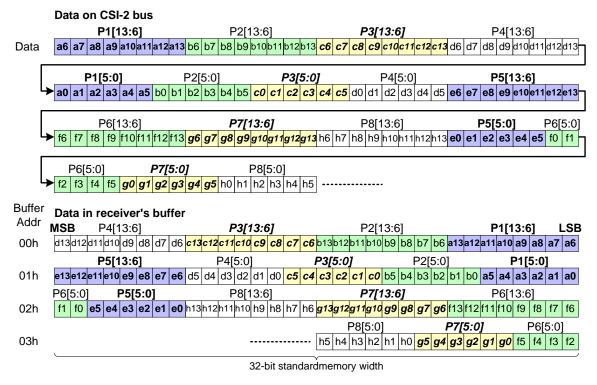


Figure 126 RAW 14 Data Format Reception

15761577

# 1566 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.
- 1570 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.
- 1575 The JPEG8 data flow is illustrated in the Figure 127 and Figure 128.

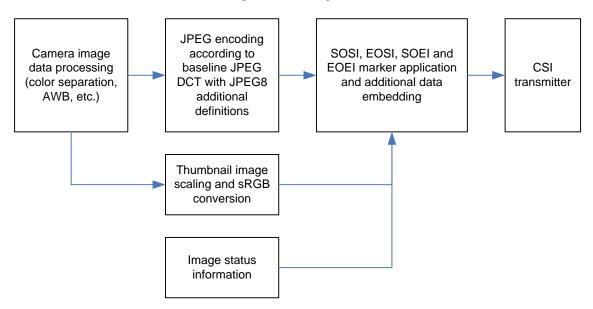
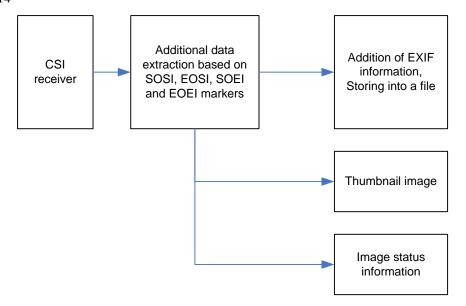


Figure 127 JPEG8 Data Flow in the Encoder



1580

1581 1582

1588

1589

1590

1591

Figure 128 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 129.
- 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.

Start of Image (SOI)
JFIF / EXIF Data
Quantization Table (DQT)
Huffman Table (DHT)
Frame Header (SOF)
Scan Header
Compressed Data
End Of Image (EOI)

15921593

1594

Figure 129 EXIF Compatible Baseline JPEG DCT Format

### A.3 Image Status Information

- 1595 Information of at least the following items has to be stored in the end of the JPEG sequence as illustrated in Figure 130:
- 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.
- 1605 for printing service. An example of register settings is following:
- Sample frequency
- **1607 ●** Exposure
- Analog and digital gain
- 1609 Gamma
- Color gamut conversion matrix
- 1611 Contrast
- 1612 Brightness
- 1613 Pre-gain

1617

1620

1614 The status information content has to be defined in the product specification of each camera module 1615 containing the JPEG8 feature. The format and content is manufacturer specific.

The image status data should be arranged so that each byte is split into two 4-bit nibbles and "1010" padding 1616

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. 1618

1619 The SOSI and EOSI markers are defined in Section A.5.

### **Table 28 Status Data Padding**

Data Word	After Padding
D7D6D5D4 D3D2D1D0	1010D7D6D5D4 1010D3D2D1D0

Start of Image (SOI)
JFIF / EXIF Data
Quantization Table (DQT)
Huffman Table (DHT)
Frame Header (SOF)
Scan Header
Compressed Data
End Of Image (EOI)
Start of Status Information (SOSI)
Image Status Information
End of Status Information (EOSI)

1621 1622

1623

### Figure 130 Status Information Field in the End of Baseline JPEG Frame

#### **A.4 Embedded Images**

- 1624 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 1625
- 1626 thumbnail image.
- 1627 The philosophy of embedded / interleaved thumbnail additions is to minimize the needed frame memory.
- 1628 The EI (Embedded Image) data can be included in any part of the compressed image data segment and in as
- 1629 many pieces as needed. See Figure 131.
- 1630 Embedded Image data is separated from compressed data by SOEI (Start Of Embedded Image) and EOEI
- (End Of Embedded Image) non-standard markers, which are defined in Section A.5. The amount of fields 1631
- separated by SOEI and EOEI is not limited. 1632

1640

1641

16471648

1649

- The pixel to byte packing for image data within an EI data field should be as specified for the equivalent CSI-
- 1634 2 data format. However there is an additional restriction; the embedded image data must not generate any
- false JPEG marker sequences (0xFFXX).
- The suggested method of preventing false JPEG marker codes from occurring within the embedded image data it to limit the data range for the pixel values. For example
- For RGB888 data the suggested way to solve the false synchronization code issue is to constrain the numerical range of R, G and B values from 1 to 254.
  - For RGB565 data the suggested way to solve the false synchronization code issue is to constrain the numerical range of G component from 1-62 and R component from 1-30.
- Each EI data field is separated by the SOEI / EOEI markers, and has to contain an equal amount bytes and a complete number of pixels. An EI data field may contain multiple lines or a full frame of image data.
- The embedded image data is decoded and removed apart from the JPEG compressed data prior to writing the JPEG into a file. In the process, EI data fields are appended one after each other, in order of occurrence in the received JPEG data.

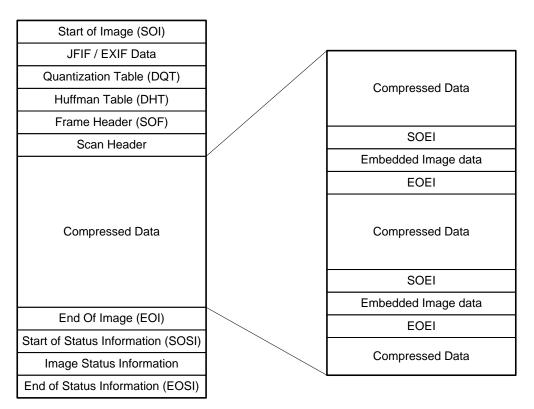


Figure 131 Example of TN Image Embedding Inside the Compressed JPEG Data Block

### A.5 JPEG8 Non-standard Markers

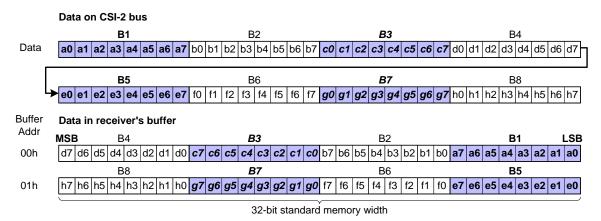
- JPEG8 uses the reserved JPEG data markers for special purposes, marking the additional segments inside the data file. These segments are not part of the JPEG, JFIF [0], EXIF [0] or any other specifications; instead their use is specified in this document in Section A.3 and Section A.4.
- The use of the non-standard markers is always internal to a product containing the JPEG8 camera module, and these markers are always removed from the JPEG data before storing it into a file.

**Table 29 JPEG8 Additional Marker Codes Listing** 

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

# 1656 A.6 JPEG8 Data Reception

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



16581659

Figure 132 JPEG8 Data Format Reception

1662

1663

1664

1665

1666 1667

1669

1676

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

### B.1 Overview

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

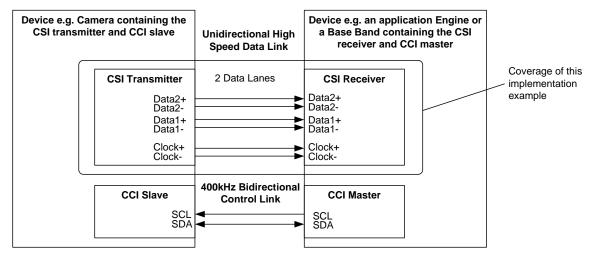


Figure 133 Implementation Example Block Diagram and Coverage

1668 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 134.

1680

1681

1682

1683

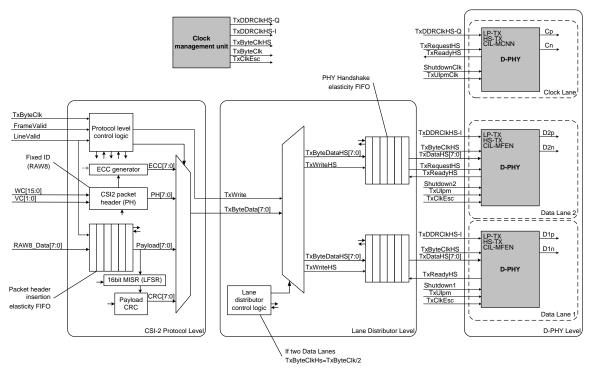
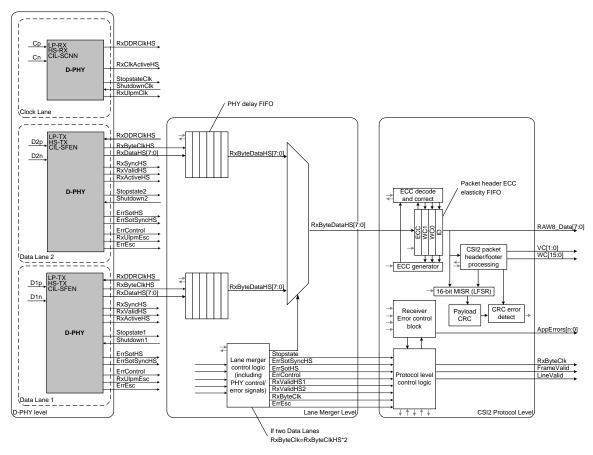


Figure 134 CSI-2 Transmitter Block Diagram

# 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 135.



1684 1685

1686

Figure 135 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 136.

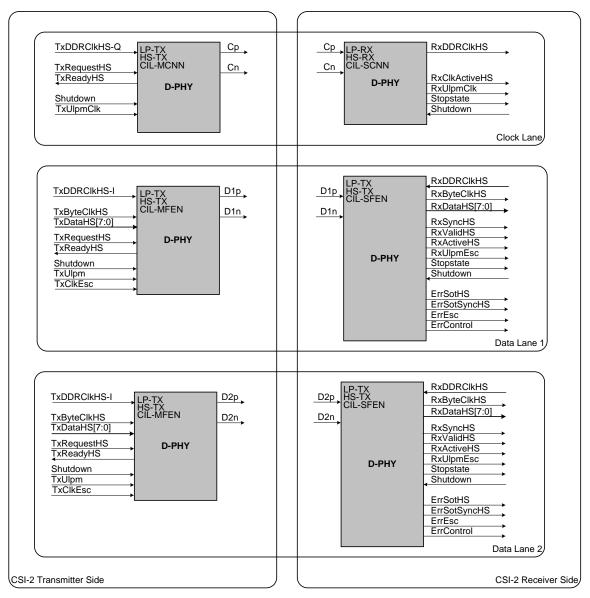


Figure 136 D-PHY Level Block Diagram

1690 The components can be categorized as:

• CSI-2 Transmitter side:

1688

1689

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

1701 1702

1704

1705

1706

1708

1709

17101711

1712

1713

1714

1715

1716 1717

1718

1719

1720

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

1700 The suggested implementation can be seen in Figure 137.

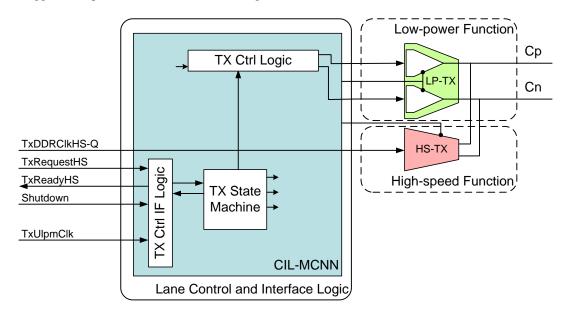


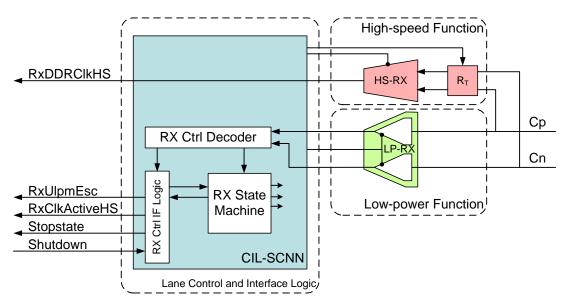
Figure 137 CSI-2 Clock Lane Transmitter

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

- **LP-TX** for the Low-power function
- **HS-TX** for the High-speed function
- **CIL-MCNN** for the Lane control and interface logic
- 1707 The PPI interface signals to the CSI-2 clock lane transmitter are:
  - TxDDRClkHS-Q (Input): High-Speed Transmit DDR Clock (Quadrature).
  - TxRequestHS (Input): High-Speed Transmit Request. This active high signal causes the lane module to begin transmitting a high-speed clock.
  - **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that the clock lane is transmitting HS clock.
  - Shutdown (Input): Shutdown Lane Module. This active high signal forces the lane module into "shutdown", disabling all activity. All line drivers, including terminators, are turned off when Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on any clock.
  - TxUlpmClk (Input): Transmit Ultra Low-Power mode on Clock Lane This active high signal is asserted to cause a Clock Lane module to enter the Ultra Low-Power mode. The lane module remains in this mode until TxUlpmClk is de-asserted.

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

1722 The suggested implementation can be seen in Figure 138.



1723

1724

1727

1730

1731

1732

1733

1734 1735

1736

1737

1738

1739

1740 1741

1742

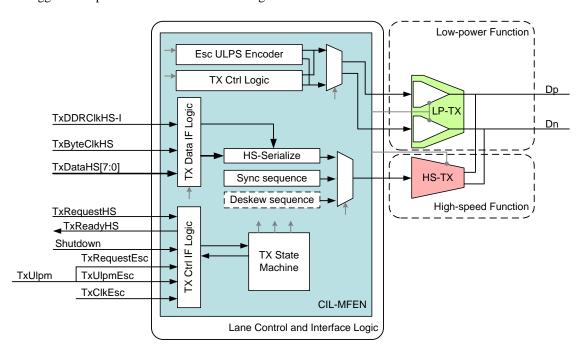
1743

Figure 138 CSI-2 Clock Lane Receiver

- 1725 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
- 1729 The PPI interface signals to the CSI-2 clock lane receiver are:
  - **RxDDRClkHS** (Output): High-Speed Receive DDR Clock used to sample the data in all data lanes.
  - **RxClkActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the clock lane is receiving valid clock. This signal is asynchronous.
  - **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is currently in Stop state. This signal is asynchronous.
  - Shutdown (Input): Shutdown Lane Module. This active high signal forces the lane module into "shutdown", disabling all activity. All line drivers, including terminators, are turned off when Shutdown is asserted. When Shutdown is high, all PPI outputs are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on any clock.
  - **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is asserted to indicate that the lane module has entered the ultra low power mode. The lane module remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane interconnect.

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

1745 The suggested implementation can be seen in Figure 139.



1746 1747 1748

1751

1752

1753

1754

1756

1757 1758

1759

1760

1761

1762 1763

1764

1765

1766 1767

1768

1769

Figure 139 CSI-2 Data Lane Transmitter

- The modular D-PHY components used to build a CSI-2 data lane transmitter are:
- **LP-TX** for the Low-power function
  - **HS-TX** for the High-speed function
    - **CIL-MFEN** for the Lane control and interface logic. For optional deskew calibration support, the data lane transmitter transmits a deskew sequence. The deskew sequence transmission is enabled by a mechanism out of the scope of this specification.
- 1755 The PPI interface signals to the CSI-2 data lane transmitter are:
  - TxDDRClkHS-I (Input): High-Speed Transmit DDR Clock (in-phase).
  - TxByteClkHS (Input): High-Speed Transmit Byte Clock. This is used to synchronize PPI signals in the high-speed transmit clock domain. It is recommended that both transmitting data lane modules share one TxByteClkHS signal. The frequency of TxByteClkHS must be exactly 1/8 the high-speed bit rate.
  - TxDataHS[7:0] (Input): High-Speed Transmit Data. Eight bit high-speed data to be transmitted. The signal connected to TxDataHS[0] is transmitted first. Data is registered on rising edges of TxByteClkHS.
  - TxRequestHS (Input): High-Speed Transmit Request. A low-to-high transition on TxRequestHS causes the lane module to initiate a Start-of-Transmission sequence. A high-to-low transition on TxRequest causes the lane module to initiate an End-of-Transmission sequence. This active high signal also indicates that the protocol is driving valid data on TxByteDataHS to be transmitted. The lane module accepts the data when both TxRequestHS and TxReadyHS are active on the same rising TxByteClkHS clock edge. The protocol always provides valid transmit data when

1776

1777

1778

1779 1780

1781

1782

1783

1784

1785

1786

1787

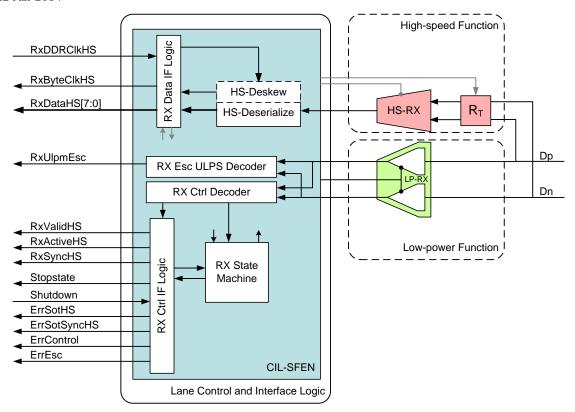
- TxRequestHS is active. Once asserted, TxRequestHS should remain high until the all the data has been accepted.
- TxReadyHS (Output): High-Speed Transmit Ready. This active high signal indicates that

  TxDataHS is accepted by the lane module to be serially transmitted. TxReadyHS is valid on rising
  edges of TxByteClkHS. Valid data has to be provided for the whole duration of active

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

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

1792 The suggested implementation can be seen in Figure 140.



1797

1798

1799

1800

1802

1803

Figure 140 CSI-2 Data Lane Receiver

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

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

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

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

1804 1805 1806

1807 1808 1809

1810 1811 1812

1813 1814

- **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)

1844	Annex C CSI-2 Recommended Receiver Error Behavior (informative)
1845	C.1 Overview
1846 1847 1848 1849 1850 1851 1852	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.
1853 1854 1855	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:
1856 1857	• <i>D-PHY Level errors</i> Refers to any PHY related transmission error and is unrelated to the transmission's contents:
1858	• Start of Transmission (SoT) errors, which can be:
1859	• Recoverable, if the PHY successfully identifies the Sync code but an error was detected.
1860 1861	<ul> <li>Unrecoverable, if the PHY does not successfully identify the sync code but does detect a HS transmission.</li> </ul>
1862 1863	• <i>Control Error</i> , which signals that the PHY has detected a control sequence that should not be present in this implementation of the Link.
1864 1865	<ul> <li>Packet Level errors         This type of error refers strictly to data integrity of the received Packet Header and payload data:     </li> </ul>
1866	• Packet Header errors, signaled through the ECC code, that result in:
1867	<ul> <li>A single bit-error, which can be detected and corrected by the ECC code</li> </ul>
1868 1869	<ul> <li>Two bit-errors in the header, which can be detected but not corrected by the ECC code, resulting in a corrupt header</li> </ul>
1870	<ul> <li>Packet payload errors, signaled through the CRC code</li> </ul>
1871 1872 1873	<ul> <li>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:     </li> </ul>
1874 1875	• Frame Sync Error, caused when a FS could not be successfully paired with a FE on a given virtual channel
1876 1877	• <i>Unrecognized ID</i> , caused by the presence of an unimplemented or unrecognized ID in the header
1878 1879	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

1881 C.2 D-PHY Level Error

1880

1883

The recommended behavior for handling this error level covers only those errors generated by the Data

should implement sequential behavior using a state machine for proper operation.

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

1888

1889

1890 1891

1892

1893

1894 1895

1896

1897

1898

1900

1901 1902

1903 1904

1905

1906

1907

1908

1909

- 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.
- 1912 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.
- 1915 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.
- Also, computing the CRC code over the whole payload of the received packet could generate the following errors:
- ErrCrc: Asserted when the computed CRC code is different than the received CRC code.
- **ErrID:** Asserted when a Packet Header is decoded with an unrecognized or unimplemented data ID.

1929

1930

1931

1932 1933

1934

1935

1936

1939

1940

1941

1942

1943

1944

1950

1951

1955

1956

19571958

1959

1960

1961

1962

1963

1964

1965 1966

1967 1968

1969

- 1928 The recommended receiver error behavior for this level is:
  - **ErrEccDouble** should be passed to the Application Layer since assertion of this signal proves that the Packet Header information is corrupt, and therefore the WC is not usable, and thus the packet end cannot be estimated. Commonly, this type of error will be accompanied with an ErrCrc. This type of error should also be passed to the Protocol Decoding Level, since the whole transmission until D-PHY Stop state should be ignored.
    - **ErrEccCorrected** should be passed to the Application Layer since the application should be informed that an error had occurred but was corrected, so the received Packet Header was unaffected, although the confidence in the data integrity is reduced.
- **ErrEccNoError** can be passed to the Protocol Decoding Level to signal the validity of the current Packet Header.
  - **ErrCrc** should be passed to the Protocol Decoding Level to indicate that the packet's payload data might be corrupt.
  - **ErrID** should be passed to the Application Layer to indicate that the data packet is unidentified and cannot be unpacked by the receiver. This signal should be asserted after the ID has been identified and de-asserted on the first Frame End (FE) on same virtual channel.

# C.4 Protocol Decoding Level Error

- The recommended behavior for this error level covers errors caused by decoding the Packet Header information and detecting a sequence that is not allowed by the CSI-2 protocol or a sequence of detected errors by the previous layers. CSI-2 implementers will commonly choose to implement this level of error handling using a state machine that should be paired with the corresponding virtual channel. The state machine should generate at least the following error signals:
  - **ErrFrameSync:** Asserted when a Frame End (FE) is not paired with a Frame Start (FS) on the same virtual channel. An ErrSotSyncHS should also generate this error signal.
- **ErrFrameData:** Asserted after a FE when the data payload received between FS and FE contains errors.
- 1954 The recommended receiver error behavior for this level is:
  - **ErrFrameSync** should be passed to the Application Layer with the corresponding virtual channel, since the frame could not be successfully identified. Several error cases on the same virtual channel can be identified for this type of error.
    - If a FS is followed by a second FS on the same virtual channel, the frame corresponding to the first FS is considered in error.
    - If a Packet Level ErrEccDouble was signaled from the Protocol Layer, the whole transmission until the first D-PHY Stop-state should be ignored since it contains no information that can be safely decoded and cannot be qualified with a data valid signal.
    - If a FE is followed by a second FE on the same virtual channel, the frame corresponding to the second FE is considered in error.
    - If an ErrSotSyncHS was signaled from the PHY Layer, the whole transmission until the first D-PHY Stop state should be ignored since it contains no information that can be safely decoded and cannot be qualified with a data valid signal.
  - **ErrFrameData**: should be passed to the Application Layer to indicate that the frame contains data errors. This signal should be asserted on any ErrCrc and de-asserted on the first FE.

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

### 1971 **D.1 Overview**

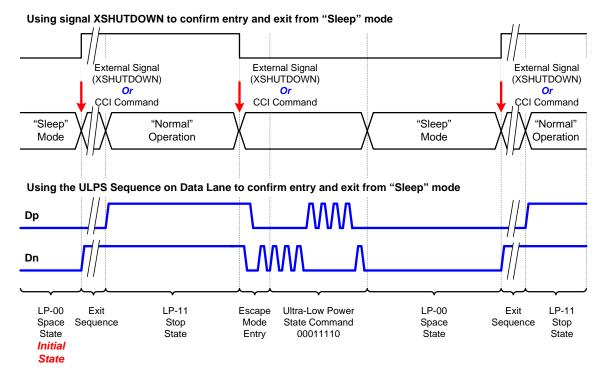
- 1972 Since a camera in a mobile terminal spends most of its time in an inactive state, implementers need a way to
- 1973 put the CSI-2 Link into a low power mode that approaches, or may be as low as, the leakage level. This
- section proposes one approach for putting a CSI-2 Link in a "Sleep Mode" (SLM). Although the section is
- informative and therefore does not affect compliance for CSI-2, the approach is offered by the MIPI Camera
- 1976 Working Group as a recommended approach.
- 1977 This approach relies on an aspect of a D-PHY transmitter's behavior that permits regulators to be disabled
- 1978 safely when LP-00 (Space state) is on the Link. Accordingly, this will be the output state for a CSI-2 camera
- transmitter in SLM.
- 1980 SLM can be thought of as a three-phase process:
- 1981 1. SLM Command Phase. The 'ENTER SLM' command is issued to the TX side only, or to both sides of the Link.
- 1983 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.
- 1987 In general, when in SLM, both sides of the interface will be in ULPS, as defined in [MIPI01].

### 1988 D.2 SLM Command Phase

- 1989 For the first phase, initiation of SLM occurs by a mechanism outside the scope of CSI-2. Of the many
- mechanisms available, two examples would be:
- 1. An External SLEEP signal input to the CSI-2 transmitter and optionally also to the CSI-2 Receiver.
- When at logic 0, the CSI-2 Transmitter and the CSI Receiver (if connected) will enter Sleep mode.
- When at logic 1, normal operation will take place.
- 1994 2. A CCI control command, provided on the I2C control Link, is used to trigger ULPS.

## 1995 **D.3 SLM Entry Phase**

- 1996 For the second phase, consider one option:
- 1997 Only the TX side enters SLM and propagates the ULPS to the RX side by sending a D-PHY 'ULPS'
- 1998 command on Clock Lane and on Data Lane(s). In the following picture only Data Lane 'ULPS' command is
- used as an example.



2002

2003

2004

2005

2006

2007

2008

2009

2010

2011

2012

Figure 141 SLM Synchronization

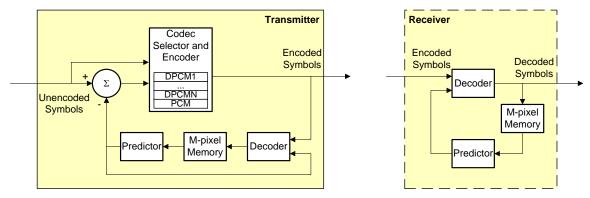
### D.4 SLM Exit Phase

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

- 1. Use a SLEEP signal to power-up both sides of the interface.
- 2. Detect any CCI activity on the I2C control Link, which was in the 00 state ({SCL, SDA}), after receiving the I2C instruction to enter ULPS command as per Section D.2, option 2. Any change on those lines should wake up the camera peripheral. The drawback of this method is that I2C lines are used exclusively for control of the camera.
- 3. Detect a wake-up sequence on the I2C lines. This sequence, which may vary by implementation, shall not disturb the I2C interface so that it can be used by other devices. One example sequence is: StopI2C-StartI2C-StopI2C. See Section 6 for details on CCI.
- A handshake using the 'ULPS' mechanism in the as described in [MIPI01] should be used for powering up the interface.

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

- 2016 A CSI-2 implementation using RAW data types may support compression on the interface to reduce the data
- 2017 bandwidth requirements between the host processor and a camera module. Data compression is not mandated
- by this Specification. However, if data compression is used, it shall be implemented as described in this
- 2019 annex.
- 2020 Data compression schemes use an X–Y–Z naming convention where X is the number of bits per pixel in the
- original image, Y is the encoded (compressed) bits per pixel and Z is the decoded (uncompressed) bits per
- 2022 pixel.
- 2023 The following data compression schemes are defined:
- 2024 12–8–12
- 2025 12–7–12
- 2026 12–6–12
- 2027 10-8-10
- 2028 10–7–10
- 2029 10-6-10
- 2030 To identify the type of data on the CSI-2 interface, packets with compressed data shall have a User Defined
- 2031 Data Type value as indicated in Table 27. Note that User Defined data type codes are not reserved for
- 2032 compressed data types. Therefore, a CSI-2 device shall be able to communicate over the CCI the data
- 2033 compression scheme represented by a particular User Defined data type code for each scheme supported by
- 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
- 2040 code.
- The data compression schemes in this annex are lossy and designed to encode each line independent of the
- other lines in the image.
- 2043 The following definitions are used in the description of the data compression schemes:
- **Xorig** is the original pixel value
- **Xpred** is the predicted pixel value
- **Xdiff** is the difference value (**Xorig Xpred**)
- **Xenco** is the encoded value
- **Xdeco** is the decoded pixel value
- The data compression system consists of encoder, decoder and predictor blocks as shown in Figure 142.



2052

2053

2054

2055

2056

2057

2058

2059

2060

2061

2062

2063

Figure 142 Data Compression System Block Diagram

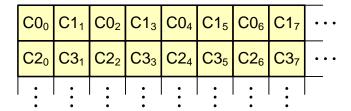
The encoder uses a simple algorithm to encode the pixel values. A fixed number of pixel values at the beginning of each line are encoded without using prediction. These first few values are used to initialize the predictor block. The remaining pixel values on the line are encoded using prediction.

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

### E.1 Predictors

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

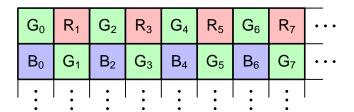
The order of pixels in a raw image is shown in Figure 143.



20642065

Figure 143 Pixel Order of the Original Image

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



20672068

Figure 144 Example Pixel Order of the Original Image

2069 Two predictors are defined for use in the data compression schemes.

- 2070 Predictor1 uses a very simple algorithm and is intended to minimize processing power and memory size
- 2071 requirements. Typically, this predictor is used when the compression requirements are modest and the
- 2072 original image quality is high. Predictor1 should be used with 10-8-10, 10-7-10 and 12-8-12 data
- 2073 compression schemes.
- 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.
- 2076 Predictor2 should be used with 10–6–10, 12–7–12, and 12–6–12 data compression schemes.
- 2077 Both receiver and transmitter shall support Predictor1 for all data compression schemes.

# 2078 **E.1.1 Predictor1**

- 2079 Predictor1 uses only the previous same color component value as the prediction value. Therefore, only a
- 2080 two-pixel deep memory is required.
- The first two pixels  $(C0_0, C1_1 / C2_0, C3_1 \text{ or as in example } G_0, R_1 / B_0, G_1)$  in a line are encoded without
- 2082 prediction.
- 2083 The prediction values for the remaining pixels in the line are calculated using the previous same color
- decoded value, **Xdeco**. Therefore, the predictor equation can be written as follows:
- 2085  $\mathbf{Xpred(n)} = \mathbf{Xdeco(n-2)}$

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

- 2087 Predictor2 uses the four previous pixel values, when the prediction value is evaluated. This means that also
- 2088 the other color component values are used, when the prediction value has been defined. The predictor
- 2089 equations can be written as shown in the following formulas.
- 2090 Predictor2 uses all color components of the four previous pixel values to create the prediction value.
- 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
- 2094 different color value as a prediction value. The second pixel is predicted with the following equation:
- 2095  $\mathbf{Xpred}(\mathbf{n}) = \mathbf{Xdeco}(\mathbf{n-1})$
- The third pixel  $(C0_2 / C2_2)$  or as in example  $G_2 / G_2$  in a line is predicted using the previous decoded same
- 2097 color value as a prediction value. The third pixel is predicted with the following equation:
- 2098  $\mathbf{Xpred}(\mathbf{n}) = \mathbf{Xdeco}(\mathbf{n-2})$
- The fourth pixel  $(C1_3 / C3_3)$  or as in example  $R_3 / G_3$  in a line is predicted using the following equation:

2105 endif

```
Version 1.2
22 Jan-2014
```

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

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

### 2116 E.2 Encoders

- There are six different encoders available, one for each data compression scheme.
- 2118 For all encoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
- 2119 for predicted pixels.

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

- The 10–8–10 coder offers a 20% bit rate reduction with very high image quality.
- 2122 Pixels without prediction are encoded using the following formula:

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

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

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

2128 Pixels with prediction are encoded using the following formula:

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

### 2138 E.2.1.1 DPCM1 for 10-8-10 Coder

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

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

```
22 Jan-2014
2141
                 where,
2142
                     "00" is the code word
                     "s" is the sign bit
2143
                     "xxxxx" is the five bit value field
2144
2145
         The coder equation is described as follows:
2146
                 if (Xdiff(n) \le 0) then
2147
                     sign = 1
2148
                 else
                     sign = 0
2149
                 endif
2150
                 value = abs(Xdiff( n ))
2151
2152
         Note: Zero code has been avoided (0 is sent as -0).
                      DPCM2 for 10-8-10 Coder
2153
         E.2.1.2
2154
         Xenco( n ) has the following format:
                 Xenco( n ) = "010 \text{ s } xxxx"
2155
2156
                 where,
2157
                     "010" is the code word
2158
                     "s" is the sign bit
                     "xxxx" is the four bit value field
2159
2160
         The coder equation is described as follows:
2161
                 if (Xdiff(n) < 0) then
2162
                     sign = 1
2163
                 else
2164
                     sign = 0
2165
                 endif
                 value = (abs(Xdiff(n)) - 32) / 2
2166
2167
         E.2.1.3
2168
                      DPCM3 for 10-8-10 Coder
2169
         Xenco( n ) has the following format:
2170
                 Xenco(n) = "011 s xxxx"
2171
                 where,
2172
                     "011" is the code word
                     "s" is the sign bit
2173
                     "xxxx" is the four bit value field
2174
```

```
2176
        The coder equation is described as follows:
2177
                if (Xdiff(n) < 0) then
                    sign = 1
2178
2179
                else
2180
                    sign = 0
2181
                endif
2182
                value = (abs(Xdiff(n)) - 64) / 4
        E.2.1.4
2183
                     PCM for 10-8-10 Coder
2184
        Xenco( n ) has the following format:
                Xenco( n ) = "1 xxxxxxx"
2185
2186
                where,
2187
                    "1" is the code word
                    the sign bit is not used
2188
2189
                    "xxxxxxx" is the seven bit value field
2190
        The coder equation is described as follows:
2191
                value = Xorig(n)/8
        E.2.2
                    Coder for 10-7-10 Data Compression
2192
2193
        The 10–7–10 coder offers 30% bit rate reduction with high image quality.
2194
        Pixels without prediction are encoded using the following formula:
2195
                Xenco(n) = Xorig(n)/8
2196
        To avoid a full-zero encoded value, the following check is performed:
                if (Xenco(n) == 0) then
2197
                    Xenco(n) = 1
2198
2199
        Pixels with prediction are encoded using the following formula:
                if (abs(Xdiff(n)) < 8) then
2200
2201
                    use DPCM1
2202
                else if (abs(Xdiff(n)) < 16) then
2203
                    use DPCM2
                else if (abs(Xdiff(n)) < 32) then
2204
                    use DPCM3
2205
2206
                else if (abs(Xdiff( n )) < 160) then
2207
                    use DPCM4
2208
                else
                    use PCM
2209
2210
                endif
```

```
2211
         E.2.2.1
                      DPCM1 for 10-7-10 Coder
2212
         Xenco( n ) has the following format:
2213
                 Xenco( n ) = "000 \text{ s } xxx"
2214
                 where,
2215
                     "000" is the code word
2216
                     "s" is the sign bit
                     "xxx" is the three bit value field
2217
2218
         The coder equation is described as follows:
2219
                 if (Xdiff(n) \le 0) then
2220
                     sign = 1
2221
                 else
2222
                     sign = 0
2223
                 endif
2224
                 value = abs(Xdiff( n ))
2225
         Note: Zero code has been avoided (0 is sent as -0).
         E.2.2.2
                      DPCM2 for 10-7-10 Coder
2226
2227
         Xenco( n ) has the following format:
                 Xenco( n ) = "0010 \text{ s xx}"
2228
2229
                 where,
2230
                     "0010" is the code word
2231
                     "s" is the sign bit
2232
                     "xx" is the two bit value field
2233
         The coder equation is described as follows:
                 if (Xdiff(n) < 0) then
2234
2235
                     sign = 1
2236
                 else
2237
                     sign = 0
2238
                 endif
2239
                 value = (abs(Xdiff(n)) - 8) / 2
2240
         E.2.2.3
                      DPCM3 for 10-7-10 Coder
2241
         Xenco( n ) has the following format:
                 Xenco( n ) = "0011 \text{ s } xx"
2242
2243
                 where,
                     "0011" is the code word
2244
2245
                     "s" is the sign bit
                     "xx" is the two bit value field
2246
```

```
Version 1.2
         22 Jan-2014
2247
         The coder equation is described as follows:
2248
                 if (Xdiff(n) < 0) then
                      sign = 1
2249
2250
                 else
2251
                     sign = 0
2252
                 endif
2253
                 value = (abs(Xdiff(n)) - 16) / 4
         E.2.2.4
2254
                       DPCM4 for 10-7-10 Coder
2255
         Xenco( n ) has the following format:
                 Xenco( n ) = "01 \text{ s } xxxx"
2256
2257
                 where,
2258
                     "01" is the code word
                     "s" is the sign bit
2259
2260
                     "xxxx" is the four bit value field
2261
         The coder equation is described as follows:
2262
                 if (Xdiff(n) < 0) then
2263
                     sign = 1
2264
                 else
2265
                     sign = 0
2266
                 endif
                 value = (abs(Xdiff(n)) - 32) / 8
2267
         E.2.2.5
                      PCM for 10-7-10 Coder
2268
2269
         Xenco( n ) has the following format:
2270
                 Xenco( n ) = "1 \times x \times x \times x"
2271
                 where.
2272
                     "1" is the code word
2273
                     the sign bit is not used
2274
                      "xxxxxx" is the six bit value field
2275
         The coder equation is described as follows:
                 value = Xorig(n) / 16
2276
```

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

- 2278 The 10–6–10 coder offers 40% bit rate reduction with acceptable image quality.
- 2279 Pixels without prediction are encoded using the following formula:
- 2280 Xenco(n) = Xorig(n) / 16

```
2281
        To avoid a full-zero encoded value, the following check is performed:
2282
                if (Xenco(n) == 0) then
                     Xenco(n) = 1
2283
                endif
2284
2285
        Pixels with prediction are encoded using the following formula:
2286
                if (abs(Xdiff(n)) < 1) then
2287
                     use DPCM1
2288
                else if (abs(Xdiff(n)) < 3) then
                     use DPCM2
2289
                else if (abs(Xdiff(n)) < 11) then
2290
                     use DPCM3
2291
                else if (abs(Xdiff(n)) < 43) then
2292
                     use DPCM4
2293
                else if (abs(Xdiff( n )) < 171) then
2294
                     use DPCM5
2295
2296
                else
                     use PCM
2297
2298
                endif
        E.2.3.1
                     DPCM1 for 10-6-10 Coder
2299
2300
        Xenco( n ) has the following format:
2301
                Xenco( n ) = "00000 \text{ s}"
2302
                where,
2303
                     "00000" is the code word
                     "s" is the sign bit
2304
                     the value field is not used
2305
2306
        The coder equation is described as follows:
2307
                sign = 1
2308
        Note: Zero code has been avoided (0 is sent as -0).
2309
        E.2.3.2
                      DPCM2 for 10-6-10 Coder
2310
        Xenco( n ) has the following format:
2311
                Xenco( n ) = "00001 \text{ s}"
2312
                where.
2313
                     "00001" is the code word
2314
                     "s" is the sign bit
2315
                     the value field is not used
2316
```

```
22 Jan-2014
2317
        The coder equation is described as follows:
2318
                 if (Xdiff(n) < 0) then
                     sign = 1
2319
2320
                 else
                     sign = 0
2321
                 endif
2322
2323
        E.2.3.3
                      DPCM3 for 10-6-10 Coder
2324
        Xenco( n ) has the following format:
                 Xenco( n ) = "0001 \text{ s x}"
2325
2326
                 where,
2327
                     "0001" is the code word
2328
                     "s" is the sign bit
                     "x" is the one bit value field
2329
2330
        The coder equation is described as follows:
2331
                 if (Xdiff(n) < 0) then
2332
                     sign = 1
2333
                 else
2334
                     sign = 0
2335
                 value = (abs(Xdiff(n)) - 3) / 4
2336
                 endif
        E.2.3.4
2337
                      DPCM4 for 10-6-10 Coder
2338
        Xenco( n ) has the following format:
2339
                 Xenco( n ) = "001 \text{ s } xx"
2340
                 where,
2341
                     "001" is the code word
2342
                     "s" is the sign bit
2343
                     "xx" is the two bit value field
2344
        The coder equation is described as follows:
2345
                 if (Xdiff(n) < 0) then
                     sign = 1
2346
2347
                 else
2348
                     sign = 0
2349
                 endif
2350
                 value = (abs(Xdiff(n)) - 11) / 8
2351
        E.2.3.5
                      DPCM5 for 10-6-10 Coder
2352
        Xenco( n ) has the following format:
```

**Xenco**( n ) = "01 s xxx"

2353

```
22 Jan-2014
2354
                 where,
2355
                     "01" is the code word
                     "s" is the sign bit
2356
                     "xxx" is the three bit value field
2357
2358
         The coder equation is described as follows:
2359
                 if (Xdiff(n) < 0) then
2360
                     sign = 1
2361
                 else
                     sign = 0
2362
2363
                 endif
                 value = (abs(Xdiff(n)) - 43) / 16
2364
2365
         E.2.3.6
                      PCM for 10–6–10 Coder
2366
        Xenco( n ) has the following format:
2367
                 Xenco( n ) = "1 \times x \times x \times"
2368
                 where,
2369
                     "1" is the code word
2370
                     the sign bit is not used
                     "xxxxx" is the five bit value field
2371
2372
         The coder equation is described as follows:
2373
                 value = Xorig(n) / 32
2374
         E.2.4
                    Coder for 12–8–12 Data Compression
2375
         The 12–8–12 coder offers 33% bit rate reduction with very high image quality.
2376
        Pixels without prediction are encoded using the following formula:
2377
                 Xenco(n) = Xorig(n) / 16
         To avoid a full-zero encoded value, the following check is performed:
2378
                 if (Xenco(n) == 0) then
2379
2380
                     Xenco(n) = 1
2381
                 endif
2382
         Pixels with prediction are encoded using the following formula:
                 if (abs(Xdiff(n)) < 8) then
2383
2384
                     use DPCM1
2385
                 else if (abs(Xdiff(n)) < 40) then
2386
                     use DPCM2
2387
                 else if (abs(Xdiff( n )) < 104) then
2388
                     use DPCM3
2389
                 else if (abs(Xdiff( n )) < 232) then
                     use DPCM4
2390
```

```
22 Jan-2014
2391
                 else if (abs(Xdiff(n)) < 360) then
2392
                     use DPCM5
2393
                 else
                     use PCM
2394
2395
         E.2.4.1
                      DPCM1 for 12-8-12 Coder
2396
        Xenco( n ) has the following format:
2397
                 Xenco( n ) = "0000 \text{ s xxx}"
2398
                 where.
2399
                     "0000" is the code word
2400
                     "s" is the sign bit
                     "xxx" is the three bit value field
2401
2402
        The coder equation is described as follows:
2403
                 if (Xdiff(n) \le 0) then
                     sign = 1
2404
2405
                 else
2406
                     sign = 0
2407
                 endif
                 value = abs(Xdiff( n ))
2408
2409
        Note: Zero code has been avoided (0 is sent as -0).
2410
        E.2.4.2
                      DPCM2 for 12-8-12 Coder
2411
        Xenco( n ) has the following format:
2412
                 Xenco( n ) = "011 s xxxx"
2413
                 where,
2414
                     "011" is the code word
                     "s" is the sign bit
2415
                     "xxxx" is the four bit value field
2416
2417
        The coder equation is described as follows:
2418
                 if (Xdiff(n) < 0) then
2419
                     sign = 1
2420
                 else
                     sign = 0
2421
2422
                 endif
2423
                 value = (abs(Xdiff(n)) - 8) / 2
2424
        E.2.4.3
                      DPCM3 for 12-8-12 Coder
2425
        Xenco( n ) has the following format:
                 Xenco( n ) = "010 \text{ s } xxxx"
2426
```

```
22 Jan-2014
2427
                 where,
2428
                     "010" is the code word
                     "s" is the sign bit
2429
                     "xxxx" is the four bit value field
2430
2431
         The coder equation is described as follows:
                 if (Xdiff(n) < 0) then
2432
2433
                     sign = 1
2434
                 else
                     sign = 0
2435
                 endif
2436
                 value = (abs(Xdiff(n)) - 40) / 4
2437
2438
         E.2.4.4
                      DPCM4 for 12–8–12 Coder
2439
         Xenco( n ) has the following format:
2440
                 Xenco( n ) = "001 \text{ s } xxxx"
2441
                 where,
2442
                     "001" is the code word
2443
                     "s" is the sign bit
                     "xxxx" is the four bit value field
2444
2445
         The coder equation is described as follows:
                 if (Xdiff(n) < 0) then
2446
2447
                     sign = 1
2448
                 else
2449
                     sign = 0
2450
                 endif
2451
                 value = (abs(Xdiff(n)) - 104) / 8
2452
         E.2.4.5
                      DPCM5 for 12-8-12 Coder
2453
         Xenco( n ) has the following format:
2454
                 Xenco( n ) = "0001 \text{ s } xxx"
2455
                 where,
                     "0001" is the code word
2456
                     "s" is the sign bit
2457
                     "xxx" is the three bit value field
2458
2459
         The coder equation is described as follows:
                 if (Xdiff(n) < 0) then
2460
2461
                     sign = 1
2462
                 else
2463
                     sign = 0
```

endif

2464

```
2465
                value = (abs(Xdiff(n)) - 232) / 16
2466
        E.2.4.6
                      PCM for 12-8-12 Coder
2467
        Xenco( n ) has the following format:
2468
                Xenco( n ) = "1 \times x \times x \times x \times x"
2469
                where,
                     "1" is the code word
2470
                     the sign bit is not used
2471
                     "xxxxxxx" is the seven bit value field
2472
2473
        The coder equation is described as follows:
2474
                value = Xorig(n)/32
        E.2.5
2475
                    Coder for 12–7–12 Data Compression
2476
        The 12–7–12 coder offers 42% bit rate reduction with high image quality.
2477
        Pixels without prediction are encoded using the following formula:
2478
                Xenco(n) = Xorig(n)/32
2479
         To avoid a full-zero encoded value, the following check is performed:
2480
                if (Xenco(n) == 0) then
                     Xenco(n) = 1
2481
2482
                endif
        Pixels with prediction are encoded using the following formula:
2483
2484
                if (abs(Xdiff(n)) < 4) then
2485
                     use DPCM1
                else if (abs(Xdiff(n)) < 12) then
2486
2487
                     use DPCM2
2488
                else if (abs(Xdiff(n)) < 28) then
                     use DPCM3
2489
                else if (abs(Xdiff(n)) < 92) then
2490
                     use DPCM4
2491
                else if (abs(Xdiff( n )) < 220) then
2492
                     use DPCM5
2493
2494
                else if (abs(Xdiff(n)) < 348) then
                     use DPCM6
2495
2496
                else
                     use PCM
2497
                endif
2498
        E.2.5.1
                      DPCM1 for 12-7-12 Coder
2499
2500
        Xenco( n ) has the following format:
2501
                Xenco( n ) = "0000 \text{ s xx}"
```

```
22 Jan-2014
2502
                 where,
2503
                     "0000" is the code word
                     "s" is the sign bit
2504
                     "xx" is the two bit value field
2505
2506
         The coder equation is described as follows:
2507
                 if (Xdiff(n) \le 0) then
2508
                     sign = 1
2509
                 else
                     sign = 0
2510
                 endif
2511
                 value = abs(Xdiff( n ))
2512
2513
         Note: Zero code has been avoided (0 is sent as -0).
2514
         E.2.5.2
                      DPCM2 for 12-7-12 Coder
2515
         Xenco( n ) has the following format:
                 Xenco( n ) = "0001 \text{ s xx}"
2516
2517
                 where,
2518
                     "0001" is the code word
2519
                     "s" is the sign bit
                     "xx" is the two bit value field
2520
2521
         The coder equation is described as follows:
                 if (Xdiff(n) < 0) then
2522
2523
                     sign = 1
2524
                 else
2525
                     sign = 0
2526
                 endif
                 value = (abs(Xdiff(n)) - 4) / 2
2527
         E.2.5.3
                      DPCM3 for 12-7-12 Coder
2528
2529
        Xenco(n) has the following format:
2530
                 Xenco( n ) = "0010 \text{ s xx}"
2531
                 where,
2532
                     "0010" is the code word
2533
                     "s" is the sign bit
                     "xx" is the two bit value field
2534
2535
         The coder equation is described as follows:
2536
                 if (Xdiff(n) < 0) then
2537
                     sign = 1
2538
                 else
```

```
22 Jan-2014
2539
                     sign = 0
2540
                 endif
                 value = (abs(Xdiff(n)) - 12) / 4
2541
         E.2.5.4
                      DPCM4 for 12-7-12 Coder
2542
2543
         Xenco( n ) has the following format:
2544
                 Xenco( n ) = "010 \text{ s } xxx"
2545
                 where,
2546
                     "010" is the code word
                     "s" is the sign bit
2547
2548
                     "xxx" is the three bit value field
2549
         The coder equation is described as follows:
                 if (Xdiff(n) < 0) then
2550
2551
                     sign = 1
2552
                 else
2553
                     sign = 0
2554
                 endif
2555
                 value = (abs(Xdiff(n)) - 28) / 8
2556
         E.2.5.5
                      DPCM5 for 12-7-12 Coder
2557
         Xenco(n) has the following format:
                 Xenco( n ) = "011 \text{ s } xxx"
2558
2559
                 where,
2560
                     "011" is the code word
2561
                     "s" is the sign bit
2562
                     "xxx" is the three bit value field
2563
         The coder equation is described as follows:
                 if (Xdiff(n) < 0) then
2564
2565
                     sign = 1
2566
                 else
2567
                     sign = 0
                 endif
2568
2569
                 value = (abs(Xdiff(n)) - 92) / 16
         E.2.5.6
                      DPCM6 for 12-7-12 Coder
2570
2571
        Xenco( n ) has the following format:
                 Xenco( n ) = "0011 \text{ s xx}"
2572
```

```
22 Jan-2014
2573
                 where,
2574
                     "0011" is the code word
                     "s" is the sign bit
2575
                     "xx" is the two bit value field
2576
2577
         The coder equation is described as follows:
2578
                 if (Xdiff(n) < 0) then
2579
                     sign = 1
2580
                 else
                     sign = 0
2581
2582
                 endif
                 value = (abs(Xdiff(n)) - 220) / 32
2583
2584
        E.2.5.7
                      PCM for 12–7–12 Coder
2585
        Xenco( n ) has the following format:
2586
                 Xenco( n ) = "1 \times x \times x \times x"
2587
                 where,
2588
                     "1" is the code word
2589
                     the sign bit is not used
2590
                     "xxxxxx" is the six bit value field
2591
         The coder equation is described as follows:
2592
                 value = Xorig(n) / 64
2593
         E.2.6
                    Coder for 12–6–12 Data Compression
2594
         The 12–6–12 coder offers 50% bit rate reduction with acceptable image quality.
2595
         Pixels without prediction are encoded using the following formula:
2596
                 Xenco(n) = Xorig(n) / 64
         To avoid a full-zero encoded value, the following check is performed:
2597
                 if (Xenco(n) == 0) then
2598
2599
                     Xenco(n) = 1
2600
                 endif
2601
        Pixels with prediction are encoded using the following formula:
                 if (abs(Xdiff(n)) < 2) then
2602
                     use DPCM1
2603
                 else if (abs(Xdiff(n)) < 10) then
2604
                     use DPCM3
2605
2606
                 else if (abs(Xdiff(n)) < 42) then
                     use DPCM4
2607
                 else if (abs(Xdiff(n)) < 74) then
2608
```

```
22 Jan-2014
2609
                     use DPCM5
2610
                else if (abs(Xdiff( n )) < 202) then
2611
                     use DPCM6
2612
                else if (abs(Xdiff( n )) < 330) then
                     use DPCM7
2613
2614
                else
                     use PCM
2615
2616
                endif
2617
        Note: DPCM2 is not used.
2618
        E.2.6.1
                      DPCM1 for 12-6-12 Coder
        Xenco( n ) has the following format:
2619
2620
                Xenco( n ) = "0000 s x"
2621
                where,
                     "0000" is the code word
2622
                     "s" is the sign bit
2623
                     "x" is the one bit value field
2624
2625
        The coder equation is described as follows:
                if (Xdiff(n) \le 0) then
2626
                     sign = 1
2627
2628
                else
                     sign = 0
2629
2630
                endif
                value = abs(Xdiff( n ))
2631
2632
        Note: Zero code has been avoided (0 is sent as -0).
        E.2.6.2
                      DPCM3 for 12-6-12 Coder
2633
2634
        Xenco( n ) has the following format:
2635
                Xenco( n ) = "0001 \text{ s x}"
2636
                where,
2637
                     "0001" is the code word
2638
                     "s" is the sign bit
                     "x" is the one bit value field
2639
2640
        The coder equation is described as follows:
                if (Xdiff(n) < 0) then
2641
2642
                     sign = 1
2643
                else
2644
                     sign = 0
2645
                endif
                value = (abs(Xdiff(n)) - 2) / 4
2646
```

```
2647
         E.2.6.3
                      DPCM4 for 12-6-12 Coder
2648
         Xenco( n ) has the following format:
2649
                 Xenco( n ) = "010 s xx"
2650
                 where,
2651
                     "010" is the code word
2652
                     "s" is the sign bit
                     "xx" is the two bit value field
2653
2654
         The coder equation is described as follows:
2655
                 if (Xdiff(n) < 0) then
                     sign = 1
2656
2657
                 else
                     sign = 0
2658
2659
                 endif
                 value = (abs(Xdiff(n)) - 10) / 8
2660
2661
         E.2.6.4
                      DPCM5 for 12–6–12 Coder
2662
         Xenco( n ) has the following format:
2663
                 Xenco( n ) = "0010 \text{ s x}"
2664
                 where,
                     "0010" is the code word
2665
2666
                     "s" is the sign bit
                     "x" is the one bit value field
2667
2668
         The coder equation is described as follows:
                 if (Xdiff(n) < 0) then
2669
2670
                     sign = 1
2671
                 else
2672
                     sign = 0
                 endif
2673
                 value = (abs(Xdiff(n)) - 42) / 16
2674
         E.2.6.5
                      DPCM6 for 12-6-12 Coder
2675
         Xenco( n ) has the following format:
2676
2677
                 Xenco( n ) = "011 s xx"
2678
                 where,
2679
                     "011" is the code word
2680
                     "s" is the sign bit
2681
                     "xx" is the two bit value field
```

```
Version 1.2
         22 Jan-2014
2682
        The coder equation is described as follows:
2683
                if (Xdiff(n) < 0) then
                     sign = 1
2684
2685
                else
2686
                     sign = 0
2687
                endif
2688
                value = (abs(Xdiff(n)) - 74) / 32
         E.2.6.6
2689
                      DPCM7 for 12–6–12 Coder
2690
        Xenco( n ) has the following format:
                Xenco(n) = "0011 s x"
2691
2692
                 where,
2693
                     "0011" is the code word
2694
                     "s" is the sign bit
2695
                     "x" is the one bit value field
2696
        The coder equation is described as follows:
2697
                if (Xdiff(n) < 0) then
2698
                     sign = 1
2699
                else
2700
                     sign = 0
2701
                endif
                value = (abs(Xdiff(n)) - 202) / 64
2702
        E.2.6.7
                      PCM for 12-6-12 Coder
2703
2704
        Xenco(n) has the following format:
2705
                Xenco( n ) = "1 xxxxx"
2706
                where,
2707
                     "1" is the code word
2708
                     the sign bit is not used
2709
                     "xxxxx" is the five bit value field
2710
        The coder equation is described as follows:
2711
                value = Xorig(n) / 128
```

- 2712 E.3 Decoders
- 2713 There are six different decoders available, one for each data compression scheme.
- For all decoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
- 2715 for predicted pixels.

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

2717 Pixels without prediction are decoded using the following formula:

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

2719 Pixels with prediction are decoded using the following formula:

```
if (Xenco(\mathbf{n}) & 0xc0 == 0x00) then
2720
                     use DPCM1
2721
2722
                 else if (Xenco(\mathbf{n}) & 0xe0 == 0x40) then
                     use DPCM2
2723
                 else if (Xenco(\mathbf{n}) & 0xe0 == 0x60) then
2724
2725
                     use DPCM3
2726
                 else
                     use PCM
2727
2728
                 endif
2729
```

#### 2730 E.3.1.1 DPCM1 for 10-8-10 Decoder

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

```
2732 Xenco( n ) = "00 s xxxxx"
```

where,

2751

```
2734 "00" is the code word
2735 "s" is the sign bit
2736 "www." is the five bit value fi
```

2736 "xxxxx" is the five bit **value** field

2737 The decoder equation is described as follows:

```
      2738
      sign = Xenco( n ) & 0x20

      2739
      value = Xenco( n ) & 0x1f

      2740
      if (sign > 0) then

      2741
      Xdeco( n ) = Xpred( n ) - value

      2742
      else

      2743
      Xdeco( n ) = Xpred( n ) + value

      2744
      endif
```

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

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

```
2747 Xenco(n) = "010 s xxxx"
2748 where,
2749 "010" is the code word
2750 "s" is the sign bit
```

"xxxx" is the four bit value field

Version 1.2 22 Jan-2014 2752 The decoder equation is described as follows: 2753 sign = Xenco(n) & 0x10value = 2 \* (Xenco(n) & 0xf) + 322754 if (sign > 0) then 2755 Xdeco(n) = Xpred(n) - value2756 2757 else 2758 Xdeco(n) = Xpred(n) + valueendif 2759 2760 E.3.1.3 DPCM3 for 10-8-10 Decoder 2761 **Xenco( n )** has the following format: **Xenco( n )** = "011 s xxxx" 2762 2763 where, 2764 "011" is the code word 2765 "s" is the **sign** bit "xxxx" is the four bit value field 2766 2767 The decoder equation is described as follows: 2768 sign = Xenco(n) & 0x10value = 4 \* (Xenco(n) & 0xf) + 64 + 12769 2770 if (sign > 0) then Xdeco(n) = Xpred(n) - value 2771 if (Xdeco(n) < 0) then 2772 Xdeco(n) = 02773 2774 endif 2775 else Xdeco(n) = Xpred(n) + value2776 if (Xdeco(n) > 1023) then 2777 2778 **Xdeco**(n) = 1023 2779 endif endif 2780 2781 E.3.1.4 PCM for 10-8-10 Decoder 2782 **Xenco**(**n**) has the following format: 2783 **Xenco( n )** = " $1 \times x \times x \times x \times x$ " 2784 where, 2785 "1" is the code word 2786 the **sign** bit is not used 2787 "xxxxxxx" is the seven bit value field

The codec equation is described as follows:

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

if (value > Xpred( n )) then Xdeco( n ) = value + 3

2788

27892790

2791

Copyright © 2005-2014 MIPI Alliance, Inc.
All rights reserved.

```
22 Jan-2014
2792
                endif
2793
                else
2794
                    Xdeco(n) = value + 4
2795
                endif
2796
        E.3.2
                    Decoder for 10-7-10 Data Compression
2797
        Pixels without prediction are decoded using the following formula:
2798
                Xdeco(n) = 8 * Xenco(n) + 4
2799
        Pixels with prediction are decoded using the following formula:
                if (Xenco(n) & 0x70 == 0x00) then
2800
2801
                     use DPCM1
2802
                else if (Xenco(\mathbf{n}) & 0x78 == 0x10) then
                    use DPCM2
2803
2804
                else if (Xenco(\mathbf{n}) & 0x78 == 0x18) then
                    use DPCM3
2805
                else if (Xenco(\mathbf{n}) & 0x60 == 0x20) then
2806
                    use DPCM4
2807
2808
                else
                    use PCM
2809
2810
                endif
2811
        E.3.2.1
                     DPCM1 for 10-7-10 Decoder
2812
        Xenco( n ) has the following format:
2813
                Xenco( n ) = "000 \text{ s } xxx"
2814
                where,
2815
                    "000" is the code word
2816
                    "s" is the sign bit
                    "xxx" is the three bit value field
2817
2818
        The codec equation is described as follows:
                sign = Xenco(n) & 0x8
2819
                value = Xenco(n) & 0x7
2820
2821
                if (sign > 0) then
2822
                    Xdeco( n ) = Xpred( n ) - value
2823
                else
2824
                    Xdeco(n) = Xpred(n) + value
                endif
2825
2826
        E.3.2.2
                     DPCM2 for 10-7-10 Decoder
        Xenco( n ) has the following format:
2827
2828
                Xenco( n ) = "0010 \text{ s xx}"
```

```
22 Jan-2014
2829
                where,
2830
                    "0010" is the code word
                    "s" is the sign bit
2831
2832
                    "xx" is the two bit value field
2833
        The codec equation is described as follows:
2834
                sign = Xenco(n) & 0x4
2835
                value = 2 * (Xenco(n) & 0x3) + 8
2836
                if (sign > 0) then
                    Xdeco( n ) = Xpred( n ) - value
2837
2838
                else
                    Xdeco(n) = Xpred(n) + value
2839
2840
                endif
2841
        E.3.2.3
                     DPCM3 for 10-7-10 Decoder
2842
        Xenco( n ) has the following format:
                Xenco( n ) = "0011 \text{ s } xx"
2843
2844
                where,
2845
                    "0011" is the code word
                    "s" is the sign bit
2846
2847
                     "xx" is the two bit value field
2848
        The codec equation is described as follows:
                sign = Xenco(n) & 0x4
2849
2850
                value = 4 * (Xenco(n) & 0x3) + 16 + 1
                if (sign > 0) then
2851
2852
                    Xdeco( n ) = Xpred( n ) - value
                    if (Xdeco(n) < 0) then
2853
2854
                         Xdeco(n) = 0
2855
                    endif
                else
2856
                    Xdeco(n) = Xpred(n) + value
2857
                    if (Xdeco(n) > 1023) then
2858
                         Xdeco(n) = 1023
2859
2860
                    endif
2861
                endif
2862
        E.3.2.4
                     DPCM4 for 10-7-10 Decoder
2863
        Xenco( n ) has the following format:
                Xenco( n ) = "01 \text{ s } xxxx"
2864
2865
                where,
2866
                    "01" is the code word
                    "s" is the sign bit
2867
                     "xxxx" is the four bit value field
2868
```

Copyright © 2005-2014 MIPI Alliance, Inc. All rights reserved.

```
2869 The codec equation is described as follows:
```

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

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

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

```
2885 Xenco( n ) = "1 xxxxxx"
```

where,

2897

```
2887 "1" is the code word
2888 the sign bit is not used
2889 "xxxxxx" is the six bit value field
```

2890 The codec equation is described as follows:

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

2898 Pixels without prediction are decoded using the following formula:

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

2900 Pixels with prediction are decoded using the following formula:

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

```
22 Jan-2014
2911
                else
2912
                    use PCM
2913
                endif
2914
        E.3.3.1
                     DPCM1 for 10-6-10 Decoder
2915
        Xenco( n ) has the following format:
                Xenco( n ) = "00000 \text{ s}"
2916
2917
                where,
2918
                     "00000" is the code word
2919
                     "s" is the sign bit
2920
                    the value field is not used
2921
        The codec equation is described as follows:
                Xdeco(n) = Xpred(n)
2922
        E.3.3.2
                     DPCM2 for 10-6-10 Decoder
2923
        Xenco( n ) has the following format:
2924
                Xenco( n ) = "00001 \text{ s}"
2925
2926
                where,
2927
                     "00001" is the code word
2928
                     "s" is the sign bit
2929
                    the value field is not used
2930
        The codec equation is described as follows:
2931
                sign = Xenco(n) & 0x1
                value = 1
2932
                if (sign > 0) then
2933
2934
                    Xdeco(n) = Xpred(n) - value
2935
                else
                    Xdeco(n) = Xpred(n) + value
2936
2937
                endif
2938
        E.3.3.3
                     DPCM3 for 10-6-10 Decoder
2939
        Xenco(n) has the following format:
2940
                Xenco( n ) = "0001 s x"
2941
                where,
2942
                    "0001" is the code word
2943
                    "s" is the sign bit
2944
                    "x" is the one bit value field
```

```
2945
        The codec equation is described as follows:
2946
                sign = Xenco(n) & 0x2
                value = 4 * (Xenco(n) & 0x1) + 3 + 1
2947
                if (sign > 0) then
2948
                    Xdeco(n) = Xpred(n) - value
2949
                    if (Xdeco(n) < 0) then
2950
2951
                        Xdeco(n) = 0
2952
                    endif
2953
                else
2954
                    Xdeco(n) = Xpred(n) + value
                    if (Xdeco(n) > 1023) then
2955
                        Xdeco(n) = 1023
2956
                    endif
2957
                endif
2958
        E.3.3.4
2959
                     DPCM4 for 10-6-10 Decoder
2960
        Xenco( n ) has the following format:
2961
                Xenco( n ) = "001 \text{ s xx}"
2962
                where,
2963
                    "001" is the code word
2964
                    "s" is the sign bit
2965
                    "xx" is the two bit value field
2966
        The codec equation is described as follows:
2967
                sign = Xenco(n) & 0x4
                value = 8 * (Xenco(n) & 0x3) + 11 + 3
2968
                if (sign > 0) then
2969
                    Xdeco( n ) = Xpred( n ) - value
2970
2971
                    if (Xdeco(n) < 0) then
2972
                        Xdeco(n) = 0
2973
                    endif
2974
                else
2975
                    Xdeco(n) = Xpred(n) + value
                    if (Xdeco(n) > 1023) then
2976
2977
                        Xdeco(n) = 1023
2978
                    endif
2979
                endif
2980
                     DPCM5 for 10-6-10 Decoder
        E.3.3.5
2981
        Xenco( n ) has the following format:
                Xenco( n ) = "01 \text{ s } xxx"
2982
2983
                where,
2984
                    "01" is the code word
                    "s" is the sign bit
2985
2986
                    "xxx" is the three bit value field
```

```
2987
        The codec equation is described as follows:
2988
                sign = Xenco(n) & 0x8
                value = 16 * (Xenco(n) & 0x7) + 43 + 7
2989
                if (sign > 0) then
2990
                    Xdeco(n) = Xpred(n) - value
2991
                    if (Xdeco(n) < 0) then
2992
2993
                         Xdeco(n) = 0
2994
                    endif
2995
                else
2996
                    Xdeco(n) = Xpred(n) + value
                    if (Xdeco(n) > 1023) then
2997
2998
                        Xdeco(n) = 1023
                    endif
2999
                endif
3000
3001
        E.3.3.6
                     PCM for 10-6-10 Decoder
3002
        Xenco( n ) has the following format:
3003
                Xenco( n ) = "1 \times x \times x \times"
3004
                where,
3005
                    "1" is the code word
                    the sign bit is not used
3006
                     "xxxxx" is the five bit value field
3007
3008
        The codec equation is described as follows:
                value = 32 * (Xenco(n) & 0x1f)
3009
                if (value > Xpred( n )) then
3010
                    Xdeco(n) = value + 15
3011
3012
                else
3013
                    Xdeco(n) = value + 16
3014
                endif
        E.3.4
3015
                    Decoder for 12–8–12 Data Compression
3016
        Pixels without prediction are decoded using the following formula:
3017
                Xdeco(n) = 16 * Xenco(n) + 8
3018
        Pixels with prediction are decoded using the following formula:
3019
                if (Xenco(\mathbf{n}) & 0xf0 == 0x00) then
3020
                     use DPCM1
3021
                else if (Xenco(\mathbf{n}) & 0xe0 == 0x60) then
                     use DPCM2
3022
                else if (Xenco(\mathbf{n}) & 0xe0 == 0x40) then
3023
                    use DPCM3
3024
3025
                else if (Xenco(\mathbf{n}) & 0xe0 == 0x20) then
3026
                     use DPCM4
                else if (Xenco(\mathbf{n}) & 0xf0 == 0x10) then
3027
                    use DPCM5
3028
```

```
Version 1.2
        22 Jan-2014
3029
                else
3030
                    use PCM
3031
                endif
        E.3.4.1
                     DPCM1 for 12-8-12 Decoder
3032
3033
        Xenco( n ) has the following format:
3034
                Xenco( n ) = "0000 \text{ s xxx}"
3035
                where,
3036
                    "0000" is the code word
                    "s" is the sign bit
3037
3038
                    "xxx" is the three bit value field
3039
        The codec equation is described as follows:
3040
                sign = Xenco(n) & 0x8
                value = Xenco(n) & 0x7
3041
                if (sign > 0) then
3042
3043
                    Xdeco( n ) = Xpred( n ) - value
3044
                else
3045
                    Xdeco(n) = Xpred(n) + value
3046
                endif
        E.3.4.2
3047
                     DPCM2 for 12-8-12 Decoder
3048
        Xenco( n ) has the following format:
                Xenco( n ) = "011 s xxxx"
3049
3050
                where,
3051
                    "011" is the code word
3052
                    "s" is the sign bit
                    "xxxx" is the four bit value field
3053
3054
        The codec equation is described as follows:
                sign = Xenco(n) & 0x10
3055
                value = 2 * (Xenco(n) & 0xf) + 8
3056
                if (sign > 0) then
3057
                    Xdeco(n) = Xpred(n) - value
3058
3059
                else
3060
                    Xdeco(n) = Xpred(n) + value
                endif
3061
3062
        E.3.4.3
                     DPCM3 for 12-8-12 Decoder
3063
        Xenco( n ) has the following format:
3064
                Xenco( n ) = "010 \text{ s } xxxx"
```

```
22 Jan-2014
3065
                where,
3066
                    "010" is the code word
                    "s" is the sign bit
3067
3068
                    "xxxx" is the four bit value field
3069
        The codec equation is described as follows:
3070
                sign = Xenco(n) & 0x10
3071
                value = 4 * (Xenco(n) & 0xf) + 40 + 1
3072
                if (sign > 0) then
                    Xdeco(n) = Xpred(n) - value
3073
                    if (Xdeco(n) < 0) then
3074
                        Xdeco(n) = 0
3075
                    endif
3076
3077
                else
3078
                    Xdeco(n) = Xpred(n) + value
                    if (Xdeco(n) > 4095) then
3079
                        Xdeco(n) = 4095
3080
3081
                    endif
                endif
3082
        E.3.4.4
                     DPCM4 for 12-8-12 Decoder
3083
3084
        Xenco( n ) has the following format:
                Xenco( n ) = "001 \text{ s } xxxx"
3085
3086
                where,
3087
                    "001" is the code word
                    "s" is the sign bit
3088
                    "xxxx" is the four bit value field
3089
3090
        The codec equation is described as follows:
3091
                sign = Xenco(n) & 0x10
                value = 8 * (Xenco(n) & 0xf) + 104 + 3
3092
                if (sign > 0) then
3093
                    Xdeco( n ) = Xpred( n ) - value
3094
                    if (Xdeco(n) < 0) then
3095
3096
                        Xdeco(n) = 0
3097
                    endif
3098
                else
                    Xdeco(n) = Xpred(n) + value
3099
                    if (Xdeco(n) > 4095)
3100
                        Xdeco(n) = 4095
3101
3102
                    endif
3103
                endif
3104
        E.3.4.5
                     DPCM5 for 12-8-12 Decoder
3105
        Xenco( n ) has the following format:
3106
                Xenco( n ) = "0001 \text{ s } xxx"
```

```
22 Jan-2014
3107
                where,
3108
                    "0001" is the code word
3109
                    "s" is the sign bit
3110
                    "xxx" is the three bit value field
3111
        The codec equation is described as follows:
3112
                sign = Xenco(n) & 0x8
3113
                value = 16 * (Xenco(n) & 0x7) + 232 + 7
3114
                if (sign > 0) then
                    Xdeco(n) = Xpred(n) - value
3115
                    if (Xdeco(n) < 0) then
3116
                        Xdeco(n) = 0
3117
3118
                    endif
3119
                else
3120
                    Xdeco(n) = Xpred(n) + value
                    if (Xdeco(n) > 4095) then
3121
                        Xdeco(n) = 4095
3122
3123
                    endif
                endif
3124
        E.3.4.6
3125
                     PCM for 12–8–12 Decoder
3126
        Xenco( n ) has the following format:
                Xenco(\mathbf{n}) = "1 xxxxxxx"
3127
3128
                where.
3129
                    "1" is the code word
                    the sign bit is not used
3130
                    "xxxxxxx" is the seven bit value field
3131
        The codec equation is described as follows:
3132
3133
                value = 32 * (Xenco(n) & 0x7f)
                if (value > Xpred(n)) then
3134
                    Xdeco(n) = value + 15
3135
3136
                else
                    Xdeco(n) = value + 16
3137
3138
                endif
3139
        E.3.5
                   Decoder for 12-7-12 Data Compression
3140
        Pixels without prediction are decoded using the following formula:
3141
                Xdeco(n) = 32 * Xenco(n) + 16
3142
        Pixels with prediction are decoded using the following formula:
3143
                if (Xenco(n) & 0x78 == 0x00) then
                    use DPCM1
3144
                else if (Xenco(\mathbf{n}) & 0x78 == 0x08) then
3145
```

use **DPCM2** 

3146

Version 1.2

Copyright © 2005-2014 MIPI Alliance, Inc.

```
22 Jan-2014
3147
                else if (Xenco(\mathbf{n}) & 0x78 == 0x10) then
                    use DPCM3
3148
3149
                else if (Xenco(\mathbf{n}) & 0x70 == 0x20) then
                    use DPCM4
3150
                else if (Xenco(\mathbf{n}) & 0x70 == 0x30) then
3151
                     use DPCM5
3152
                else if (Xenco(\mathbf{n}) & 0x78 == 0x18) then
3153
                    use DPCM6
3154
3155
                else
                    use PCM
3156
3157
                endif
3158
        DPCM1 for 12-7-12 Decoder
3159
        Xenco( n ) has the following format:
                Xenco( n ) = "0000 \text{ s xx}"
3160
3161
                where,
3162
                    "0000" is the code word
3163
                    "s" is the sign bit
3164
                     "xx" is the two bit value field
3165
        The codec equation is described as follows:
3166
                sign = Xenco(n) & 0x4
                value = Xenco(n) & 0x3
3167
3168
                if (sign > 0) then
3169
                    Xdeco(n) = Xpred(n) - value
3170
                else
                    Xdeco(n) = Xpred(n) + value
3171
                endif
3172
        E.3.5.1
3173
                     DPCM2 for 12-7-12 Decoder
3174
        Xenco( n ) has the following format:
3175
                Xenco( n ) = "0001 \text{ s xx}"
3176
                where,
3177
                    "0001" is the code word
3178
                    "s" is the sign bit
3179
                     "xx" is the two bit value field
3180
        The codec equation is described as follows:
3181
                sign = Xenco(n) & 0x4
3182
                value = 2 * (Xenco(n) & 0x3) + 4
                if (sign > 0) then
3183
                    Xdeco(n) = Xpred(n) - value
3184
3185
                else
3186
                     Xdeco(n) = Xpred(n) + value
                endif
3187
```

```
3188
        E.3.5.2
                     DPCM3 for 12-7-12 Decoder
3189
        Xenco( n ) has the following format:
3190
                Xenco( n ) = "0010 \text{ s xx}"
3191
                where.
3192
                    "0010" is the code word
3193
                    "s" is the sign bit
                    "xx" is the two bit value field
3194
3195
        The codec equation is described as follows:
3196
                sign = Xenco(n) & 0x4
                value = 4 * (Xenco(n) & 0x3) + 12 + 1
3197
3198
                if (sign > 0) then
                    Xdeco(n) = Xpred(n) - value
3199
                    if (Xdeco(n) < 0) then
3200
                        Xdeco(n) = 0
3201
3202
                    endif
3203
                else
                    Xdeco(n) = Xpred(n) + value
3204
                    if (Xdeco(n) > 4095) then
3205
3206
                        Xdeco(n) = 4095
3207
                    endif
                endif
3208
        E.3.5.3
                     DPCM4 for 12-7-12 Decoder
3209
3210
        Xenco( n ) has the following format:
                Xenco( n ) = "010 \text{ s } xxx"
3211
3212
                where,
3213
                    "010" is the code word
3214
                    "s" is the sign bit
                    "xxx" is the three bit value field
3215
3216
        The codec equation is described as follows:
                sign = Xenco(n) & 0x8
3217
3218
                value = 8 * (Xenco(n) & 0x7) + 28 + 3
                if (\mathbf{sign} > 0) then
3219
                    Xdeco(n) = Xpred(n) - value
3220
3221
                    if (Xdeco(n) < 0) then
                        Xdeco(n) = 0
3222
3223
                    endif
3224
                else
3225
                    Xdeco(n) = Xpred(n) + value
3226
                    if (Xdeco(\mathbf{n}) > 4095) then
                        Xdeco(n) = 4095
3227
                    endif
3228
                endif
3229
```

```
3230
        E.3.5.4
                     DPCM5 for 12-7-12 Decoder
3231
        Xenco( n ) has the following format:
3232
                Xenco( n ) = "011 \text{ s } xxx"
3233
                where.
                    "011" is the code word
3234
                    "s" is the sign bit
3235
                    "xxx" is the three bit value field
3236
3237
        The codec equation is described as follows:
3238
                sign = Xenco(n) & 0x8
                value = 16 * (Xenco(n) & 0x7) + 92 + 7
3239
3240
                if (sign > 0) then
                    Xdeco(n) = Xpred(n) - value
3241
                    if (Xdeco(n) < 0) then
3242
                        Xdeco(n) = 0
3243
3244
                    endif
3245
                else
                    Xdeco(n) = Xpred(n) + value
3246
                    if (Xdeco(n) > 4095) then
3247
3248
                        Xdeco(n) = 4095
3249
                    endif
                endif
3250
3251
        E.3.5.5
                     DPCM6 for 12-7-12 Decoder
3252
        Xenco( n ) has the following format:
                Xenco( n ) = "0011 \text{ s } xx"
3253
3254
                where,
3255
                    "0011" is the code word
                    "s" is the sign bit
3256
3257
                    "xx" is the two bit value field
3258
        The codec equation is described as follows:
                sign = Xenco(n) & 0x4
3259
                value = 32 * (Xenco(n) & 0x3) + 220 + 15
3260
                if (sign > 0) then
3261
3262
                    Xdeco(n) = Xpred(n) - value
3263
                    if (Xdeco(n) < 0) then
                        Xdeco(n) = 0
3264
3265
                    endif
3266
                else
                    Xdeco(n) = Xpred(n) + value
3267
                    if (Xdeco(\mathbf{n}) > 4095) then
3268
                        Xdeco(n) = 4095
3269
                    endif
3270
3271
                endif
```

```
3272
        E.3.5.6
                      PCM for 12-7-12 Decoder
3273
        Xenco( n ) has the following format:
3274
                Xenco( n ) = "1 \times x \times x \times x"
3275
                 where.
3276
                     "1" is the code word
                     the sign bit is not used
3277
                     "xxxxxx" is the six bit value field
3278
3279
        The codec equation is described as follows:
3280
                value = 64 * (Xenco( n ) & 0x3f)
                if (value > Xpred( n )) then
3281
3282
                     Xdeco(n) = value + 31
3283
                else
                     Xdeco(n) = value + 32
3284
3285
                endif
3286
        E.3.6
                    Decoder for 12-6-12 Data Compression
3287
        Pixels without prediction are decoded using the following formula:
                 Xdeco(n) = 64 * Xenco(n) + 32
3288
3289
        Pixels with prediction are decoded using the following formula:
3290
                if (Xenco(\mathbf{n}) & 0x3c == 0x00) then
3291
                     use DPCM1
                else if (Xenco(\mathbf{n}) & 0x3c == 0x04) then
3292
                     use DPCM3
3293
                else if (Xenco(\mathbf{n}) & 0x38 == 0x10) then
3294
3295
                     use DPCM4
                else if (Xenco(\mathbf{n}) & 0x3c == 0x08) then
3296
3297
                     use DPCM5
3298
                else if (Xenco(\mathbf{n}) & 0x38 == 0x18) then
                     use DPCM6
3299
                else if (Xenco( n ) & 0x3c == 0x0c) then
3300
                     use DPCM7
3301
3302
                else
3303
                     use PCM
                endif
3304
3305
        Note: DPCM2 is not used.
3306
        E.3.6.1
                      DPCM1 for 12-6-12 Decoder
3307
        Xenco( n ) has the following format:
                Xenco( n ) = "0000 s x"
3308
```

```
22 Jan-2014
3309
                where,
3310
                    "0000" is the code word
                    "s" is the sign bit
3311
                    "x" is the one bit value field
3312
3313
        The codec equation is described as follows:
3314
                sign = Xenco(n) & 0x2
3315
                value = Xenco(n) & 0x1
3316
                if (sign > 0) then
                    Xdeco(n) = Xpred(n) - value
3317
3318
                else
                    Xdeco(n) = Xpred(n) + value
3319
3320
                endif
3321
        E.3.6.2
                     DPCM3 for 12-6-12 Decoder
3322
        Xenco( n ) has the following format:
                Xenco( n ) = "0001 s x"
3323
3324
                where,
3325
                    "0001" is the code word
                    "s" is the sign bit
3326
3327
                    "x" is the one bit value field
3328
        The codec equation is described as follows:
                sign = Xenco(n) & 0x2
3329
3330
                value = 4 * (Xenco(n) & 0x1) + 2 + 1
                if (sign > 0) then
3331
                    Xdeco(n) = Xpred(n) - value
3332
                    if (Xdeco(n) < 0) then
3333
3334
                        Xdeco(n) = 0
3335
                    endif
                else
3336
                    Xdeco(n) = Xpred(n) + value
3337
                    if (Xdeco(\mathbf{n}) > 4095) then
3338
                        Xdeco(n) = 4095
3339
3340
                    endif
3341
                endif
3342
        E.3.6.3
                     DPCM4 for 12-6-12 Decoder
3343
        Xenco( n ) has the following format:
                Xenco( n ) = "010 s xx"
3344
3345
                where,
3346
                    "010" is the code word
3347
                    "s" is the sign bit
                    "xx" is the two bit value field
3348
```

Copyright © 2005-2014 MIPI Alliance, Inc.

```
3349
        The codec equation is described as follows:
3350
                sign = Xenco(n) & 0x4
                value = 8 * (Xenco(n) & 0x3) + 10 + 3
3351
                if (sign > 0) then
3352
                    Xdeco(n) = Xpred(n) - value
3353
                    if (Xdeco(n) < 0) then
3354
3355
                        Xdeco(n) = 0
                    endif
3356
3357
                else
3358
                    Xdeco(n) = Xpred(n) + value
                    if (Xdeco(n) > 4095) then
3359
                        Xdeco(n) = 4095
3360
                    endif
3361
                endif
3362
3363
        E.3.6.4
                     DPCM5 for 12-6-12 Decoder
3364
        Xenco( n ) has the following format:
3365
                Xenco( n ) = "0010 s x"
3366
                where,
3367
                    "0010" is the code word
3368
                    "s" is the sign bit
                    "x" is the one bit value field
3369
3370
        The codec equation is described as follows:
3371
                sign = Xenco(n) & 0x2
                value = 16 * (Xenco(n) & 0x1) + 42 + 7
3372
                if (sign > 0) then
3373
                    Xdeco( n ) = Xpred( n ) - value
3374
3375
                    if (Xdeco(n) < 0) then
                        Xdeco(n) = 0
3376
                    endif
3377
3378
                else
                    Xdeco(n) = Xpred(n) + value
3379
                    if (Xdeco(\mathbf{n}) > 4095) then
3380
3381
                        Xdeco(n) = 4095
3382
                    endif
3383
                endif
                     DPCM6 for 12-6-12 Decoder
3384
        E.3.6.5
3385
        Xenco( n ) has the following format:
                Xenco( n ) = "011 \text{ s } xx"
3386
3387
                where,
3388
                    "011" is the code word
                    "s" is the sign bit
3389
3390
                    "xx" is the two bit value field
```

```
3391
        The codec equation is described as follows:
3392
                sign = Xenco(n) & 0x4
                value = 32 * (Xenco(n) & 0x3) + 74 + 15
3393
                if (sign > 0) then
3394
                    Xdeco(n) = Xpred(n) - value
3395
                    if (Xdeco(n) < 0) then
3396
3397
                        Xdeco(n) = 0
                    endif
3398
3399
                else
3400
                    Xdeco(n) = Xpred(n) + value
                    if (Xdeco(n) > 4095) then
3401
                        Xdeco(n) = 4095
3402
                    endif
3403
                endif
3404
3405
        E.3.6.6
                     DPCM7 for 12-6-12 Decoder
3406
        Xenco( n ) has the following format:
3407
                Xenco(n) = "0011 s x"
3408
                where,
3409
                    "0011" is the code word
3410
                    "s" is the sign bit
                    "x" is the one bit value field
3411
3412
        The codec equation is described as follows:
3413
                sign = Xenco(n) & 0x2
                value = 64 * (Xenco( n ) & 0x1) + 202 + 31
3414
                if (sign > 0) then
3415
                    Xdeco(n) = Xpred(n) - value
3416
3417
                    if (Xdeco(n) < 0) then
3418
                        Xdeco(n) = 0
3419
                    endif
3420
                else
3421
                    Xdeco(n) = Xpred(n) + value
                    if (Xdeco(\mathbf{n}) > 4095) then
3422
3423
                        Xdeco(n) = 4095
3424
                    endif
3425
                endif
                     PCM for 12-6-12 Decoder
3426
        E.3.6.7
3427
        Xenco( n ) has the following format:
                Xenco( n ) = "1 \times x \times x \times"
3428
3429
                where,
                    "1" is the code word
3430
                    the sign bit is not used
3431
3432
                    "xxxxx" is the five bit value field
```

3433 The codec equation is described as follows:

3434	value = 128 * (Xenco( n ) & 0x1f)
3435	if (value > Xpred( n )) then
3436	Xdeco(n) = value + 63
3437	else
3438	Xdeco(n) = value + 64
3439	endif

3440

3460 3461

## Annex F JPEG Interleaving (informative)

- 3441 This annex illustrates how the standard features of the CSI-2 protocol should be used to interleave (multiplex)
- 3442 JPEG image data with other types of image data, e.g. RGB565 or YUV422, without requiring a custom JPEG
- 3443 format such as JPEG8.
- 3444 The Virtual Channel Identifier and Data Type value in the CSI-2 Packet Header provide simple methods of
- 3445 interleaving multiple data streams or image data types at the packet level. Interleaving at the packet level
- minimizes the amount of buffering required in the system.
- 3447 The Data Type value in the CSI-2 Packet Header should be used to multiplex different image data types at
- 3448 the CSI-2 transmitter and de-multiplex the data types at the CSI-2 receiver.
- 3449 The Virtual Channel Identifier in the CSI-2 Packet Header should be used to multiplex different data streams
- 3450 (channels) at the CSI-2 transmitter and de-multiplex the streams at the CSI-2 receiver.
- The main difference between the two interleaving methods is that images with different Data Type values
- 3452 within the same Virtual Channel use the same frame and line synchronization information, whereas multiple
- 3453 Virtual Channels (data streams) each have their own independent frame and line synchronization information
- and thus potentially each channel may have different frame rates.
- 3455 Since the predefined Data Type values represent only YUV, RGB and RAW data types, one of the User
- 3456 Defined Data Type values should be used to represent JPEG image data.
- 3457 Figure 145 illustrates interleaving JPEG image data with YUV422 image data using Data Type values.
- Figure 146 illustrates interleaving JPEG image data with YUV422 image data using both Data Type values and Virtual Channel Identifiers.

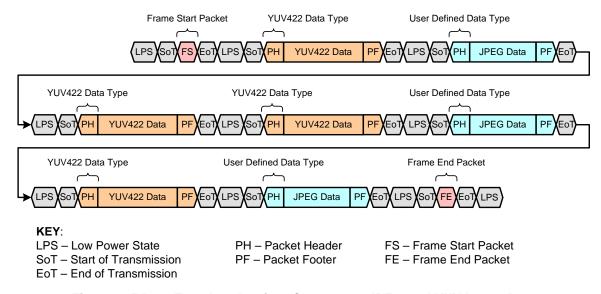


Figure 145 Data Type Interleaving: Concurrent JPEG and YUV Image Data

3462 3463

3464

3465

3468

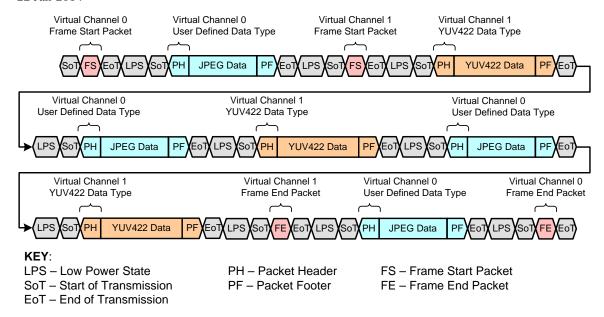


Figure 146 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data

Both Figure 145 and Figure 146 can be similarly extended to the interleaving of JPEG image data with any other type of image data, e.g. RGB565.

- Figure 147 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
- 3470 Again, these examples could also represent interleaving JPEG data with any other image data type.

34713472

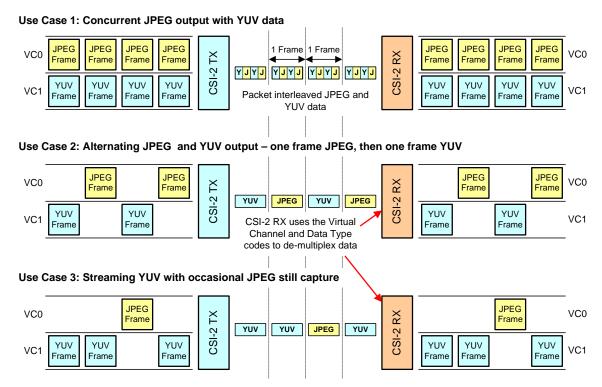


Figure 147 Example JPEG and YUV Interleaving Use Cases