



Specification for Camera Serial Interface 2 (CSI-2[®])

**Version 4.0.1
23 May 2022**

MIPI Board Adopted 14 November 2022

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

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

NOTICE OF DISCLAIMER

The material contained herein is provided on an “AS IS” basis. To the maximum extent permitted by applicable law, this material is provided AS IS AND WITH ALL FAULTS, and the authors and developers of this material and MIPI Alliance Inc. (“MIPI”) hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of negligence. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THIS MATERIAL.

IN NO EVENT WILL ANY AUTHOR OR DEVELOPER OF THIS MATERIAL OR MIPI BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT RELATING TO THIS MATERIAL, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

The material contained herein is not a license, either expressly or impliedly, to any IPR owned or controlled by any of the authors or developers of this material or MIPI. Any license to use this material is granted separately from this document. This material is protected by copyright laws, and may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of MIPI Alliance. MIPI, MIPI Alliance and the dotted rainbow arch and all related trademarks, service marks, tradenames, and other intellectual property are the exclusive property of MIPI Alliance Inc. and cannot be used without its express prior written permission. The use or implementation of this material may involve or require the use of intellectual property rights (“IPR”) including (but not limited to) patents, patent applications, or copyrights owned by one or more parties, whether or not members of MIPI. MIPI does not make any search or investigation for IPR, nor does MIPI require or request the disclosure of any IPR or claims of IPR as respects the contents of this material or otherwise.

Without limiting the generality of the disclaimers stated above, users of this material are further notified that MIPI: (a) does not evaluate, test or verify the accuracy, soundness or credibility of the contents of this material; (b) does not monitor or enforce compliance with the contents of this material; and (c) does not certify, test, or in any manner investigate products or services or any claims of compliance with MIPI specifications or related material.

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

MIPI Alliance, Inc.
c/o IEEE-ISTO
445 Hoes Lane, Piscataway New Jersey 08854, United States
Attn: Managing Director

Contents

Figures	ix
Tables	xviii
Release History	xxi
1 Introduction	1
1.1 Scope	1
1.2 Purpose	1
2 Terminology	2
2.1 Use of Special Terms	2
2.2 Definitions	2
2.3 Abbreviations.....	4
2.4 Acronyms.....	5
3 References	7
4 Overview of CSI-2	9
5 CSI-2 Layer Definitions	11
6 Camera Control Interface (CCI)	13
6.1 CCI (I ² C) Data Transfer Protocol	14
6.1.1 CCI (I ² C) Message Type	14
6.1.2 CCI (I ² C) Read/Write Operations	15
6.2 CCI (I3C) Data Transfer Protocol.....	21
6.2.1 CCI (I3C SDR) Data Transfer Protocol	21
6.2.2 CCI (I3C DDR) Data Transfer Protocol	29
6.3 CCI (I3C) Error Detection and Recovery	39
6.3.1 CCI (I3C SDR) Error Detection and Recovery Method	39
6.3.2 CCI (I3C DDR) Error Detection and Recovery Method	41
6.3.3 Error Detection and Recovery for CCI (I3C) Controller Devices	47
6.4 CCI (I ² C) Target Addresses.....	47
6.5 CCI (I3C) Target Addresses.....	47
6.6 CCI Multi-Byte Registers	48
6.6.1 Overview.....	48
6.6.2 Transmission Byte Order for Multi-Byte Register Values	50
6.6.3 Multi-Byte Register Protocol (Informative)	51
6.7 CCI I/O Electrical and Timing Specifications	56
7 Physical Layer	59
7.1 D-PHY Physical Layer Option	59
7.2 C-PHY Physical Layer Option.....	60
7.3 PHY Support for the CSI-2 Unified Serial Link (USL) Feature.....	61
7.3.1 D-PHY Support Requirements for USL Feature.....	61
7.3.2 C-PHY Support Requirements for USL Feature	62

8 Multi-Lane Distribution and Merging	63
8.1 Lane Distribution for the D-PHY Physical Layer Option.....	67
8.2 Lane Distribution for the C-PHY Physical Layer Option.....	70
8.3 Multi-Lane Interoperability	72
8.3.1 C-PHY Lane Deskew	74
9 Low Level Protocol.....	75
9.1 Low Level Protocol Packet Format	76
9.1.1 Low Level Protocol Long Packet Format.....	76
9.1.2 Low Level Protocol Short Packet Format.....	81
9.2 Data Identifier (DI).....	82
9.3 Virtual Channel Identifier	82
9.4 Data Type (DT).....	84
9.5 Packet Header Error Correction Code for D-PHY Physical Layer Option.....	85
9.5.1 General Hamming Code Applied to Packet Header.....	85
9.5.2 Hamming-Modified Code.....	86
9.5.3 ECC Generation on TX Side.....	90
9.5.4 Applying ECC on RX Side (Informative)	91
9.6 Checksum Generation.....	93
9.7 Packet Spacing.....	95
9.8 Synchronization Short Packet Data Type Codes.....	96
9.8.1 Frame Synchronization Packets.....	96
9.8.2 Line Synchronization Packets.....	97
9.9 Generic Short Packet Data Type Codes	99
9.10 Packet Spacing Examples Using the Low Power State	100
9.11 Latency Reduction and Transport Efficiency (LRTE)	103
9.11.1 Interpacket Latency Reduction (ILR)	103
9.11.2 Using ILR and Enhanced Transport Efficiency Together	114
9.11.3 LRTE Register Tables	115
9.12 Unified Serial Link (USL)	118
9.12.1 USL Technical Overview	119
9.12.2 USL Command Payload Constructs	120
9.12.3 USL Operation Procedures	123
9.12.4 Monitoring USL Command Transport Integrity	125
9.12.5 USL Powerup / Reset, SNS Configuration, and Mode Switching	126
9.13 Data Scrambling	138
9.13.1 CSI-2 Scrambling for D-PHY	139
9.13.2 CSI-2 Scrambling for C-PHY	140
9.13.3 Scrambling Details.....	143
9.14 Smart Region of Interest (SROI)	148
9.14.1 Overview of SROI Frame Format.....	149
9.14.2 Transmission of SROI Embedded Data Packet	151
9.14.3 SROI Packet Detection Options.....	153
9.14.4 SROI Use Cases (Informative)	155
9.14.5 Format of SROI Embedded Data Packet (SEDP)	157
9.15 Packet Data Payload Size Rules	160
9.16 Frame Format Examples.....	161

9.17 Data Interleaving	164
9.17.1 Data Type Interleaving.....	164
9.17.2 Virtual Channel Identifier Interleaving.....	167
10 Color Spaces.....	169
10.1 RGB Color Space Definition	169
10.2 YUV Color Space Definition.....	169
11 Data Formats	171
11.1 Generic 8-bit Long Packet Data Types	173
11.1.1 Null and Blanking Data	173
11.1.2 Embedded Information	174
11.1.3 Generic Long Packet Data Types 1 Through 4	174
11.2 YUV Image Data	175
11.2.1 Legacy YUV420 8-bit.....	176
11.2.2 YUV420 8-bit	178
11.2.3 YUV420 10-bit	182
11.2.4 YUV422 8-bit	184
11.2.5 YUV422 10-bit	186
11.3 RGB Image Data.....	188
11.3.1 RGB888	189
11.3.2 RGB666	191
11.3.3 RGB565	193
11.3.4 RGB555	195
11.3.5 RGB444	196
11.4 RAW Image Data.....	197
11.4.1 RAW6	198
11.4.2 RAW7	199
11.4.3 RAW8	200
11.4.4 RAW10	201
11.4.5 RAW12	203
11.4.6 RAW14	204
11.4.7 RAW16	206
11.4.8 RAW20	207
11.4.9 RAW24	209
11.4.10 RAW28	211
11.5 User Defined Data Formats	212
12 Recommended Memory Storage.....	215
12.1 General/Arbitrary Data Reception.....	215
12.2 RGB888 Data Reception	216
12.3 RGB666 Data Reception	216
12.4 RGB565 Data Reception	217
12.5 RGB555 Data Reception	217
12.6 RGB444 Data Reception	218
12.7 YUV422 8-bit Data Reception	218
12.8 YUV422 10-bit Data Reception	219
12.9 YUV420 8-bit (Legacy) Data Reception.....	220
12.10 YUV420 8-bit Data Reception	221
12.11 YUV420 10-bit Data Reception	222

12.12	RAW6 Data Reception.....	223
12.13	RAW7 Data Reception.....	223
12.14	RAW8 Data Reception.....	224
12.15	RAW10 Data Reception.....	224
12.16	RAW12 Data Reception.....	225
12.17	RAW14 Data Reception.....	225
12.18	RAW16 Data Reception.....	226
12.19	RAW20 Data Reception.....	226
12.20	RAW24 Data Reception.....	227
12.21	RAW28 Data Reception.....	227
13	Always-On Sentinel Conduit (AOSC)	229
13.1	Introduction.....	229
13.1.1	VDSP and APP In-Band Communication.....	231
13.1.2	AOSC SNS Features and Capabilities	232
13.1.3	VDSP and APP Switching and Graceful Failure	232
13.1.4	Support for Privacy	233
13.2	Optimal Transport Mode (OTM) Overview	234
13.2.1	Protocol Overview	237
13.2.2	SNS FIFO Requirement.....	239
13.2.3	VDSP Initiated Commands to the SNS.....	239
13.2.4	SNS Status Communication to VDSP Using IBI.....	240
13.2.5	Support for On-Demand Frame and Streaming Frames	241
13.2.6	Support for Frame Squelching	241
13.2.7	Support for Interconnect Synchronization and Dynamic ISPB Insertion	242
13.2.8	OTM Errors Detected by the VDSP	243
13.3	Smart Transport Mode (STM) Overview.....	244
Annex A	JPEG8 Data Format (informative).....	249
A.1	Introduction.....	249
A.2	JPEG Data Definition	250
A.3	Image Status Information	251
A.4	Embedded Images.....	253
A.5	JPEG8 Non-standard Markers	254
A.6	JPEG8 Data Reception	254
Annex B	CSI-2 Implementation Example (informative)	255
B.1	Overview.....	255
B.2	CSI-2 Transmitter Detailed Block Diagram	256
B.3	CSI-2 Receiver Detailed Block Diagram.....	257
B.4	Details on the D-PHY Implementation.....	258
B.4.1	CSI-2 Clock Lane Transmitter	259
B.4.2	CSI-2 Clock Lane Receiver	260
B.4.3	CSI-2 Data Lane Transmitter	261
B.4.4	CSI-2 Data Lane Receiver	263

Annex C CSI-2 Recommended Receiver Error Behavior (informative)	265
C.1 Overview.....	265
C.2 D-PHY Level Error.....	266
C.3 Packet Level Error	267
C.4 Protocol Decoding Level Error.....	268
Annex D CSI-2 Sleep Mode (informative).....	269
D.1 Overview.....	269
D.2 SLM Command Phase	269
D.3 SLM Entry Phase	270
D.4 SLM Exit Phase	270
Annex E Data Compression for RAW Data Types (normative)	271
E.1 Predictors	273
E.1.1 Predictor1.....	273
E.1.2 Predictor2.....	274
E.2 Encoders	275
E.2.1 Coder for 10–8–10 Data Compression	275
E.2.2 Coder for 10–7–10 Data Compression	277
E.2.3 Coder for 10–6–10 Data Compression	280
E.2.4 Coder for 12–10–12 Data Compression.....	283
E.2.5 Coder for 12–8–12 Data Compression	285
E.2.6 Coder for 12–7–12 Data Compression	288
E.2.7 Coder for 12–6–12 Data Compression	291
E.3 Decoders	294
E.3.1 Decoder for 10–8–10 Data Compression.....	294
E.3.2 Decoder for 10–7–10 Data Compression.....	297
E.3.3 Decoder for 10–6–10 Data Compression.....	300
E.3.4 Decoder for 12–10–12 Data Compression.....	303
E.3.5 Decoder for 12–8–12 Data Compression.....	306
E.3.6 Decoder for 12–7–12 Data Compression.....	309
E.3.7 Decoder for 12–6–12 Data Compression.....	313
Annex F JPEG Interleaving (informative).....	317
Annex G Scrambler Seeds for Lanes 9 and Above.....	321
Annex H Guidance on CSI-2 Over C-PHY ALP and PPI	323
H.1 CSI-2 with C-PHY ALP Mode	323
H.1.1 Concepts of ALP Mode and Legacy LP Mode	323
H.1.2 Burst Examples Using ALP Mode	326
H.1.3 Transmission and Reception of ALP Commands Through the PPI	331
H.1.4 Multi-Lane Operation Using ALP Mode	336
H.1.5 LP and ALP Operation	338
H.1.6 Bi-Directional Lane Turnaround.....	338

Annex I Multi-Pixel Compression (MPC)	347
I.1 MPC for Bayer Image Sensor	352
I.1.1 Reference Pixels for Prediction (Bayer)	353
I.1.2 Basic Prediction Mode (Bayer).....	356
I.1.3 Advanced Prediction Mode (Bayer)	364
I.2 MPC for Tetra-Cell Image Sensor	372
I.2.1 Reference Pixels for Prediction (Tetra-Cell).....	373
I.2.2 Basic Prediction Mode (Tetra-Cell)	376
I.2.3 Advanced Prediction Mode (Tetra-Cell).....	388
I.2.4 Low Resolution Image Generation (Tetra-Cell)	400
I.2.5 Adjustment of Compression Ratio (Tetra-Cell).....	401
I.3 MPC for Nona-Cell Image Sensor.....	404
I.3.1 Reference Pixels for Prediction (Nona-Cell)	405
I.3.2 Basic Prediction Mode (Nona-Cell).....	407
I.3.3 Advanced Prediction Mode (Nona-Cell)	413
I.3.4 Low Resolution Image Generation (Nona-Cell).....	427
I.3.5 Adjustment of Compression Ratio (Nona-Cell).....	428
I.4 MPC for N x N Multi-Pixel Image Sensor [Informative]	430

Figures

Figure 1 Typical CSI-2 and CCI Transmitter and Receiver Interface for D-PHY	9
Figure 2 Typical CSI-2 and CCI Transmitter and Receiver Interface for C-PHY.....	10
Figure 3 CSI-2 Layer Definitions	11
Figure 4 CCI (I ² C) Single Read from Random Location.....	15
Figure 5 CCI (I ² C) Single Read from Current Location	16
Figure 6 CCI (I ² C) Sequential Read Starting from Random Location.....	17
Figure 7 CCI (I ² C) Sequential Read Starting from Current Location.....	18
Figure 8 CCI (I ² C) Single Write to Random Location.....	19
Figure 9 CCI (I ² C) Sequential Write Starting from Random Location.....	20
Figure 10 CCI (I3C SDR) Single Read from Random Location	23
Figure 11 CCI (I3C SDR) Single Read from Current Location	24
Figure 12 CCI (I3C SDR) Sequential Read Starting from Random Location	25
Figure 13 CCI (I3C SDR) Sequential Read Starting from Current Location.....	26
Figure 14 CCI (I3C SDR) Single Write to Random Location	27
Figure 15 CCI (I3C SDR) Sequential Write Starting from Random Location.....	28
Figure 16 CCI (I3C DDR) Sequential Read from Random Location: 8-bit LENGTH & INDEX	32
Figure 17 CCI (I3C DDR) Sequential Read from Random Location: 16-bit LENGTH & INDEX	33
Figure 18 CCI (I3C DDR) Concatenated Sequential Read, Random Location: 8-bit LENGTH & INDEX	35
Figure 19 CCI (I3C DDR) Concatenated Sequential Read, Random Location: 16-bit LENGTH & INDEX	36
Figure 20 CCI (I3C DDR) Sequential Write Starting from Random Location	38
Figure 21 Example of SS0 Error Detection	40
Figure 22 Example of SD0 Error Detection.....	42
Figure 23 Example of SD1 Error Detection.....	44
Figure 24 Example of MD0 Error Detection	46
Figure 25 Corruption of 32-bit Register During Read Message	49
Figure 26 Corruption of 32-bit Register During Write Message.....	49
Figure 27 Example 16-bit Register Write	50
Figure 28 Example 32-bit Register Write (Address Not Shown).....	50
Figure 29 Example 64-bit Register Write (Address Not Shown).....	50
Figure 30 Example 16-bit Register Read	51
Figure 31 Example 32-bit Register Read	53
Figure 32 Example 16-bit Register Write	54
Figure 33 Example 32-bit Register Write	55

Figure 34 CCI I/O Timing.....	58
Figure 35 Conceptual Overview of the Lane Distributor Function for D-PHY	63
Figure 36 Conceptual Overview of the Lane Distributor Function for C-PHY	64
Figure 37 Conceptual Overview of the Lane Merging Function for D-PHY	65
Figure 38 Conceptual Overview of the Lane Merging Function for C-PHY	66
Figure 39 Two Lane Multi-Lane Example for D-PHY	67
Figure 40 Three Lane Multi-Lane Example for D-PHY	68
Figure 41 N-Lane Multi-Lane Example for D-PHY	69
Figure 42 N-Lane Multi-Lane Example for D-PHY Short Packet Transmission.....	70
Figure 43 Two Lane Multi-Lane Example for C-PHY	71
Figure 44 Three Lane Multi-Lane Example for C-PHY	71
Figure 45 General N-Lane Multi-Lane Distribution for C-PHY	71
Figure 46 One Lane Transmitter and N-Lane Receiver Example for D-PHY	72
Figure 47 M-Lane Transmitter and N-Lane Receiver Example ($M < N$) for D-PHY	72
Figure 48 M-Lane Transmitter and One Lane Receiver Example for D-PHY.....	73
Figure 49 M-Lane Transmitter and N-Lane Receiver Example ($N < M$) for D-PHY	73
Figure 50 Example of Digital Logic to Align All RxDataHS	74
Figure 51 Low Level Protocol Packet Overview.....	75
Figure 52 Long Packet Structure for D-PHY Physical Layer Option	76
Figure 53 Long Packet Structure for C-PHY Physical Layer Option	77
Figure 54 Packet Header Lane Distribution for C-PHY Physical Layer Option.....	78
Figure 55 Minimal Filler Byte Insertion Requirements for Three Lane C-PHY	80
Figure 56 Short Packet Structure for D-PHY Physical Layer Option.....	81
Figure 57 Short Packet Structure for C-PHY Physical Layer Option	81
Figure 58 Data Identifier Byte	82
Figure 59 Logical Channel Block Diagram (Receiver)	82
Figure 60 Interleaved Video Data Streams Examples.....	83
Figure 61 26-bit ECC Generation Example	85
Figure 62 64-bit ECC Generation on TX Side	90
Figure 63 26-bit ECC Generation on TX Side	90
Figure 64 64-bit ECC on RX Side Including Error Correction.....	91
Figure 65 26-bit ECC on RX Side Including Error Correction.....	92
Figure 66 Checksum Transmission Byte Order	93
Figure 67 Checksum Generation for Long Packet Payload Data.....	93
Figure 68 Definition of 16-bit CRC Shift Register	94
Figure 69 16-bit CRC Software Implementation Example	94
Figure 70 Packet Spacing.....	95

Figure 71 Example Interlaced Frame Using LS/LE Short Packet and Line Counting.....	98
Figure 72 Multiple Packet Example.....	100
Figure 73 Single Packet Example	100
Figure 74 Line and Frame Blanking Definitions	101
Figure 75 Vertical Sync Example.....	102
Figure 76 Horizontal Sync Example	102
Figure 77 Interpacket Latency Reduction Using LRTE EPD	103
Figure 78 LRTE Efficient Packet Delimiter Example for CSI-2 Over C-PHY (2 Lanes).....	105
Figure 79 Example of LRTE EPD for CSI-2 Over D-PHY – Option 1	107
Figure 80 Example of LRTE EPD for CSI-2 Over D-PHY – Option 2	108
Figure 81 Enabling Robust Spacer Byte Detection: General Case	112
Figure 82 Enabling Robust Spacer Byte Detection: Special Case	112
Figure 83 EoTp Usage Examples.....	113
Figure 84 Using EPD with LVLP or ALP Mode Signaling.....	114
Figure 85 USL System Diagram	118
Figure 86 USL Modes Link Transitions.....	131
Figure 87 Examples of USL ALP Mode Clock Lane Management During Sensor Vblank	135
Figure 88 System Diagram Showing Per-Lane Scrambling	138
Figure 89 Example of Data Bursts in Two Lanes Using the D-PHY Physical Layer	139
Figure 90 Example of Data Bursts in Two Lanes Using the C-PHY Physical Layer.....	140
Figure 91 Generating Tx Sync Type as Seed Index (Single Lane View)	141
Figure 92 Generating Tx Sync Type Using the C-PHY Physical Layer	142
Figure 93 PRBS LFSR Serial Implementation Example	145
Figure 94 ISROI System Supporting Multi-Stack SNS with Integrated SROI VDSP.....	148
Figure 95 ESROI System Supporting Standard SNS with an External SROI VDSP	148
Figure 96 SROI Frame Format Example	150
Figure 97 SROI Packet Option 1	153
Figure 98 SROI Packet Option 2	154
Figure 99 Use Case 1: SROI Embedded Data Packet Not Transmitted.....	155
Figure 100 Use Case 2: SROI Embedded Data Packet Is Transmitted	156
Figure 101 SROI Embedded Data Packet Format	157
Figure 102 General Frame Format Example.....	161
Figure 103 Digital Interlaced Video Example.....	162
Figure 104 Digital Interlaced Video with Accurate Synchronization Timing Information.....	163
Figure 105 Interleaved Data Transmission using Data Type Value.....	164
Figure 106 Packet Level Interleaved Data Transmission.....	165
Figure 107 Frame Level Interleaved Data Transmission	166

Figure 108 Interleaved Data Transmission using Virtual Channels	167
Figure 109 Byte Packing Pixel Data to C-PHY Symbol Illustration	172
Figure 110 Frame Structure with Embedded Data at the Beginning and End of the Frame	174
Figure 111 Legacy YUV420 8-bit Transmission.....	176
Figure 112 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration	176
Figure 113 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1.....	177
Figure 114 Legacy YUV420 8-bit Frame Format.....	177
Figure 115 YUV420 8-bit Data Transmission Sequence	178
Figure 116 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration	179
Figure 117 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1	180
Figure 118 YUV420 Spatial Sampling for MPEG 2 and MPEG 4	180
Figure 119 YUV420 8-bit Frame Format.....	181
Figure 120 YUV420 10-bit Transmission	182
Figure 121 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration.....	183
Figure 122 YUV420 10-bit Frame Format	183
Figure 123 YUV422 8-bit Transmission.....	184
Figure 124 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration.....	184
Figure 125 YUV422 Co-sited Spatial Sampling.....	185
Figure 126 YUV422 8-bit Frame Format.....	185
Figure 127 YUV422 10-bit Transmitted Bytes	186
Figure 128 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration.....	186
Figure 129 YUV422 10-bit Frame Format	187
Figure 130 RGB888 Transmission.....	189
Figure 131 RGB888 Transmission in CSI-2 Bus Bitwise Illustration	189
Figure 132 RGB888 Frame Format	190
Figure 133 RGB666 Transmission with 18-bit BGR Words.....	191
Figure 134 RGB666 Transmission on CSI-2 Bus Bitwise Illustration	191
Figure 135 RGB666 Frame Format	192
Figure 136 RGB565 Transmission with 16-bit BGR Words.....	193
Figure 137 RGB565 Transmission on CSI-2 Bus Bitwise Illustration	193
Figure 138 RGB565 Frame Format	194
Figure 139 RGB555 Transmission on CSI-2 Bus Bitwise Illustration	195
Figure 140 RGB444 Transmission on CSI-2 Bus Bitwise Illustration	196
Figure 141 RAW6 Transmission.....	198
Figure 142 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration	198
Figure 143 RAW6 Frame Format.....	198
Figure 144 RAW7 Transmission	199

Figure 145 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration	199
Figure 146 RAW7 Frame Format.....	199
Figure 147 RAW8 Transmission	200
Figure 148 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration	200
Figure 149 RAW8 Frame Format.....	200
Figure 150 RAW10 Transmission.....	201
Figure 151 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration	201
Figure 152 RAW10 Frame Format.....	202
Figure 153 RAW12 Transmission.....	203
Figure 154 RAW12 Data Transmission on CSI-2 Bus Bitwise Illustration.....	203
Figure 155 RAW12 Frame Format.....	203
Figure 156 RAW14 Transmission.....	204
Figure 157 RAW14 Data Transmission on CSI-2 Bus Bitwise Illustration.....	204
Figure 158 RAW14 Frame Format.....	205
Figure 159 RAW16 Transmission.....	206
Figure 160 RAW16 Data Transmission on CSI-2 Bus Bitwise Illustration.....	206
Figure 161 RAW16 Frame Format.....	206
Figure 162 RAW20 Transmission.....	207
Figure 163 RAW20 Data Transmission on CSI-2 Bus Bitwise Illustration.....	207
Figure 164 RAW20 Frame Format.....	208
Figure 165 RAW24 Transmission.....	209
Figure 166 RAW24 Data Transmission on CSI-2 Bus Bitwise Illustration.....	209
Figure 167 RAW24 Frame Format.....	210
Figure 168 RAW28 Transmission.....	211
Figure 169 RAW28 Data Transmission on CSI-2 Bus Bitwise Illustration.....	211
Figure 170 RAW28 Frame Format.....	211
Figure 171 User Defined 8-bit Data (128 Byte Packet)	212
Figure 172 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration	212
Figure 173 Transmission of User Defined 8-bit Data	212
Figure 174 General/Arbitrary Data Reception.....	215
Figure 175 RGB888 Data Format Reception.....	216
Figure 176 RGB666 Data Format Reception.....	216
Figure 177 RGB565 Data Format Reception.....	217
Figure 178 RGB555 Data Format Reception.....	217
Figure 179 RGB444 Data Format Reception.....	218
Figure 180 YUV422 8-bit Data Format Reception.....	218
Figure 181 YUV422 10-bit Data Format Reception	219

Figure 182 YUV420 8-bit Legacy Data Format Reception	220
Figure 183 YUV420 8-bit Data Format Reception	221
Figure 184 YUV420 10-bit Data Format Reception	222
Figure 185 RAW6 Data Format Reception	223
Figure 186 RAW7 Data Format Reception	223
Figure 187 RAW8 Data Format Reception	224
Figure 188 RAW10 Data Format Reception	224
Figure 189 RAW12 Data Format Reception	225
Figure 190 RAW 14 Data Format Reception	225
Figure 191 RAW16 Data Format Reception	226
Figure 192 RAW20 Data Format Reception	226
Figure 193 RAW24 Data Format Reception	227
Figure 194 RAW28 Data Format Reception	227
Figure 195 Point-To-Point AOSC Systems with USL Solutions	229
Figure 196 Point-To-Point AOSC Systems with Non-USL Solutions	230
Figure 197 System Supporting AOSC and CCI Operations Over Multi-Drop I3C	230
Figure 198 System Supporting Discrete Multicontrollers Mapped to Single SNS I3C Target Port....	231
Figure 199 AOSC Optimal Transport Mode (OTM) Operation.....	235
Figure 200 Example OTM RAW10 Format Transport (with I3C SDR Bit Transmission Order).....	237
Figure 201 SNS OTM FIFO	239
Figure 202 AOSC IBI MDB Codes	241
Figure 203 OTM ISPB Insertion and Frame Start IBI T_RESP Parameter	242
Figure 204 AOSC Smart Transport Mode (STM) Operation for the D-PHY Generic STM Types	244
Figure 205 Example AOSC STM RAW10 Data Bitwise Transport Illustration with the D-PHY Generic STM Types	247
Figure 206 JPEG8 Data Flow in the Encoder	249
Figure 207 JPEG8 Data Flow in the Decoder	249
Figure 208 EXIF Compatible Baseline JPEG DCT Format.....	250
Figure 209 Status Information Field in the End of Baseline JPEG Frame.....	252
Figure 210 Example of TN Image Embedding Inside the Compressed JPEG Data Block.....	253
Figure 211 JPEG8 Data Format Reception	254
Figure 212 Implementation Example Block Diagram and Coverage	255
Figure 213 CSI-2 Transmitter Block Diagram.....	256
Figure 214 CSI-2 Receiver Block Diagram	257
Figure 215 D-PHY Level Block Diagram.....	258
Figure 216 CSI-2 Clock Lane Transmitter.....	259
Figure 217 CSI-2 Clock Lane Receiver	260

Figure 218 CSI-2 Data Lane Transmitter.....	261
Figure 219 CSI-2 Data Lane Receiver.....	263
Figure 220 SLM Synchronization.....	270
Figure 221 Data Compression System Block Diagram	272
Figure 222 Pixel Order of the Original Image	273
Figure 223 Example Pixel Order of the Original Image	273
Figure 224 Data Type Interleaving: Concurrent JPEG and YUV Image Data.....	317
Figure 225 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data	318
Figure 226 Example JPEG and YUV Interleaving Use Cases	319
Figure 227 Comparing Data Burst Timing of Legacy LP Mode versus ALP Mode	323
Figure 228 ALP Mode General Burst Format.....	324
Figure 229 High-Speed and ALP-Pause Wake Receiver Example	325
Figure 230 Examples of Bursts to Send High-Speed Data and ALP Commands	327
Figure 231 State Transitions for an HS Data Burst.....	328
Figure 232 State Transitions to Enter the ULPS State	329
Figure 233 State Transitions to Exit from the ULPS State.....	330
Figure 234 PPI Example: HS Signals for Transmission of Data, Sync and ALP Commands.....	331
Figure 235 PPI Example Transmit Side Timing for an HS Data Burst.....	332
Figure 236 PPI Example Receive Side Timing for an HS Data Burst	333
Figure 237 PPI Example Transmit Side Timing to Enter the ULPS State	334
Figure 238 PPI Example Receive Side Timing to Enter the ULPS State.....	334
Figure 239 PPI Example Transmit Side Timing to Exit from the ULPS State.....	335
Figure 240 PPI Example Receive Side Timing to Exit from the ULPS State.....	335
Figure 241 Example Showing a Data Transmission Burst using Three Lanes	337
Figure 242 Example Showing an ALP Command Burst using Three Lanes	337
Figure 243 High-Level View of the Control Mode Lane Turnaround Procedure	338
Figure 244 High-Level View of ALP Mode with the Control Mode Lane Turnaround Procedure.....	338
Figure 245 High-Level View of the Fast Lane Turnaround Procedure with ALP Mode.....	339
Figure 246 High-Level View, Comparing Lane Turnaround Procedures.....	340
Figure 247 Detailed View of the Fast Lane Turnaround Procedure	341
Figure 248 State Transitions from ALP-Pause Stop to Turnaround.....	342
Figure 249 State Transitions from Turnaround to Turnaround.....	343
Figure 250 State Transitions from Turnaround to ALP-Pause Stop.....	344
Figure 251 Example Fast Lane Turnaround at the First Transmitting Device	345
Figure 252 Example Fast Lane Turnaround at the Second Transmitting Device.....	346
Figure 253 MPC Data Compression Schemes	347
Figure 254 RAW10 Transmission for MPC for Bayer Image Sensors	348

Figure 255 RAW10/12/14 Transmission for MPC for Tetra-Cell Image Sensors.....	349
Figure 256 RAW10/12/14 Transmission for MPC for Nona-Cell Image Sensors	350
Figure 257 MPC Data Compression System.....	351
Figure 258 MPC Packet Structures for Bayer Compression Modes	352
Figure 259 Reference Pixel Relative Coordinate Indexes (Bayer)	354
Figure 260 MPC Packet Structure for Compression Mode PD.....	356
Figure 261 MPC Packet Structure for Compression Mode DGD (Bayer).....	359
Figure 262 MPC Packet Structure for Compression Mode FNR (Bayer).....	361
Figure 263 Decoded and Reference Pixel of Outlier Pixel (Bayer)	362
Figure 264 MPC Packet Structure for Compression Mode OUT (Bayer)	362
Figure 265 MPC Packet Structure for Compression Mode eDGD	364
Figure 266 MPC Packet Structure for Compression Mode ePD.....	366
Figure 267 MPC Packet Structure for Compression Mode eSHV (Bayer).....	369
Figure 268 MPC Packet Structures for Tetra-Cell Compression Modes.....	372
Figure 269 Tetra-Cell Multi-Pixel Indexing.....	373
Figure 270 Reference Pixel Relative Coordinate Indexes (Tetra-Cell).....	374
Figure 271 MPC Packet Structure for Compression Mode AD	376
Figure 272 MPC Packet Structure for Compression Mode OD	378
Figure 273 MPC Packet Structure for Compression Mode FNR (Tetra-Cell)	385
Figure 274 Decoded and Reference Pixel of Outlier Pixel (Tetra)	386
Figure 275 MPC Packet Structure for Compression Mode OUT (Tetra-Cell).....	386
Figure 276 MPC Packet Structure for Compression Mode eMPD (Tetra-Cell).....	388
Figure 277 MPC Packet Structure for Compression Mode eHVD (Tetra-Cell).....	391
Figure 278 Pixel Couplets Definitions for eHVA Mode.....	393
Figure 279 MPC Packet Structure for Compression Mode eHVA.....	394
Figure 280 MPC Packet Structure for Compression Mode eOUT	396
Figure 281 Low Resolution Image Generation by P_{byr} (Tetra-Cell)	400
Figure 282 MPC Packet Structures for Compression Ratio 10:6 (Tetra-Cell)	402
Figure 283 MPC Packet Structures for Compression Ratio 10:7 (Tetra-Cell)	402
Figure 284 Quantization Bits for Compression Ratios (Tetra-Cell).....	403
Figure 285 MPC Packet Structures for Nona-Cell Compression Modes	404
Figure 286 Nona-Cell Multi-Pixel Indexing	405
Figure 287 Reference Pixel Relative Coordinate Indexes (Nona-Cell)	406
Figure 288 MPC Packet Structure for Compression Mode AD (Nona-Cell)	407
Figure 289 MPC Packet Structure for Compression Mode DGD (Nona-Cell).....	409
Figure 290 MPC Packet Structure for Compression Mode FNR (Nona-Cell).....	412
Figure 291 MPC Packet Structure for Compression Mode eAD (Nona-Cell).....	413

Figure 292 MPC Packet Structure for Compression Mode eHVD (Nona-Cell)	415
Figure 293 Pixel Triplet Definitions for eHVI Mode.....	418
Figure 294 MPC Packet Structure for Compression Mode eHVI.....	419
Figure 295 MPC Packet Structure for Compression Mode eSHV (Nona-Cell).....	420
Figure 296 MPC Packet Structure for Compression Mode eMPD (Nona-Cell)	423
Figure 297 Low Resolution Image Generation by P_{byr} (Nona-Cell).....	427
Figure 298 MPC Packet Structures for Compression Ratio 10:6 (Nona-Cell)	428
Figure 299 MPC Packet Structures for Compression Ratio 10:7 (Nona-Cell)	428
Figure 300 Quantization Bits for Compression Ratios (Nona-Cell)	429
Figure 301 Pixel Order Change from 4x4 Multi-Pixel to Tetra-Cell	430

Tables

Table 1 CCI (I ² C) Read/Write Operations	15
Table 2 CCI (I3C SDR) Read/Write Operations	22
Table 3 CCI (I3C DDR) Read/Write Operations	30
Table 4 CCI (I3C DDR) Read/Write Operation Command Codes	31
Table 5 CCI (I3C SDR) Target Error Types	39
Table 6 CCI (I3C DDR) Target Error Types	41
Table 7 CCI (I3C DDR) Controller Error Type.....	45
Table 8 CCI I/O Electrical Specifications	56
Table 9 CCI I/O Timing Specifications.....	57
Table 10 Data Type Classes	84
Table 11 ECC Syndrome Association Matrix	86
Table 12 ECC Parity Generation Rules.....	88
Table 13 Synchronization Short Packet Data Type Codes	96
Table 14 Generic Short Packet Data Type Codes.....	99
Table 15 Minimum Spacer Bytes per Lane for ECC Calculation	111
Table 16 LRTE Transmitter Registers for CSI-2 Over C-PHY	115
Table 17 LRTE Transmitter Registers for CSI-2 Over D-PHY.....	116
Table 18 Image Sensor LPDT LRTE Control Register.....	121
Table 19 USL Transport Control (USL_CTL) Bit Description.....	122
Table 20 USL Transport Integrity ACK and NAK Registers with TSEQ	125
Table 21 USL BTA Switch Registers	128
Table 22 Register TX_USL_REV_FWD_ENTRY	129
Table 23 Register TX_USL_SNS_BTA_ACK_TIMEOUT[15:0].....	130
Table 24 Register TX_USL_APP_BTA_ACK_TIMEOUT[15:0].....	130
Table 25 USL Operation Registers.....	132
Table 26 USL GPIO Registers	132
Table 27 USL Clock Lane Control Register	135
Table 28 Symbol Sequence Values Per Sync Type	141
Table 29 Fields That Are Not Scrambled	143
Table 30 D-PHY Scrambler PRBS Initial Seed Values for Lanes 1 Through 8	143
Table 31 C-PHY Scrambler PRBS Initial Seed Values for Lanes 1 Through 8	144
Table 32 Example of the PRBS Bit-at-a-Time Shift Sequence	146
Table 33 Example PRBS LFSR Byte Sequence for D-PHY Physical Layer	146
Table 34 Example PRBS LFSR Byte Sequence for C-PHY Physical Layer	147

Table 35 Transmission of SROI Embedded Data Packet	152
Table 36 ROI Element Information Field Format.....	157
Table 37 ROI Element Type ID Definitions.....	158
Table 38 Primary and Secondary Data Formats Definitions.....	171
Table 39 Generic 8-bit Long Packet Data Types.....	173
Table 40 YUV Image Data Types.....	175
Table 41 Legacy YUV420 8-bit Packet Data Size Constraints	176
Table 42 YUV420 8-bit Packet Data Size Constraints.....	178
Table 43 YUV420 10-bit Packet Data Size Constraints.....	182
Table 44 YUV422 8-bit Packet Data Size Constraints.....	184
Table 45 YUV422 10-bit Packet Data Size Constraints.....	186
Table 46 RGB Image Data Types.....	188
Table 47 RGB888 Packet Data Size Constraints	189
Table 48 RGB666 Packet Data Size Constraints	191
Table 49 RGB565 Packet Data Size Constraints	193
Table 50 RAW Image Data Types	197
Table 51 RAW6 Packet Data Size Constraints.....	198
Table 52 RAW7 Packet Data Size Constraints.....	199
Table 53 RAW8 Packet Data Size Constraints.....	200
Table 54 RAW10 Packet Data Size Constraints.....	201
Table 55 RAW12 Packet Data Size Constraints.....	203
Table 56 RAW14 Packet Data Size Constraints.....	204
Table 57 RAW16 Packet Data Size Constraints.....	206
Table 58 RAW20 Packet Data Size Constraints.....	207
Table 59 RAW24 Packet Data Size Constraints.....	209
Table 60 RAW28 Packet Data Size Constraints.....	211
Table 61 User Defined 8-bit Data Types	213
Table 62 SNS Registers Required to Support OTM.....	236
Table 63 OTM Bitwise Transport Mapping for All Frame Formats	238
Table 64 AOSC Operation CCC and Defining Byte Values.....	239
Table 65 AOSC OTM Low-Latency Mandatory IBI MDB Codes	240
Table 66 Header Definitions for AOSC Smart Transport Mode (STM)	246
Table 67 Status Data Padding	251
Table 68 JPEG8 Additional Marker Codes Listing.....	254
Table 69 Initial Seed Values for Lanes 9 through 32	321
Table 70 ALP Code Definitions used by CSI-2.....	330
Table 71 MPC Bits Per Block and CSI-2 Image Data Format.....	348

Table 72 MPC Compression Modes for Bayer Image Sensors	352
Table 73 Decoder for Compression Mode PD	357
Table 74 Decoder for Compression Mode DGD (Bayer).....	360
Table 75 Decoder for Compression Mode FNR (Bayer)	361
Table 76 Decoder for Compression Mode OUT (Bayer).....	363
Table 77 Decoder for Compression Mode eDGD	365
Table 78 Decoder for Compression Mode ePD	367
Table 79 Decoder for Compression Mode eSHV (Bayer).....	370
Table 80 MPC Compression Modes for Tetra-Cell Image Sensors.....	372
Table 81 Decoder for Compression Mode AD.....	377
Table 82 Decoder for Compression Mode OD.....	379
Table 83 Decoder for Compression Mode FNR (Tetra-Cell).....	385
Table 84 Decoder for Compression Mode OUT (Tetra-Cell)	387
Table 85 Decoder for Compression Mode eMPD (Tetra-Cell)	389
Table 86 Decoder for Compression Mode eHVD (Tetra-Cell)	392
Table 87 Decoder for Compression Mode eHVA	394
Table 88 Decoder for Compression Mode eOUT	397
Table 89 Decoder for Compression Mode OUT (Tetra-Cell) with 10:7 Compression Ratio.....	401
Table 90 MPC Compression Modes for Nona-Cell Image Sensors	404
Table 91 Decoder for Compression Mode AD (Nona-Cell).....	408
Table 92 Decoder for Compression Mode DGD (Nona-Cell)	410
Table 93 Decoder for Compression Mode FNR (Nona-Cell)	412
Table 94 Decoder for Compression Mode eAD (Nona-Cell).....	414
Table 95 Decoder for Compression Mode eHVD (Nona-Cell).....	416
Table 96 Decoder for Compression Mode eHVI	419
Table 97 Decoder for Compression Mode eSHV (Nona-Cell)	420
Table 98 Decoder for Compression Mode eMPD (Nona-Cell).....	423

Release History

Date	Version	Description
29-Nov-2005	v1.00	Initial Board approved release.
09-Nov-2010	v1.01.00	Board approved release.
22-Jan-2013	v1.1	Board approved release.
10-Sep-2014	v1.2	Board approved release.
07-Oct-2014	v1.3	Board approved release.
28-Mar-2017	v2.0	Board approved release.
09-Apr-2018	v2.1	Board approved release.
10-Sep-2019	v3.0	Board approved release.
08-Dec-2021	v4.0	Board approved release.
2022-11-14	v4.0.1	Board approved release.

This page intentionally left blank.

1 Introduction

1.1 Scope

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

This Revision of the Camera Serial Interface 2 Specification (v4.0) leverages both MIPI C-PHY [**MIPI02**] and MIPI D-PHY [**MIPI01**], enabling higher interface bandwidth and more flexibility in channel layout, and maintains backwards compatibility with previous CSI-2 versions with the understanding that the D-PHY and C-PHY physical layers themselves are not interoperable. For the first time, this Revision also supports the transmission of image frames over the MIPI I3C bus [**MIPI03**] for use in ultra-low power, always-on imaging applications.

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

1.2 Purpose

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

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

2 Terminology

2.1 Use of Special Terms

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

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

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

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

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

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

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

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

2.2 Definitions

AOSC Transfer Mode: Either OTM, LTM, or STM; see *Section 13.2*.

CCI (I²C): CCI supporting I²C [*NXP01*].

CCI (I3C): CCI supporting I3C [*MIPI03*].

CCI (I3C SDR) means CCI supporting I3C SDR.

CCI (I3C DDR) means CCI supporting I3C DDR.

Controller: An I²C or I3C Device that is capable of controlling an I²C or I3C Bus, respectively.

Note:

In previous versions of the CSI-2 Specification, a Controller Device was called a “Master Device”. This version of the CSI-2 Specification has deprecated the term “Master”, and now uses the updated normative term “Controller.” Please note that the technical definition of such a Device, and its Role on an I²C or I3C Bus, are unchanged.

Filler: A CSI-2 protocol element that is inserted after CSI-2 Packets in order to ensure that data transmissions on all Lanes end at the same time.

Lane: A unidirectional, point-to-point, 2- or 3-wire interface used for high-speed serial clock or data transmission; the number of wires is determined by the PHY specification in use (i.e. either D-PHY or C-PHY, respectively). A CSI-2 camera interface using the D-PHY physical layer consists of one clock Lane and one or more data Lanes. A CSI-2 camera interface using the C-PHY physical layer consists of one or more Lanes, each of which transmits both clock and data information. Note that when describing features or behavior applying to both D-PHY and C-PHY, this specification sometimes uses the term data Lane to refer to both a D-PHY data Lane and a C-PHY Lane.

Message: In CCI (I²C) or CCI (I3C SDR), a Message begins with a START or Repeated START condition, followed by the address of the Target(s), R/W bit, other data, and ends with either a STOP or Repeated START condition. In the case of CCI (I3C SDR), a START or Repeated START condition followed by 7'h7E may be added to the beginning. In CCI (I3C DDR), a Message begins with either the I3C ENTHDR0 CCC or the I3C HDR Restart Pattern, followed by an HDR-DDR Command, HDR-DDR Data, and ends with either the I3C HDR Exit Pattern or the I3C HDR Restart Pattern.

Multi-Pixel: An N x N pixel structure with the same color filter array for all of its pixels. A Tetra-Cell and a Nona-Cell are both Multi-Pixel structures.

Nona-Cell: A 3 x 3 Multi-Pixel (i.e., 9 pixels). That is, a 3 x 3 pixel structure with the same color filter array on each of its pixels.

Operation: An Operation is composed of one or more Messages in order to read or write.

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

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

Primary: The Primary is the PHY on the side of a Link that is the main data source. The Primary transmits data in the Forward Direction and receives data in the Reverse Direction. In some Bi-directional systems the functions of the Primary and Secondary are nearly equivalent. The difference between being Primary or Secondary is simply that one side is identified as the Primary, and the other side is identified as the Secondary. The side that is defined as the Primary does not change after the system is initialized.

Note:

In previous versions of the CSI-2 Specification, a Primary PHY was called a “Master PHY”. This version of the CSI-2 Specification has deprecated the term “Master”, and now uses the updated normative term “Primary.” Please note that the technical definition of such a PHY, and its Role on a Link, are unchanged.

Secondary: The Secondary is the PHY on the side of a Link that is the main data sink. The Secondary receives data in the Forward Direction and transmits data in the Reverse Direction. In some Bi-directional systems the functions of the Primary and Secondary are nearly equivalent. The difference between being Primary or Secondary is simply that one side is identified as the Primary, and the other side is identified as the Secondary. The side that is defined as the Secondary does not change after the system is initialized.

Note:

In previous versions of the CSI-2 Specification, a Secondary PHY was called a “Slave PHY”. This version of the CSI-2 Specification has deprecated the term “Slave”, and now uses the updated normative term “Secondary.” Please note that the technical definition of such a PHY, and its Role on a Link, are unchanged.

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

Spacer: An optional CSI-2 protocol element that may be inserted after CSI-2 Packets and Fillers transmitted using CSI-2 LRTE; not to be confused with the C-PHY “Spacer Code” defined in [MIPI02].

Target: A Device on an I²C or I3C Bus that can only respond to commands from a Controller.

Note:

In previous versions of the CSI-2 Specification, a Target Device was called a “Slave Device”. This version of the CSI-2 Specification has deprecated the term “Slave”, and now uses the updated normative term “Target.” Please note that the technical definition of such a Device, and its Role on an I²C or I3C Bus, are unchanged.

Tetra-Cell: A 2 x 2 Multi-Pixel (i.e., 4 pixels). That is, a 2 x 2 pixel structure with the same color filter array on each of its pixels.

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

111 **Virtual Channel:** Multiple independent data streams for up to 32 peripherals are supported by this
112 Specification. The data stream for each peripheral may be a Virtual Channel. These data streams may be
113 interleaved and sent as sequential packets, with each packet dedicated to a particular peripheral or channel.
114 Packet protocol includes information that links each packet to its intended peripheral.

2.3 Abbreviations

115	APP	AOSC host processor
116	e.g.	For example (Latin: exempli gratia)
117	i.e.	That is (Latin: id est)
118	SNS	AOSC peripheral device

2.4 Acronyms

119	ALP	Alternate Low Power
120	AOSC	Always-On Sentinel Conduit
121	BER	Bit Error Rate
122	CCI	Camera Control Interface
123	CIL	Control and Interface Logic
124	CRC	Cyclic Redundancy Check
125	CSF	Continuously Streaming Frames
126	CSI	Camera Serial Interface
127	CSPS	Chroma Shifted Pixel Sampling
128	DDR	Dual Data Rate
129	DI	Data Identifier
130	DT	Data Type
131	ECC	Error Correction Code
132	EoT	End of Transmission
133	EoTp	End of Transmission short packet
134	EPD	Efficient Packet Delimiter (PHY and / or Protocol generated signaling used in LRTE)
135	EXIF	Exchangeable Image File Format
136	FE	Frame End
137	FOV	Field of View
138	FS	Frame Start
139	HS	High-Speed; identifier for operation mode
140	HS-LPS-LS	High-Speed to Low Power State to High-Speed switching (includes LPS entry and exit latencies)
141		
142	HS-RX	High-Speed Receiver
143	HS-TX	High-Speed Transmitter
144	I ² C	Inter-Integrated Circuit <i>[NXP01]</i>
145	ILR	Interpacket Latency Reduction
146	JFIF	JPEG File Interchange Format
147	JPEG	Joint Photographic Expert Group
148	LE	Line End
149	LFSR	Linear Feedback Shift Register
150	LLP	Low Level Protocol
151	LP	Low-Power; identifier for operation mode
152	LP-RX	Low-Power Receiver (Large-Swing Single Ended)

153	LP-TX	Low-Power Transmitter (Large-Swing Single Ended)
154	LRTE	Latency Reduction Transport Efficiency
155	LS	Line Start
156	LSB	Least Significant Bit
157	LSS	Least Significant Symbol
158	LTM	Legacy Transport Mode
159	LVLP	Low Voltage Low Power
160	MPC	Multi-Pixel Compression
161	MSB	Most Significant Bit
162	MSS	Most Significant Symbol
163	ODF	On-Demand Frame
164	OTM	Optimal Transfer Mode
165	PDQ	Packet Delimiter Quick (PHY generated and consumed signaling used in LRTE)
166	PF	Packet Footer
167	PH	Packet Header
168	PHY	Physical Layer
169	PI	Packet Identifier
170	PPI	PHY Protocol Interface
171	PRBS	Pseudo-Random Binary Sequence
172	PT	Packet Type
173	RGB	Color representation (Red, Green, Blue)
174	RX	Receiver
175	SCL	Serial Clock (for CCI)
176	SDA	Serial Data (for CCI)
177	SLM	Sleep Mode
178	SoT	Start of Transmission
179	STM	Smart Transport Mode
180	TX	Transmitter
181	ULPS	Ultra Low Power State
182	VDSP	Vision Digital Signal Processor\
183	VGA	Video Graphics Array
184	YUV	Color representation (Y for luminance, U & V for chrominance)

3 References

- 185 [NXP01] UM10204, *I²C bus specification and user manual*, Revision 6,
186 NXP Semiconductors N.V., 4 April 2014.
- 187 [MIPI01] *MIPI Alliance Specification for D-PHY*, version 3.0, MIPI Alliance, Inc., 8 June 2021.
- 188 [MIPI02] *MIPI Alliance Specification for C-PHY*, version 2.1, MIPI Alliance, Inc., 1 April 2021.
- 189 [MIPI03] *MIPI Alliance Specification for I3C (Improved Inter-Integrated Circuit)*, version 1.1.1,
190 MIPI Alliance, Inc., 8 June 2021.
- 191 [MIPI04] *MIPI Alliance Specification for Camera Command Set (CCS)*, version 1.1,
192 MIPI Alliance, Inc., 12 December 2019.
- 193 [MIPI05] *MIPI Alliance Specification for Camera Service Extensions (CSE)*, version 1.0,
194 MIPI Alliance, Inc., 27 July 2021.
- 195 [MIPI06] *MIPI Alliance Specification for A-PHY*, version 1.0, MIPI Alliance, Inc., 6 August 2020.
- 196 [MIPI07] *MIPI Multi Pixel Compression (MPC) Example Code*,
197 <<https://members.mipi.org/wg/All-Members/document/folder/14072>>,
198 MIPI Alliance, Inc., last accessed 8 December 2021.

This page intentionally left blank.

4 Overview of CSI-2

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

The first option, referred to in this specification as the “D-PHY physical layer option,” is typically a unidirectional differential interface with one 2-wire clock Lane and one or more 2-wire data Lanes. The physical layer of this interface is defined by the *MIPI Alliance Specification for D-PHY [MIP101]*. **Figure 1** illustrates the connections for this option between a CSI-2 transmitter and receiver, which typically are a camera module and a receiver module, part of the mobile phone engine.

The second high-speed data transmission interface option, referred to in this specification as the “C-PHY physical layer option,” typically consists of one or more unidirectional 3-wire serial data Lanes, each of which has its own embedded clock. The physical layer of this interface is defined by the *MIPI Alliance Specification for C-PHY [MIP102]*. **Figure 2** illustrates the CSI transmitter and receiver connections for this option.

The Camera Control Interface (CCI) for both physical layer options is a bi-directional control interface compatible with the I²C standard **[NXP01]** and/or the MIPI I3C Specification **[MIP103]**.

Note that beginning with the CSI-2 v3.0 specification, Lane 1 of a D-PHY or C-PHY link interconnecting a camera with a host or application processor (i.e., Data1+ / Data1- in **Figure 1**, or Data1_A / Data1_B / Data1_C in **Figure 2**) is permitted to be bidirectional. For such links, there is no requirement to support a physically separate CCI. See **Section 9.12**.

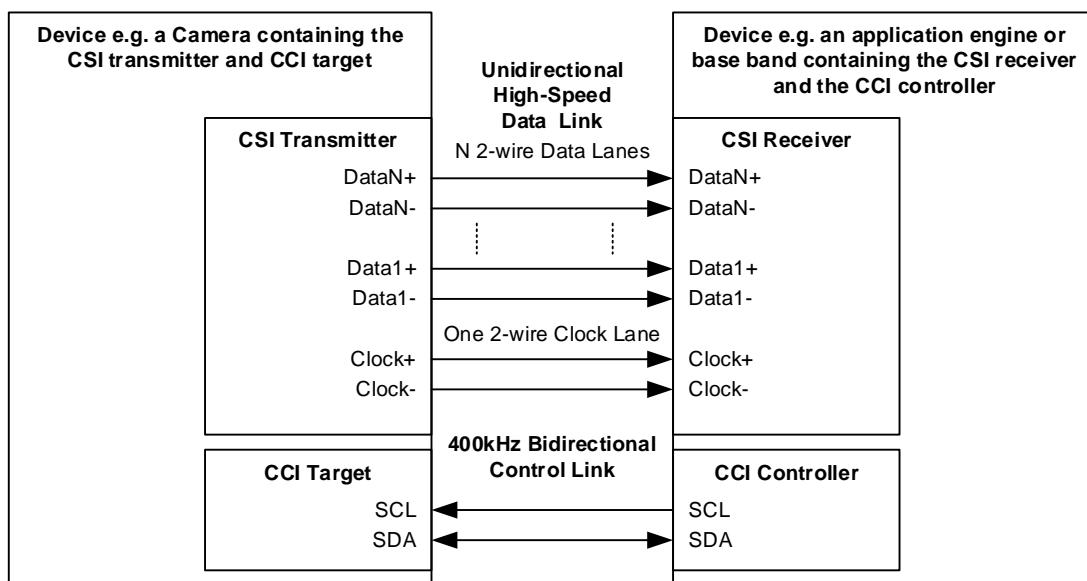
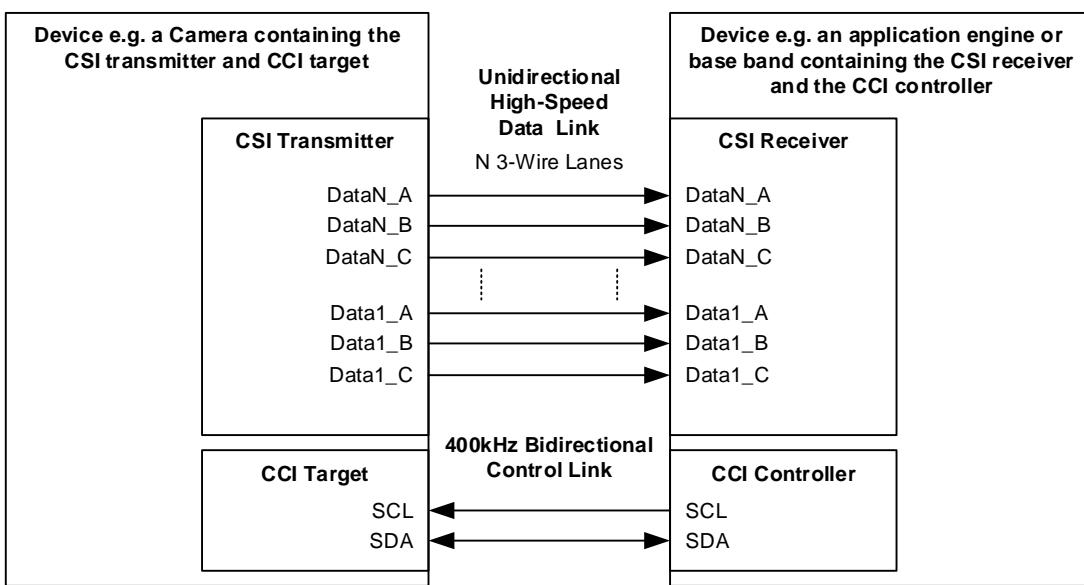


Figure 1 Typical CSI-2 and CCI Transmitter and Receiver Interface for D-PHY



218

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

5 CSI-2 Layer Definitions

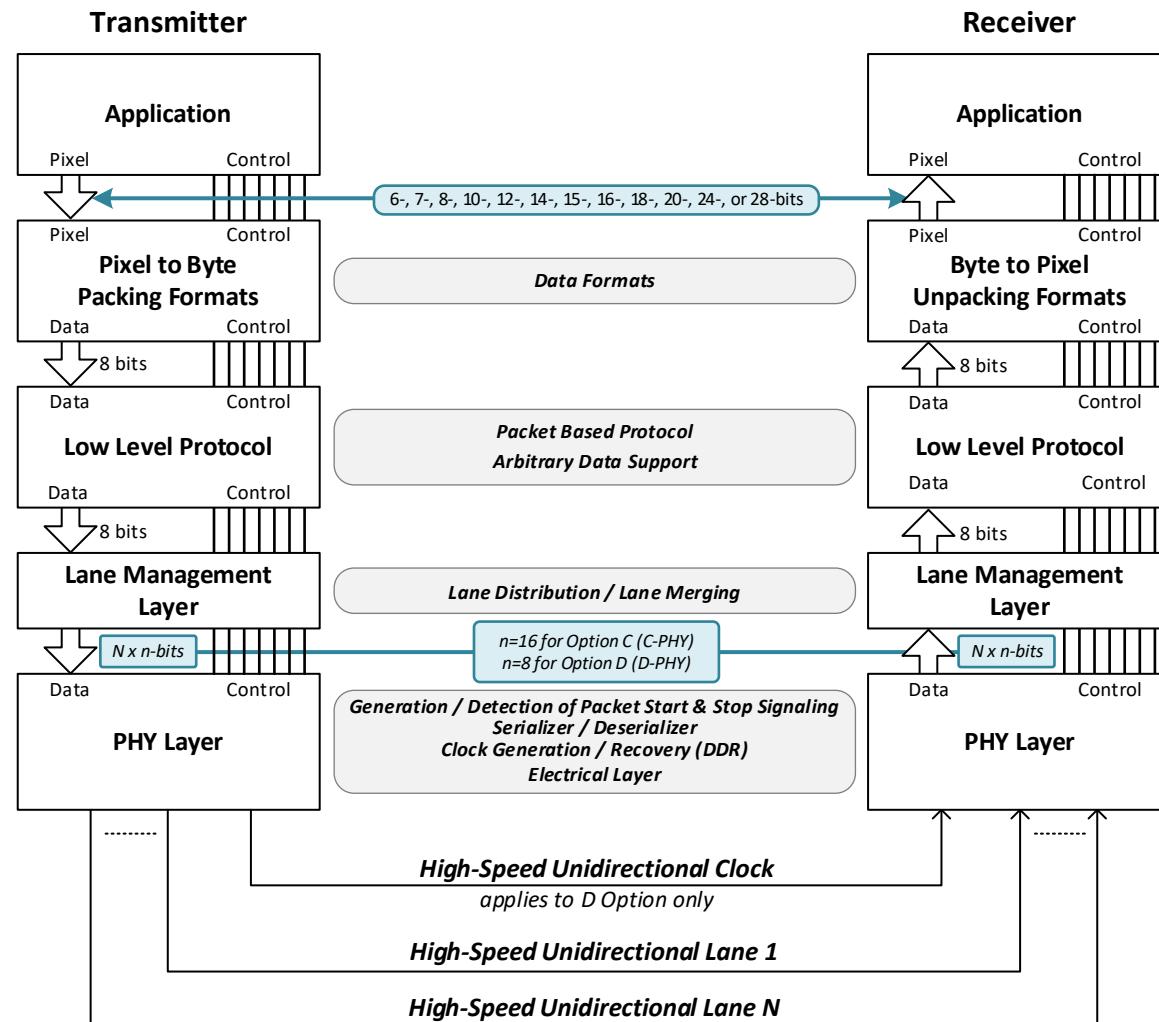


Figure 3 CSI-2 Layer Definitions

Figure 3 defines the conceptual layer structure typically used in CSI-2. The layers can be characterized as follows:

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

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

The PHY layer is described in [\[MIPI01\]](#) and [\[MIPI02\]](#).

232 • **Protocol Layer.** The Protocol layer is composed of several layers, each with distinct
233 responsibilities. The CSI-2 protocol enables multiple data streams using a single interface on the
234 host processor. The Protocol layer specifies how multiple data streams may be tagged and
235 interleaved so each data stream can be properly reconstructed.

236 • **Pixel/Byte Packing/Unpacking Layer.** The CSI-2 specification supports image applications
237 with varying pixel formats. In the transmitter this layer packs pixels from the Application layer
238 into bytes before sending the data to the Low Level Protocol layer. In the receiver this layer
239 unpacks bytes from the Low Level Protocol layer into pixels before sending the data to the
240 Application layer. Eight bits per pixel data is transferred unchanged by this layer.

241 • **Low Level Protocol.** The Low Level Protocol (LLP) includes the means of establishing bit-
242 level and byte-level synchronization for serial data transferred between SoT (Start of
243 Transmission) and EoT (End of Transmission) events and for passing data to the next layer. The
244 minimum data granularity of the LLP is one byte. The LLP also includes assignment of bit-value
245 interpretation within the byte, i.e. the “Endian” assignment.

246 • **Lane Management.** CSI-2 is Lane-scalable for increased performance. The number of data
247 Lanes is not limited by this specification and may be chosen depending on the bandwidth
248 requirements of the application. The transmitting side of the interface distributes (“distributor”
249 function) bytes from the outgoing data stream to one or more Lanes. On the receiving side, the
250 interface collects bytes from the Lanes and merges (“merger” function) them together into a
251 recombined data stream that restores the original stream sequence. For the C-PHY physical
252 layer option, this layer exclusively distributes or collects byte pairs (i.e. 16-bits) to or from the
253 data Lanes. Scrambling on a per-Lane basis is an optional feature, which is specified in detail in
254 *Section 9.17*.

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

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

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

6 Camera Control Interface (CCI)

265 CCI is a two-wire, bi-directional, half duplex, serial interface for controlling the transmitter. CCI is
266 compatible with I²C Fast-mode (Fm) or Fast-mode Plus (Fm+) **[NXP01]** variants, and with the I3C **[MIPI03]**
267 interface's Single Data Rate (SDR) or Double Data Rate (DDR) protocols. CCI shall support up to 400kbps
268 (Fm) operation and 7-bit Target addressing. In addition, CCI can optionally support up to 1Mbps (Fm+),
269 12.5Mbps (SDR), or 25Mbps (DDR).

270 This Section uses the following terms:

- 271 • **CCI (I²C)** means CCI supporting I²C
- 272 • **CCI (I3C)** means CCI supporting I3C
- 273 • **CCI (I3C SDR)** means CCI supporting I3C SDR
- 274 • **CCI (I3C DDR)** means CCI supporting I3C DDR
- 275 • **CCI** alone (without following parentheses) means both **CCI (I²C)** and **CCI (I3C)**.

276 CCI can be used with or without CSI-2 over C/D-PHY. When CCI is used as part of a CSI-2 bus, a CSI-2
277 receiver shall be configured as a Controller and a CSI-2 transmitter shall be configured as a Target. When
278 CCI is used without CSI-2 over C/D-PHY, the host should be used as a Controller. CCI is capable of handling
279 multiple Targets on the bus.

280 In **CCI (I²C)**, multi-controller mode is not supported. Any I²C commands not described in this section shall
281 be ignored, and shall not cause unintended device operation.

282 In **CCI (I3C)**, any I3C mandatory functions and 'Required' CCC commands shall be supported, and any I3C
283 optional functions and commands may be supported (e.g., Multi-Controller, In-Band Interrupt, Hot-Join).

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

285 CCI is a subset of the I²C or I3C protocol that includes the minimum combination of obligatory features for
286 I²C/I3C Target devices specified in the I²C or I3C specification. Therefore, transmitters complying with the
287 CCI specification can also be connected to the system I²C or I3C bus. However, care must be taken so that
288 I²C or I3C Controllers do not attempt to use I²C or I3C features not supported by CCI Controllers or Targets.

289 A CCI transmitter may have additional features to support I²C or I3C, but that is implementation-dependent.
290 Further details can be found on a particular device's data sheet.

291 This specification does not attempt to define the contents of control Messages sent by the CCI Controller.
292 Therefore, it is the responsibility of the implementer to define a set of control Messages and corresponding
293 frame timing and any I²C or I3C latency requirements that the CCI Controller must meet when sending such
294 control Messages to the CCI Target.

295 CCI defines an additional data protocol layer on top of I²C or I3C, as specified in the following sections.

6.1 CCI (I²C) Data Transfer Protocol

The **CCI (I²C)** data transfer protocol follows the I²C specification. The START, REPEATED START, and STOP conditions, and the data transfer protocol, are all specified in *[NXP01]*.

6.1.1 CCI (I²C) Message Type

A basic **CCI (I²C)** Message consists of:

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

And then either:

- For a write operation:
 - Data byte from Controller
 - Acknowledge/negative acknowledge from Target, and
 - STOP or Repeated START condition

Or:

- For a read operation:
 - Repeated START condition (not used in Single Read from Current Location)
 - Target address with read bit (not used in Single Read from Current Location)
 - acknowledge signal from Target (not used in Single Read from Current Location)
 - data byte from the Target
 - acknowledge or negative acknowledge from the Controller, and
 - STOP or Repeated START condition.

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

The Target address in **CCI (I²C)** is 7 bits long.

CCI (I²C) supports an 8-bit INDEX with 8-bit data, or a 16-bit INDEX with 8-bit data. The Target device in question defines what Message type is used.

6.1.2 CCI (I²C) Read/Write Operations

A CCI (I²C) compatible device shall support the four read operations and two write operations shown in *Table 1*, as detailed in the following sub-sections:

Table 1 CCI (I²C) Read/Write Operations

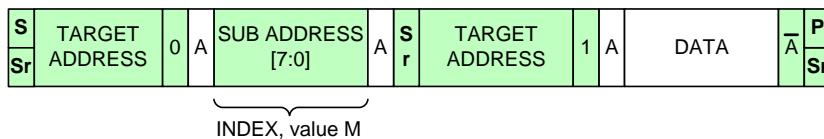
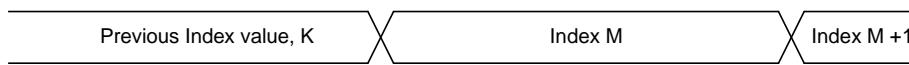
Type	Operation	Section
Read	Single Read from Random Location	6.1.2.1
	Sequential Read from Random Location	6.1.2.2
	Single Read from Current Location	6.1.2.3
	Sequential Read from Current Location	6.1.2.4
Write	Single Write to Random Location	6.1.2.5
	Sequential Write Starting from Random Location	6.1.2.6

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

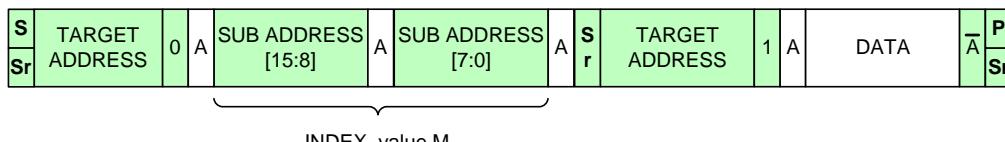
6.1.2.1 CCI (I²C) Single Read from Random Location

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

CCI (I²C) Single Read from Random Location with 8-bit index and 8-bit data (7-bit address)



CCI (I²C) Single Read from Random Location with 16-bit index and 8-bit data (7-bit address)



From Target to Controller S = START condition A = Acknowledge

From Controller to Target P = STOP condition = Negative acknowledge

Sr = REPEATED START condition

Figure 4 CCI (I²C) Single Read from Random Location

6.1.2.2 CCI (I²C) Single Read from Current Location

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

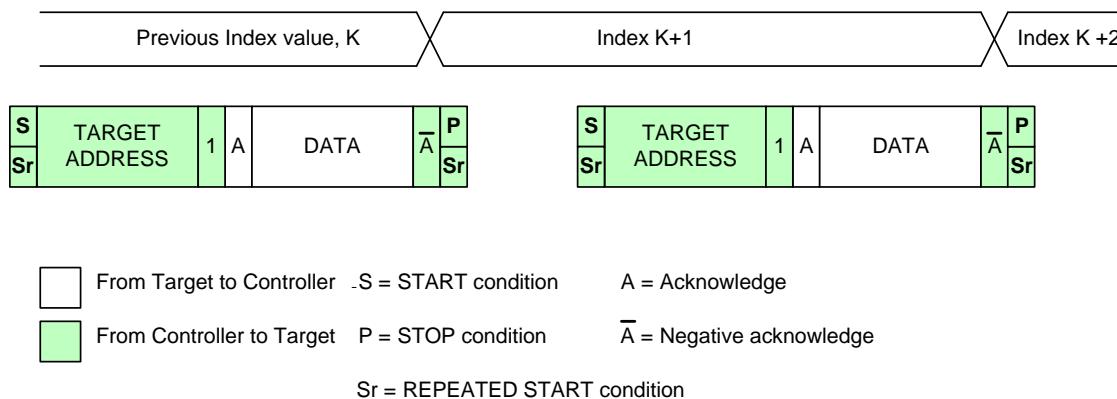


Figure 5 CCI (I²C) Single Read from Current Location

6.1.2.3 CCI (I²C) Sequential Read Starting from Random Location

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

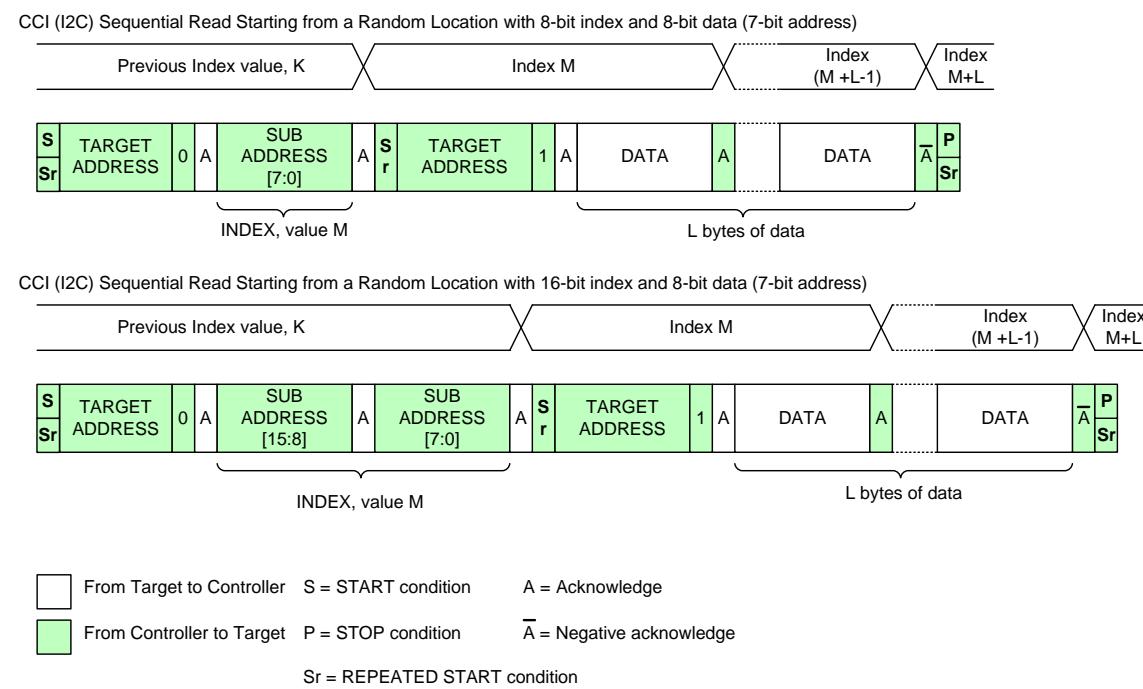
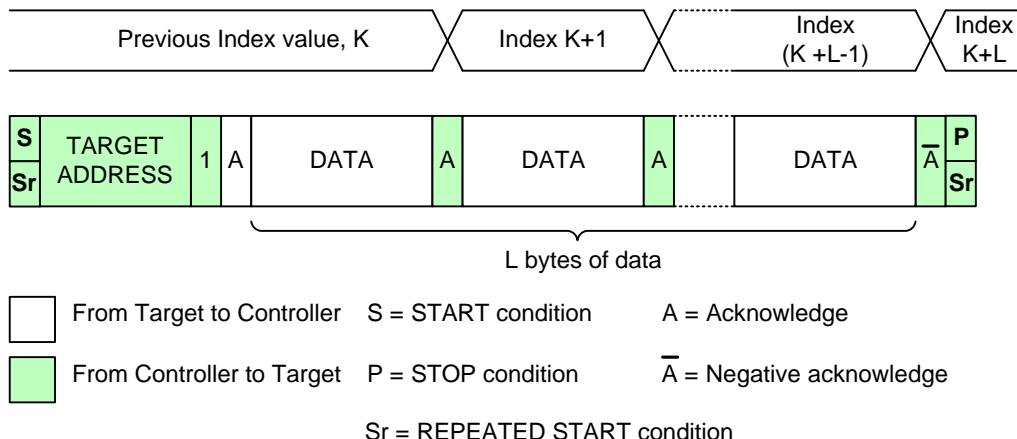


Figure 6 CCI (I²C) Sequential Read Starting from Random Location

6.1.2.4 CCI (I²C) Sequential Read Starting from Current Location

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

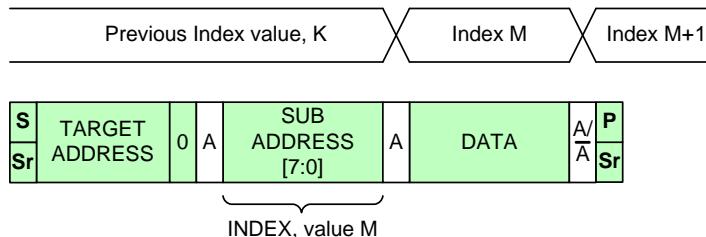
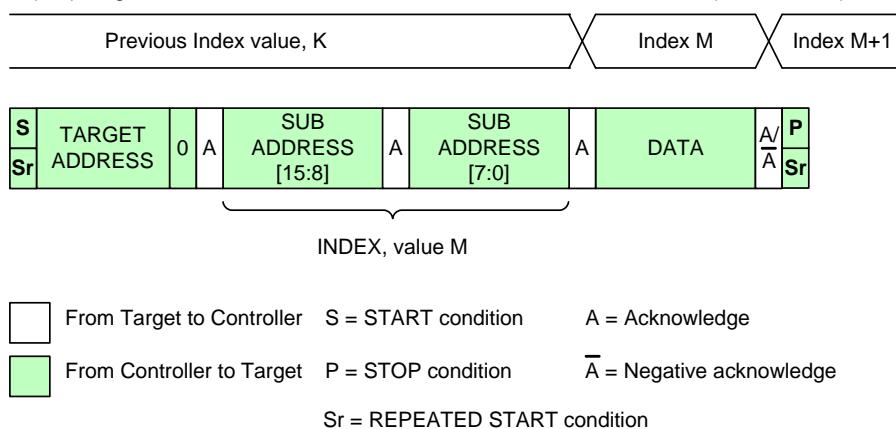


348

Figure 7 CCI (I²C) Sequential Read Starting from Current Location

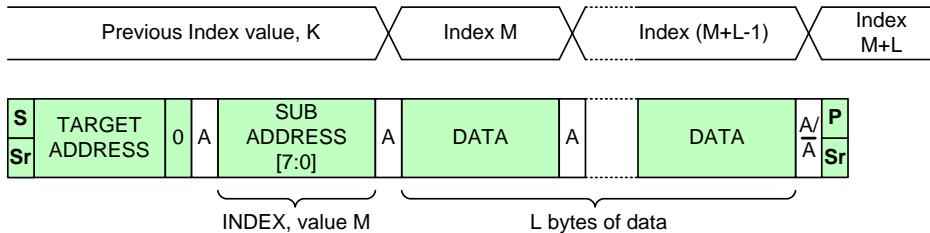
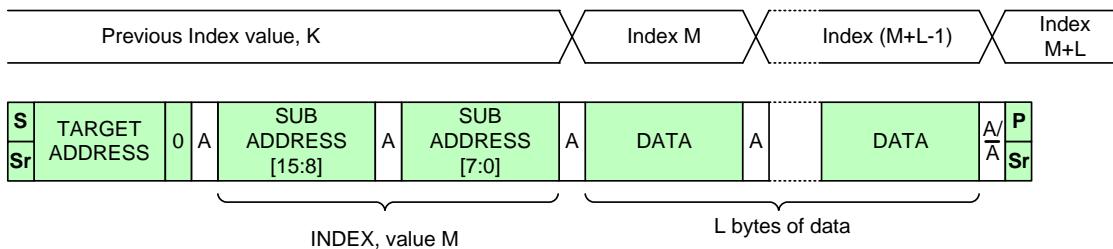
6.1.2.5 CCI (I²C) Single Write to Random Location

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

CCI (I²C) Single Write to a Random Location with 8-bit index and 8-bit data (7-bit address)CCI (I²C) Single Write to a Random Location with 16-bit index and 8-bit data (7-bit address)**Figure 8 CCI (I²C) Single Write to Random Location**

6.1.2.6 CCI (I²C) Sequential Write Starting from Random Location

353
354
355 The Sequential Write Starting from Random Location operation is illustrated in **Figure 9**. The Target auto-increments the INDEX after each data byte is received. The Sequential Write Starting from Random Location operation is terminated with a STOP or Repeated START condition from the Controller.

CCI (I²C) Sequential Write Starting from a Random Location with 8-bit index and 8-bit data (7-bit address)CCI (I²C) Sequential Write Starting from a Random Location with 16-bit index and 8-bit data (7-bit address)

- From Target to Controller S = START condition A = Acknowledge
- From Controller to Target P = STOP condition \bar{A} = Negative acknowledge
- Sr = REPEATED START condition

356

Figure 9 CCI (I²C) Sequential Write Starting from Random Location

6.2 CCI (I3C) Data Transfer Protocol

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

If **CCI (I3C)** is supported, then **CCI (I3C SDR)** shall be supported and **CCI (I3C DDR)** may be supported. The Controller shall get the Target's Max Read Length (MRL) and Max Write Length (MWL) via transmitting I3C CCCs GETMRL and GETMWL prior to **CCI (I3C)** data transfer.

6.2.1 CCI (I3C SDR) Data Transfer Protocol

6.2.1.1 CCI (I3C SDR) Message Type

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

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

- START condition
- 7'h7E with write bit
- Acknowledge from Target
- Repeated START condition
- Target address with read/write bit
- Acknowledge from Target
- Sub-address (INDEX) of a register inside the Target device (not used in Single Read from Current Location)
- Transition bit (Parity bit) from Controller (not used in Single Read from Current Location)

And then either:

- For a write operation:
 - Data byte from Controller
 - Transition bit (Parity bit) from Controller
 - STOP or Repeated START condition;

Or

- For a read operation:
 - Repeated START condition (not used in Single Read from Current Location)
 - Target address with read bit (not used in Single Read from Current Location)
 - Acknowledge from Target (not used in Single Read from Current Location)
 - Data byte from Target
 - Transition bit (End-of-Data) from Controller or Target
 - STOP or Repeated START condition.

387 Other **CCI (I3C SDR)** Messages starting a Message with a Target address consist of:
 388 • START or Repeated START condition
 389 • Target address with read/write bit
 390 • Acknowledge from Target
 391 • Sub-address (INDEX) of a register inside the Target device (not used in Single Read from Current
 392 Location)
 393 • Transition bit (Parity bit) from Controller (not used in Single Read from Current Location)

394 And then either:

- 395 • For a write operation:
 396 • Data byte from Controller
 397 • Transition bit (Parity bit) from Controller
 398 • STOP or Repeated START condition;

399 Or:

- 400 • For a read operation:
 401 • Repeated START condition (not used in Single Read from Current Location)
 402 • Target address with read bit (not used in Single Read from Current Location)
 403 • Acknowledge from Target (not used in Single Read from Current Location)
 404 • Data byte from Target
 405 • Transition bit (End-of-Data) from Controller or Target
 406 • STOP or Repeated START condition.

407 The Target address in **CCI (I3C SDR)** is 7 bits long.

408 **CCI (I3C SDR)** supports an 8-bit INDEX with 8-bit data, or a 16-bit INDEX with 8-bit data. The Target
 409 device in question defines what Message type is used.

6.2.1.2 CCI (I3C SDR) Read/Write Operations

410 A CCI (I3C SDR) compatible device shall support the four read operations and two write operations shown
 411 in *Table 2*, as detailed in the following sub-sections:

412 **Table 2 CCI (I3C SDR) Read/Write Operations**

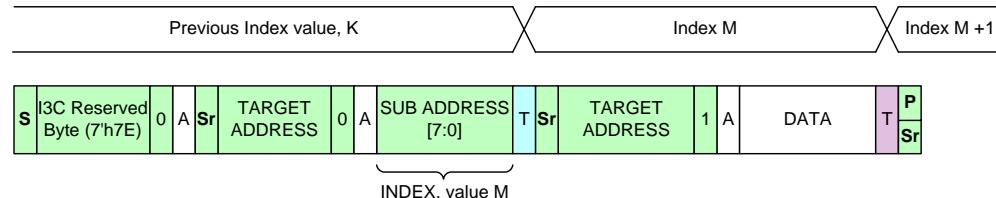
Type	Operation	Section
Read	Single Read from Random Location	6.2.1.2.1
	Single Read from Current Location	6.2.1.2.2
	Sequential Read from Random Location	6.2.1.2.3
	Sequential Read from Current Location	6.2.1.2.4
Write	Single Write to Random Location	6.2.1.2.5
	Sequential Write Starting from Random Location	6.2.1.2.6

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

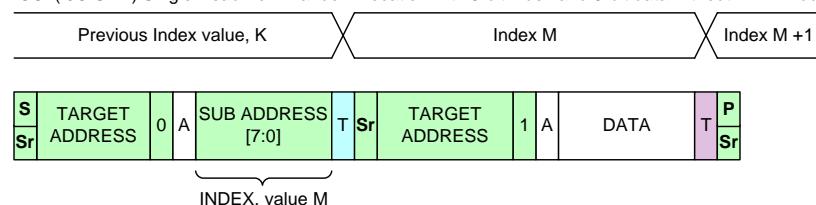
6.2.1.2.1 CCI (I3C SDR) Single Read from Random Location

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

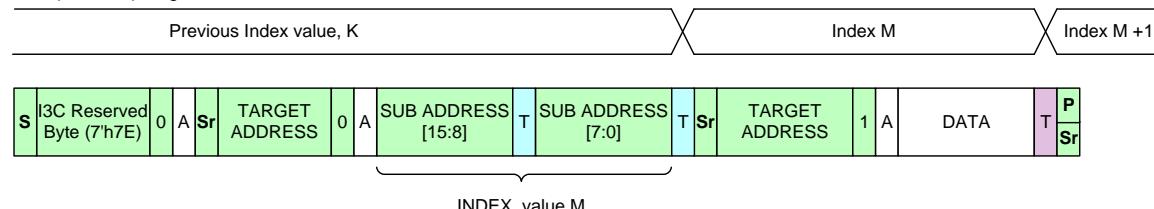
CCI (I3C SDR) Single Read from Random Location with 8-bit index and 8-bit data with 7'h7E Address



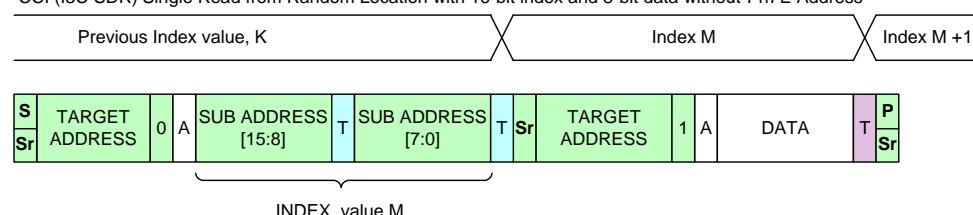
CCI (I3C SDR) Single Read from Random Location with 8-bit index and 8-bit data without 7'h7E Address



CCI (I3C SDR) Single Read from Random Location with 16-bit index and 8-bit data with 7'h7E Address



CCI (I3C SDR) Single Read from Random Location with 16-bit index and 8-bit data without 7'h7E Address



- From Target to Controller
- From Controller to Target
- Transition Bit (Parity Bit for Write Data)
- Transition Bit (End-of-Data for Read Data)

S = START condition
Sr = Repeated START condition
P = STOP condition
A = Acknowledge
T = Transition Bit alternative to ACK/NACK

419

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

6.2.1.2.2 CCI (I3C SDR) Single Read from Current Location

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

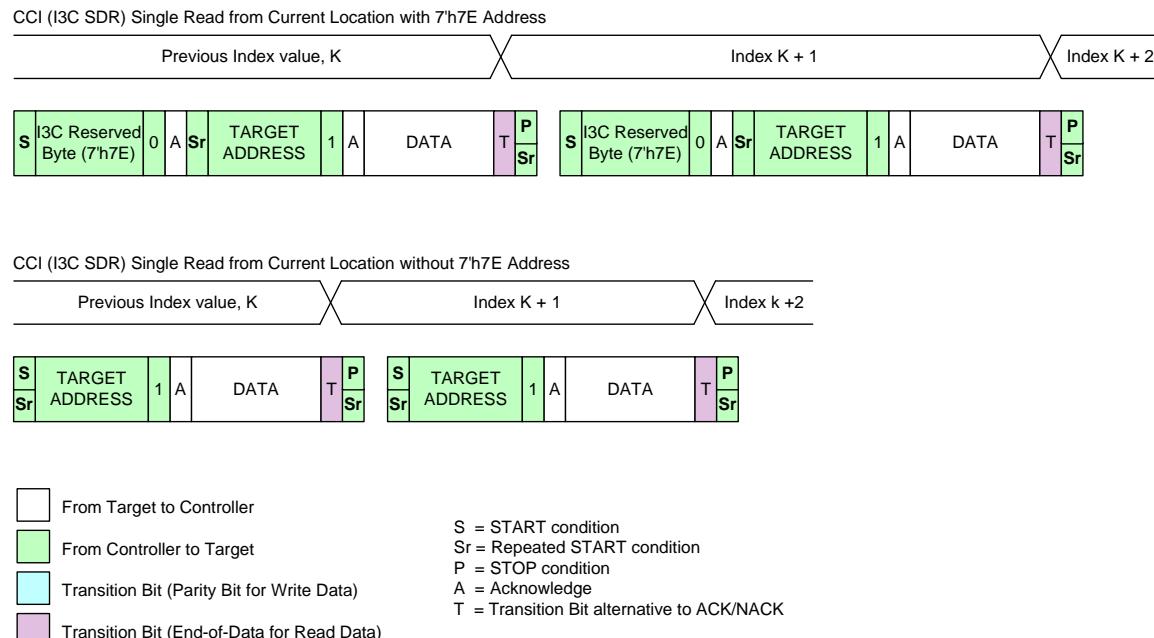


Figure 11 CCI (I3C SDR) Single Read from Current Location

6.2.1.2.3 CCI (I3C SDR) Sequential Read from Random Location

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

Note:

When selecting a suitable value for MRL, the designer of the Target device and the system designer should take into account the needs of the payload that the CCI will carry. For example, in the CCS Data Transfer Interface [MIPI04], it is beneficial to support an MRL of 64 bytes or larger (i.e. 64 bytes for Data payload).

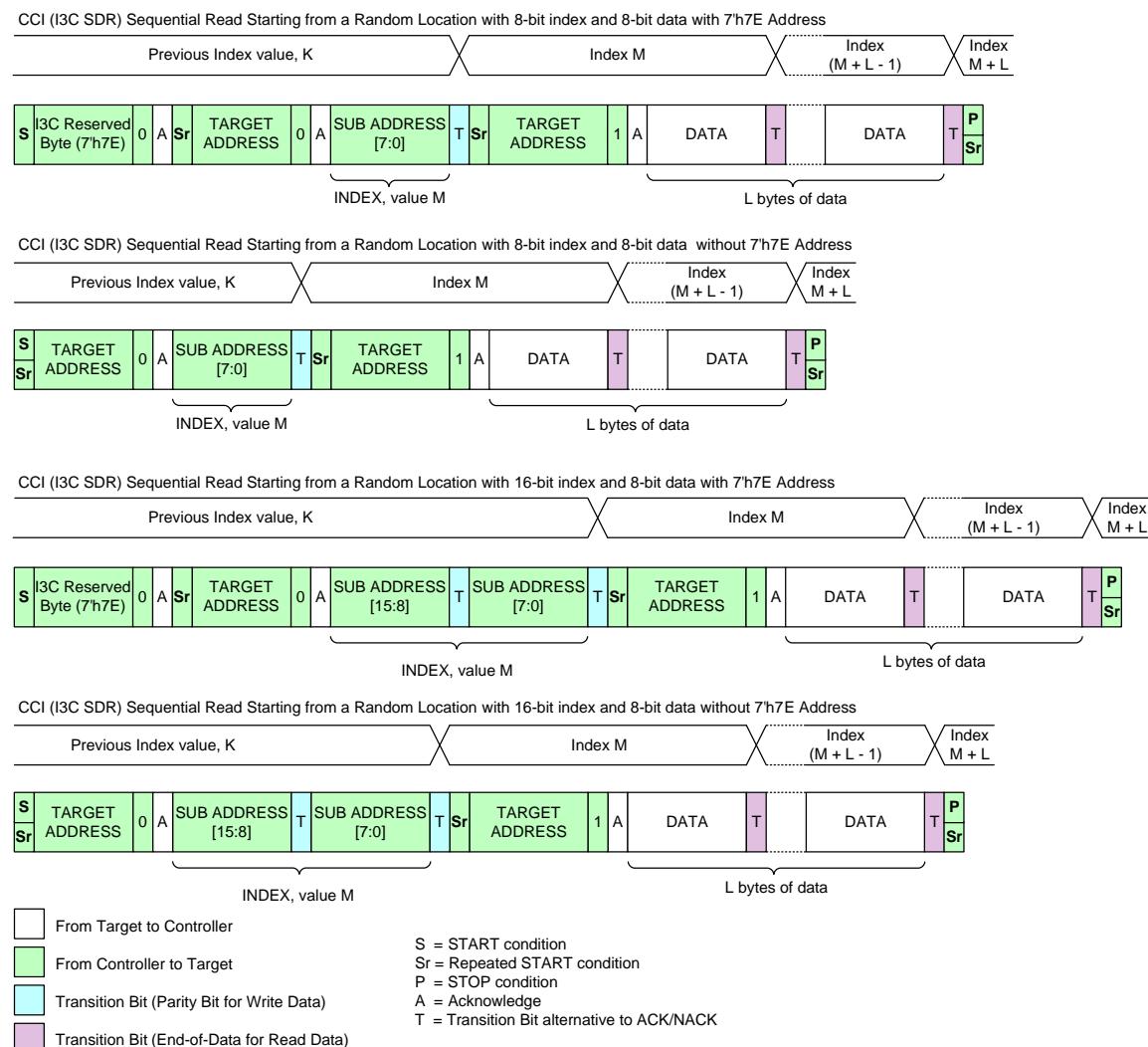


Figure 12 CCI (I3C SDR) Sequential Read Starting from Random Location

6.2.1.2.4 CCI (I3C SDR) Sequential Read from Current Location

A sequential read starting from the current location (*Figure 13*) is similar to a sequential read from a random location. The only exception is when there is no dummy write operation. The Controller or Target terminates a read transaction by setting the transition bit, and then issues a STOP or Repeated START condition.

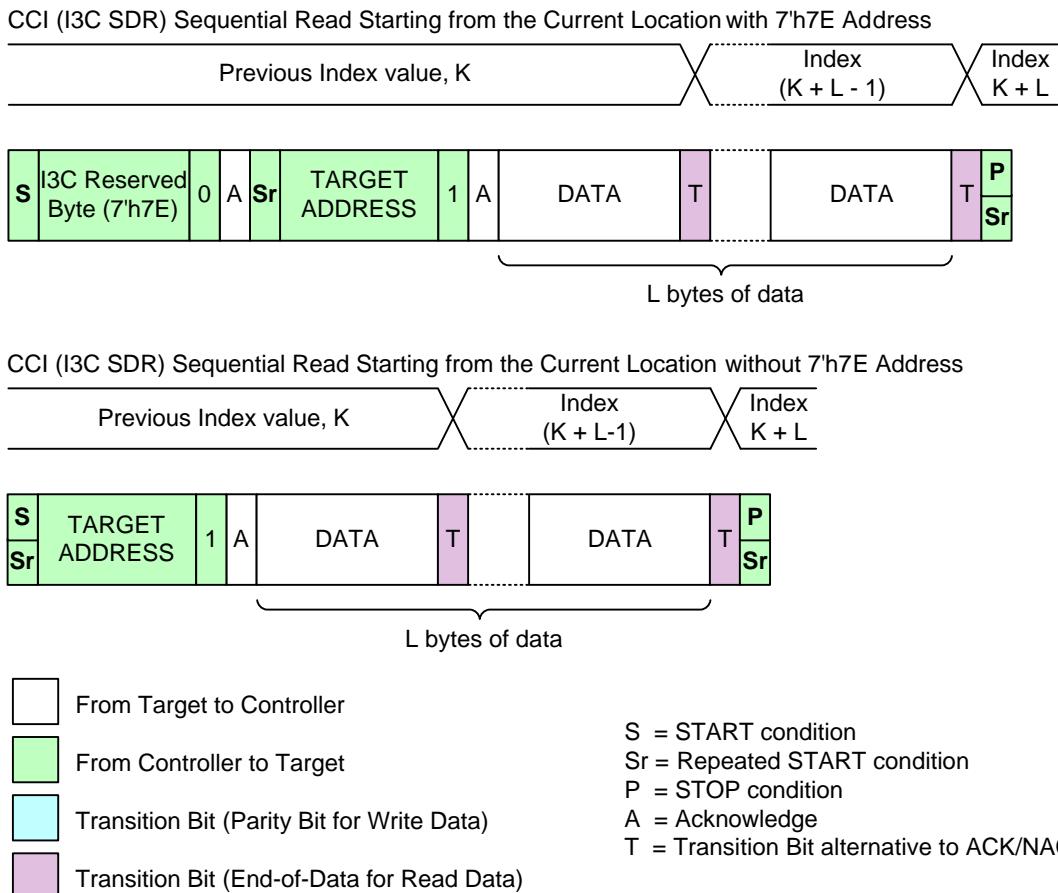
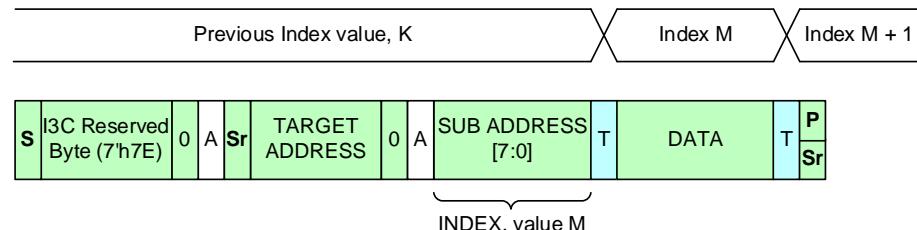


Figure 13 CCI (I3C SDR) Sequential Read Starting from Current Location

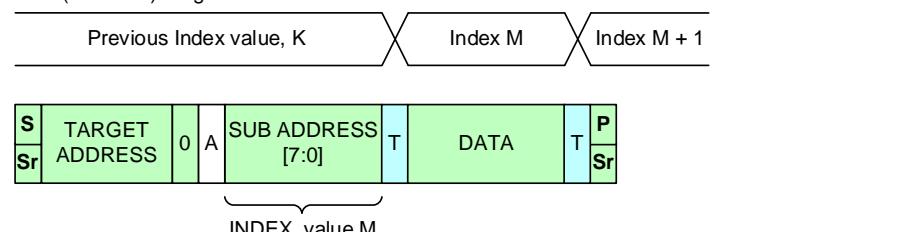
6.2.1.2.5 CCI (I3C SDR) Single Write to Random Location

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

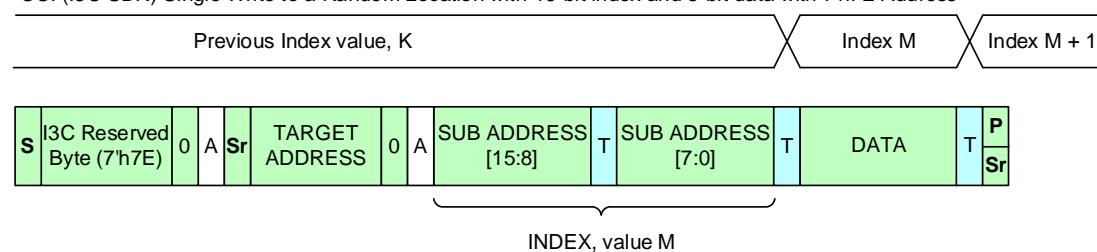
CCI (I3C SDR) Single Write to a Random Location with 8-bit index and 8-bit data with 7'h7E Address



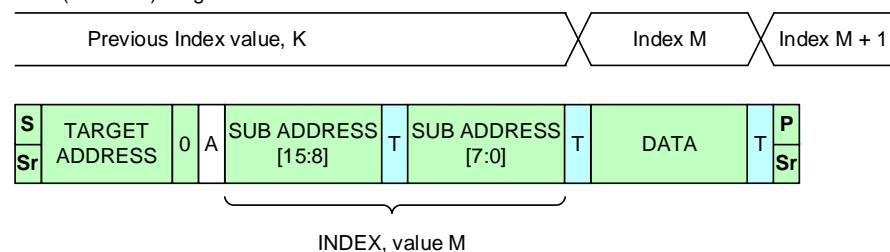
CCI (I3C SDR) Single Write to a Random Location with 8-bit index and 8-bit data without 7'h7E Address



CCI (I3C SDR) Single Write to a Random Location with 16-bit index and 8-bit data with 7'h7E Address



CCI (I3C SDR) Single Write to a Random Location with 16-bit index and 8-bit data without 7'h7E Address



From Target to Controller

S = START condition

From Controller to Target

Sr = Repeated START condition

Transition Bit (Parity Bit for Write Data)

P = STOP condition

Transition Bit (End-of-Data for Read Data)

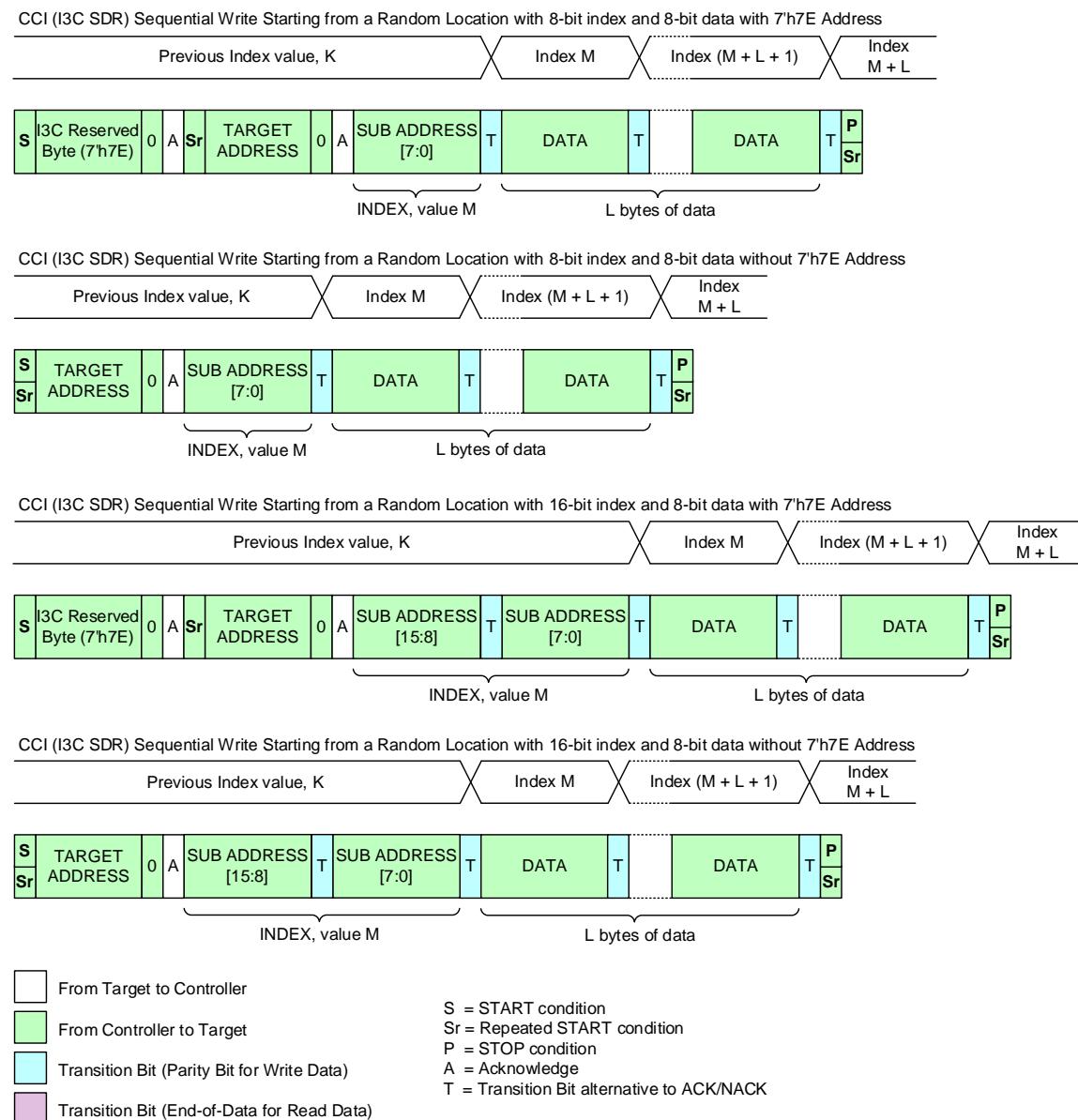
A = Acknowledge

T = Transition Bit alternative to ACK/NACK

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

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

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



450

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

6.2.2 CCI (I3C DDR) Data Transfer Protocol

6.2.2.1 CCI (I3C DDR) Message Type

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

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

CCI (I3C DDR) supports either:

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

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

or:

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

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

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

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

Examples

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

When a multi-byte register is accessed via **CCI (I3C DDR)**, the transmission byte order described in [Section 6.6](#) shall be the same as for **CCI (I²C)** and **CCI (I3C SDR)**.

Example

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

6.2.2.2 CCI (I3C DDR) Read/Write Operations

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

Table 3 CCI (I3C DDR) Read/Write Operations

Type	Operation	Section
Read	Sequential Read from Random Location	6.2.2.2.2
	Concatenated Sequential Read from Random Location	6.2.2.2.3
Write	Sequential Write Starting from Random Location	6.2.2.2.4

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

6.2.2.2.1 CCI (I3C DDR) Command Definitions

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

This single Command Code is sufficient, because the Target can still distinguish between three different R/W operations. Consider the example of 16-bit LENGTH and 16-bit INDEX:

- If the Target receives a Data Word greater than 4 bytes, then the operation is “Sequential Write Starting from Random Location”.
- If the Target receives a Data Word of 4 bytes before the HDR Restart Pattern, then there are two possibilities:
 - If the value of the LENGTH field is \leq MRL–1, then the operation is “Sequential Read Starting from a Random Location”.
 - If the value of the LENGTH field is $>$ MRL–1, then the operation is “Concatenated Sequential Read Starting from a Random Location”.

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

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

495 **Table 4 CCI (I3C DDR) Read/Write Operation Command Codes**

Type	Operation	Command Code Position	R/W Bit and Command Code See Note 1	Section
Write	Sequential Write Starting from Random Location	Command Word	0x00	6.2.2.2.4
Read	Sequential Read Starting from Random Location	Command Word for LENGTH & INDEX	0x00	6.2.2.2
		Command Word for ReadData	0x80	
	Concatenated Sequential Read Starting from Random Location	Command Word for LENGTH & INDEX	0x00	6.2.2.3
		Command Word for ReadData	0x80	

Note:

1. In all five cases, the 7-bit Command Code in the low seven bits is 7'b0000000. Only the R/W bit, which is the high bit of the byte, changes.

6.2.2.2.2 CCI (I3C DDR) Sequential Read From Random Location

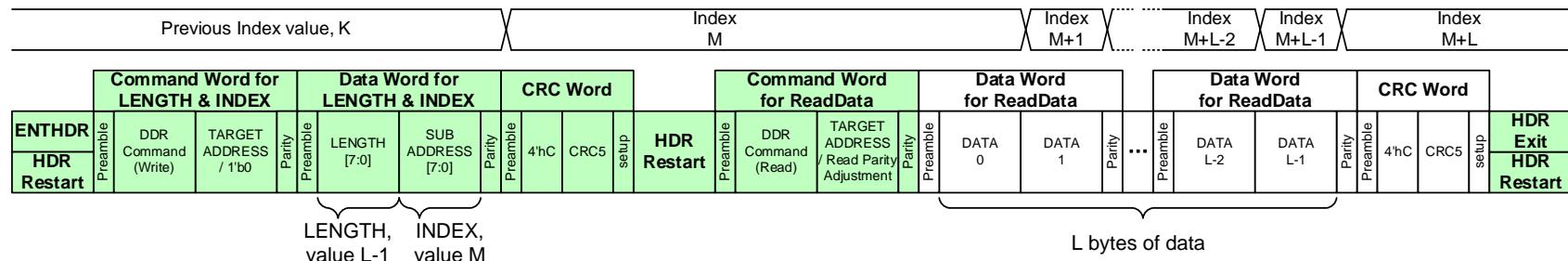
496 In a sequential read from a random location (*Figure 16* and *Figure 17*):

- 497 • The Controller shall transmit:
- 498 • The HDR-DDR Command Word for LENGTH and INDEX
- 499 • The HDR-DDR Data Word, including LENGTH and INDEX
- 500 • The HDR-DDR CRC Word
- 501 • The HDR Restart Pattern
- 502 • The HDR-DDR Command Word for ReadData
- 503 • Then the Target shall send one or more HDR-DDR Read Data Words followed by the HDR-DDR
- 504 CRC Word
- 505 • Finally the Controller shall send either the HDR Restart Pattern or the HDR Exit Pattern.

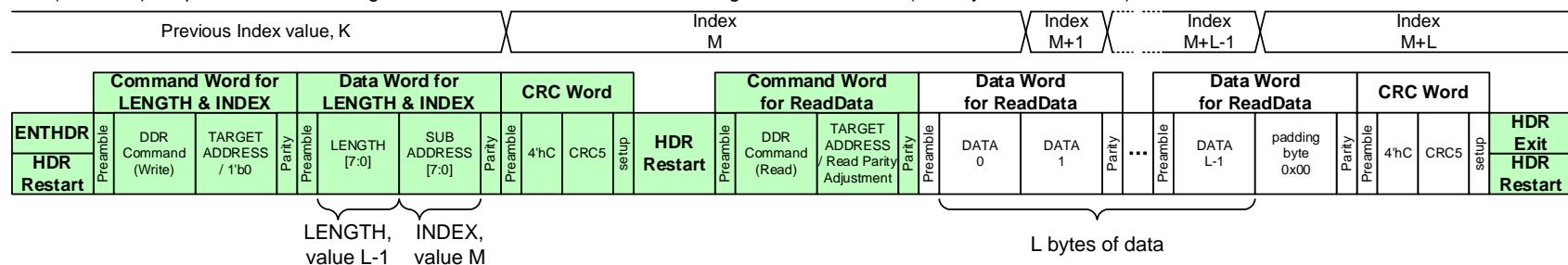
506 If the number of 8-bit data words read is odd (i.e. the value in the LENGTH field is even), then the Target
 507 shall insert one padding byte in the second byte of the last data word, with value 8'h00. The Target shall not
 508 increment INDEX by the padding byte. The Controller shall take into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

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

CCI (I3C DDR) Sequential Read Starting from a Random Location with 8-bit length and 8-bit index (even byte read transfer)



CCI (I3C DDR) Sequential Read Starting from a Random Location with 8-bit length and 8-bit index (odd byte read transfer)



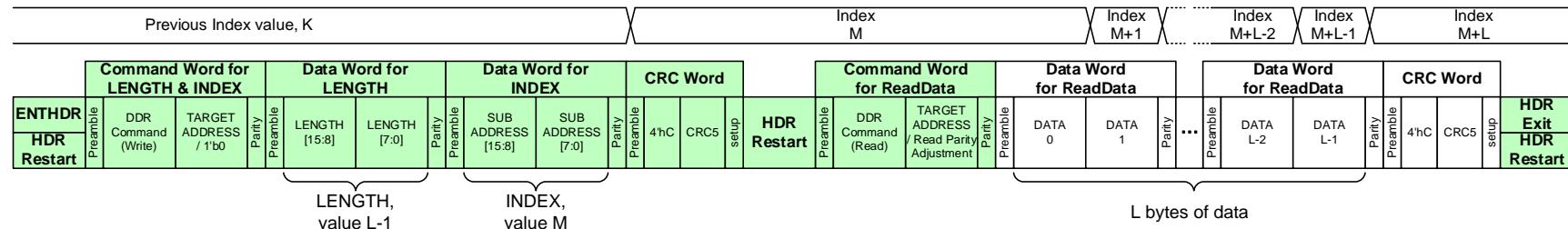
From Controller to Target

From Target to Controller

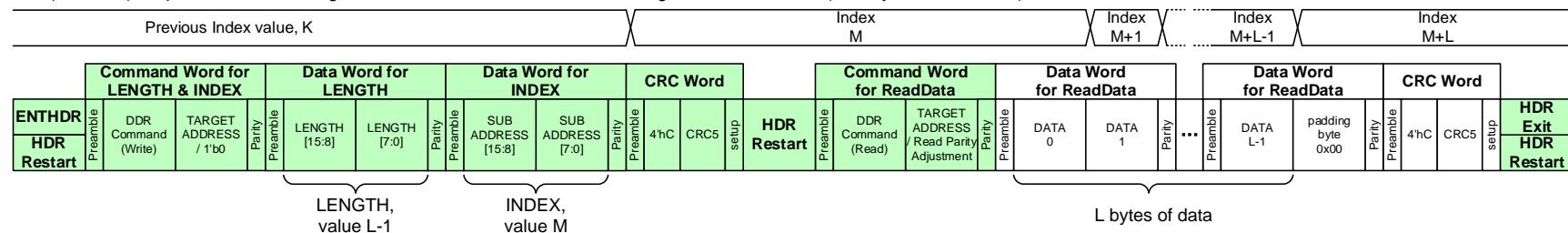
514

Figure 16 CCI (I3C DDR) Sequential Read from Random Location: 8-bit LENGTH & INDEX

CCI (I3C DDR) Sequential Read Starting from a Random Location with 16-bit length and 16-bit index (even byte read transfer)



CCI (I3C DDR) Sequential Read Starting from a Random Location with 16-bit length and 16-bit index (odd byte read transfer)



From Controller to Target

From Target to Controller

Figure 17 CCI (I3C DDR) Sequential Read from Random Location: 16-bit LENGTH & INDEX

6.2.2.2.3 CCI (I3C DDR) Concatenated Sequential Read from Random Location

When the Controller desires to read data longer than the Target's I3C Max Read Length (MRL) [**MIPI03**], the Controller can divide the data into multiple units, and efficiently read the data using the Concatenated Sequential Read from Random Location operation (**Figure 18** and **Figure 19**). The Controller shall divide the data into multiple units, where all units except the last unit shall use the MRL size, and the last unit shall use a size less than or equal to the MRL. The MRL size is programmable.

In a Concatenated Sequential Read Starting from Random Location:

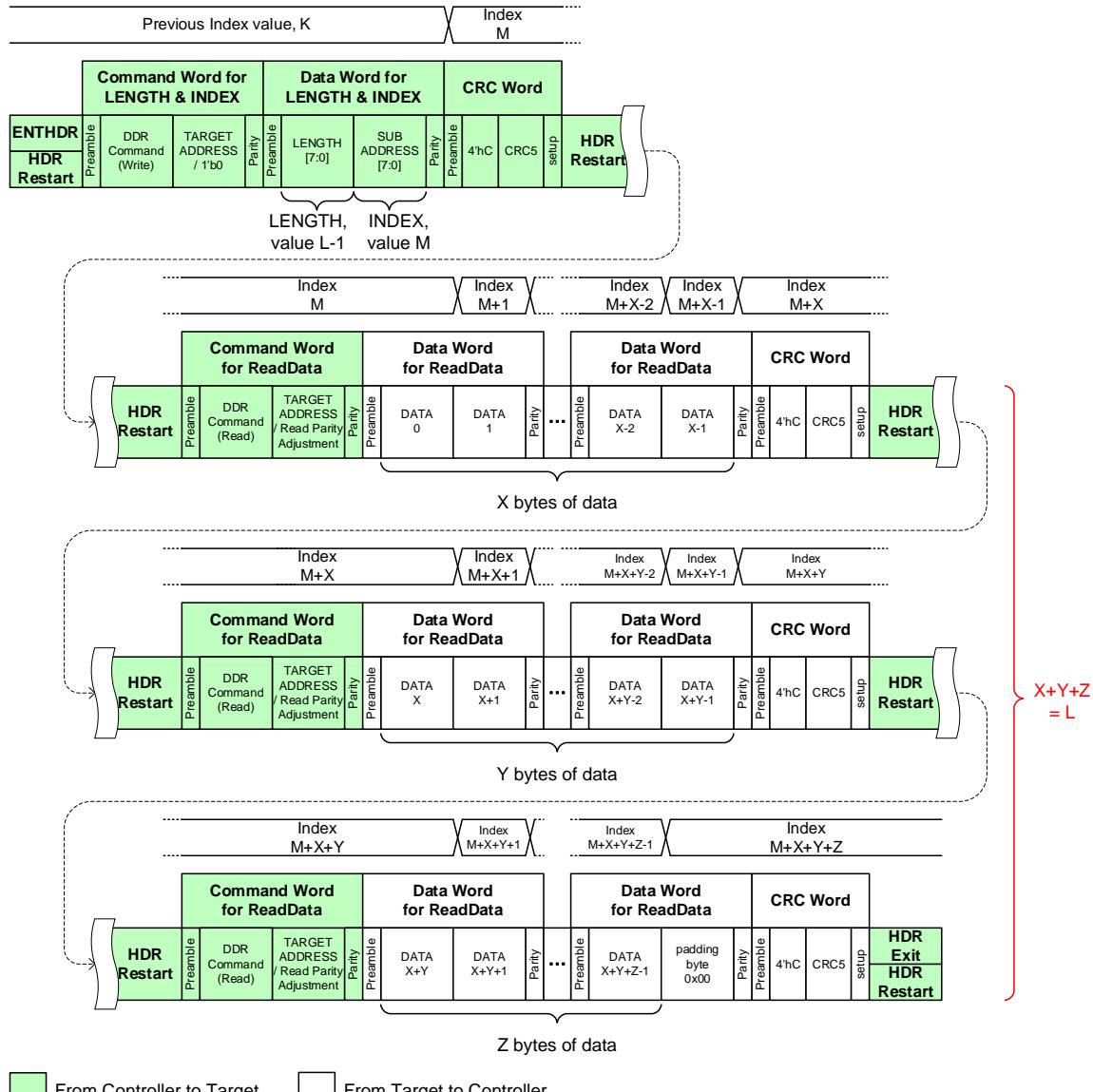
- The Controller shall first transmit the total LENGTH for the data to be read.
- The Controller shall use multiple read Messages. The Target shall transmit the initial read Messages to the Controller using the programmed MRL data bytes. And the Target may use no more than the programmed MRL data bytes to transfer the last Message.
- If the full amount of requested data has not been received yet, then the Controller shall transmit another read Message, but without LENGTH and INDEX.
- After receiving the read Message without LENGTH and INDEX, the Target shall continue transmission of the read data to the Controller, resuming from the previous LENGTH and INDEX.

The Controller shall continue to transmit read Messages without LENGTH and INDEX multiple times, until the last data is received.

Note:

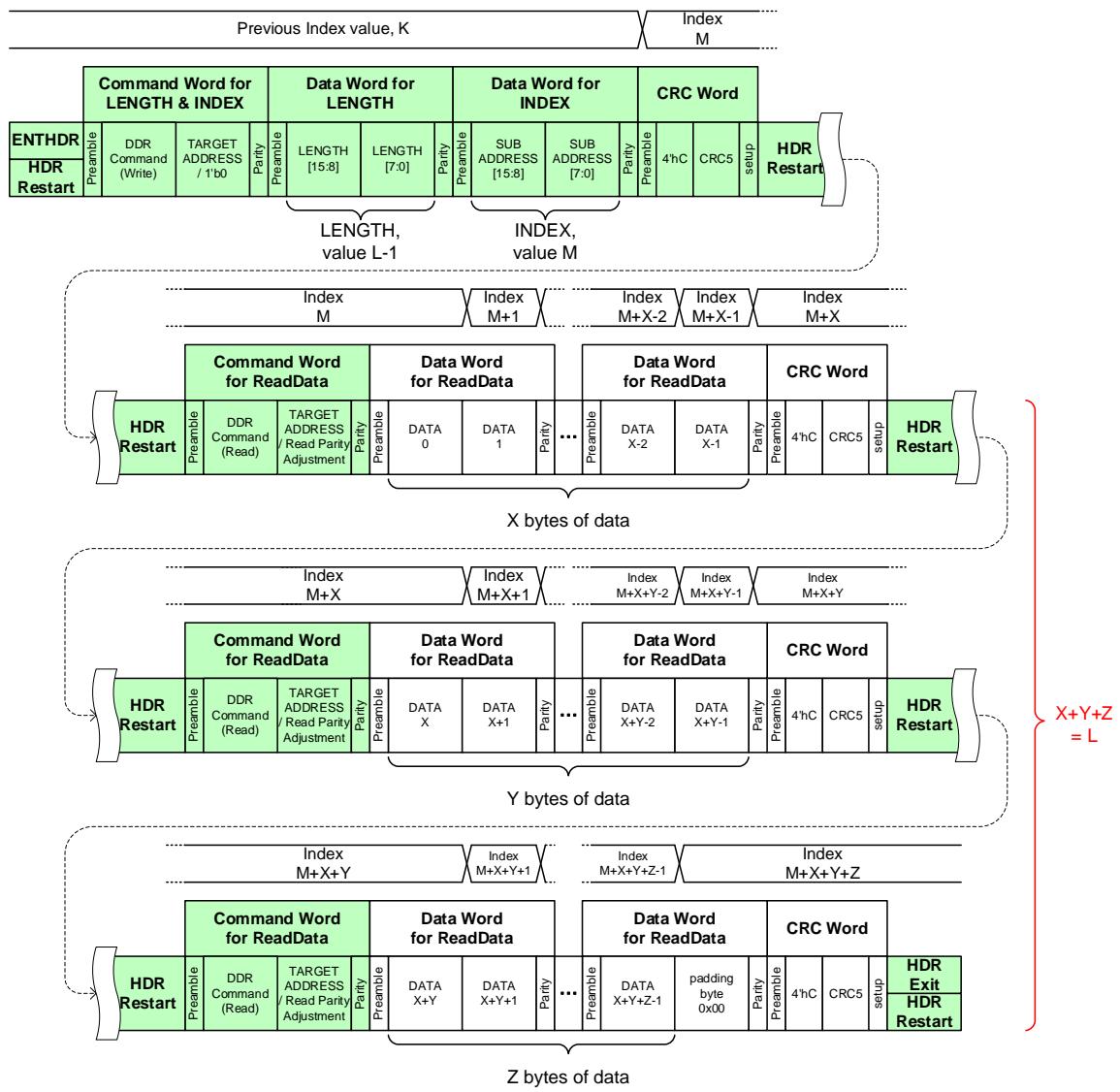
*When selecting a suitable value for MRL, the designer of the Target device and the system designer should take into account the needs of the payload that the CCI will carry. For example, in the CCS Data Transfer Interface [**MIPI04**], it is beneficial to support an MRL of 64 bytes or larger (i.e. 64 bytes for Data payload).*

CCI (I3C DDR) Concatenated Sequential Read Starting from a Random Location with 8-bit length and 8-bit index



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

CCI (I3C DDR) Concatenated Sequential Read Starting from a Random Location with 16-bit length and 16-bit index



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

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

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

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

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

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

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

Example

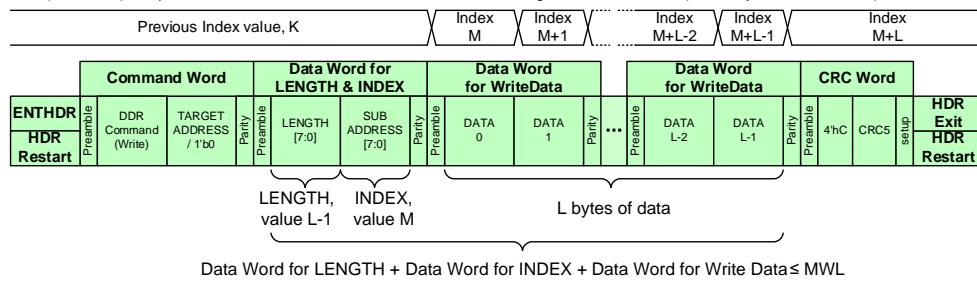
For a Target with MWL of 8 bytes, using 16-bit INDEX (so "Data Word for INDEX" is 2 bytes) and 16-bit LENGTH (so "Data Word for LENGTH" is 2 bytes), the maximum number of "Data Word for Write Data" is $8 - (2 + 2)$ bytes = 4 bytes. Since the LENGTH field is zero-based, it would contain the value 3 (16'd3).

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

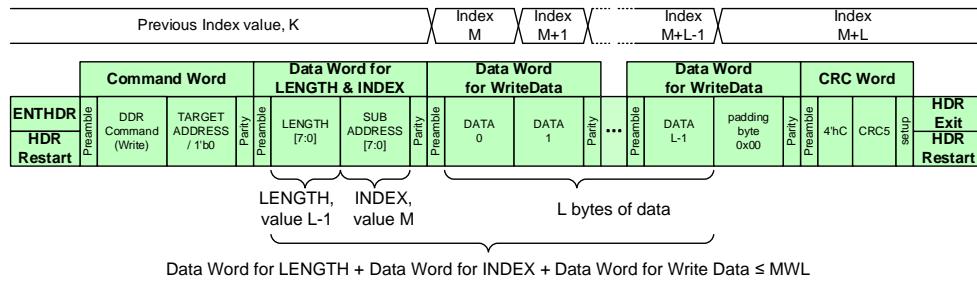
Note:

*When selecting a suitable value for MWL, the designer of the Target device and the system designer should take into account the needs of the payload that the CCI will carry. For example, in the CCS Data Transfer Interface [*MIPI04*], it is beneficial to support an MWL of 68 bytes or larger (i.e. 64 bytes for Data payload + 2 bytes for a Data Word for INDEX + 2 bytes for a Data Word for LENGTH).*

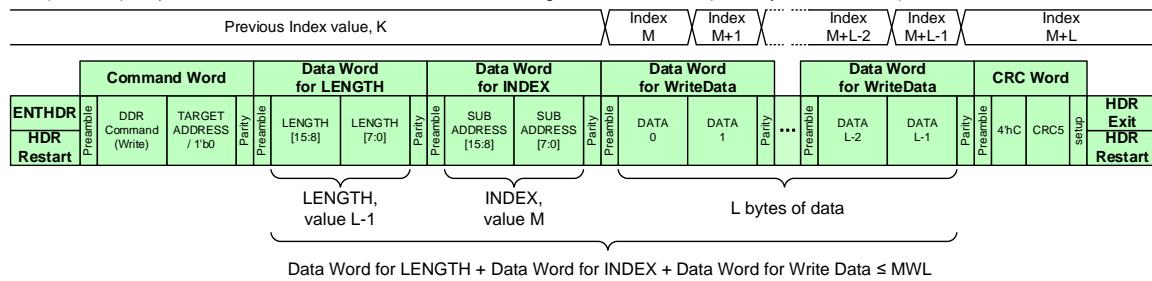
CCI (I3C DDR) Sequential Write to a Random Location with 8-bit length and 8-bit index (even byte write transfer)



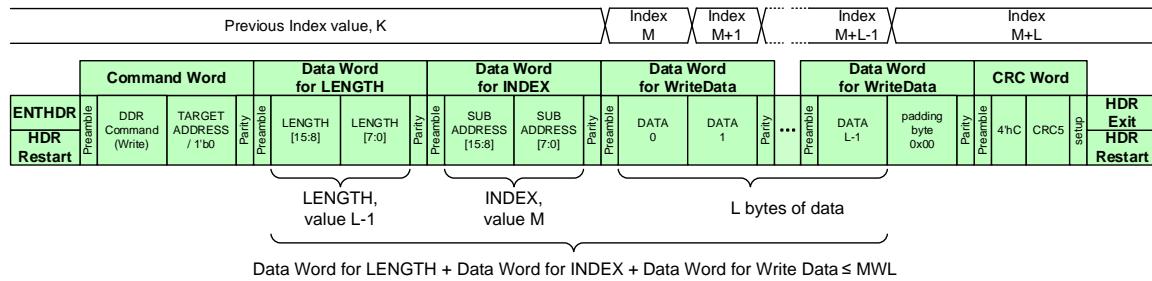
CCI (I3C DDR) Sequential Write to a Random Location with 8-bit length and 8-bit index (odd write byte transfer)



CCI (I3C DDR) Sequential Write to a Random Location with 16-bit length and 16-bit index (even byte write transfer)



CCI (I3C DDR) Sequential Write to a Random Location with 16-bit length and 16-bit index (odd write byte transfer)



From Controller to Target

From Target to Controller

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

6.3 CCI (I3C) Error Detection and Recovery

6.3.1 CCI (I3C SDR) Error Detection and Recovery Method

The error detection and recovery methods specified in this Section are provided in order to avoid fatal conditions when errors occur. The CCI (I3C SDR) error detection and recovery methods follow the I3C Specification. The I3C error detection and recovery method for the Target and Controller are specified in [MIPI03]. A CCI (I3C SDR) compatible device shall support both the methods defined by I3C and the methods defined in this Section regarding CCI (I3C SDR), respectively.

6.3.1.1 Error Detection and Recovery Method for CCI (I3C SDR) Target Devices

The SS0 error summarized in *Table 5* shall be supported for all CCI (I3C SDR) Target Devices. If the CCI Target detects the SS0 error, the CCI Target shall set to 1'b1 in Protocol Error Flag of GETSTATUS. Details of the SS0 error are described in *Section 6.3.1.1.2*.

Table 5 CCI (I3C SDR) Target Error Types

Error Type	Description	Error Detection Method	Error Recovery Method
SS0	Read without INDEX Error	Detect an error if the Target receives the Target's Dynamic Address (except 7'h7E) with a Read (R/W bit is 1) correctly but it does not have the INDEX.	Enable STOP or Repeated START detector and neglect other patterns.

6.3.1.1.1 Clearing the INDEX After Detecting I3C Error

The CCI (I3C SDR) Target shall clear the INDEX value when the I3C Target detects S2 [MIPI03] or S6 [MIPI03], optional during the “CCI (I3C SDR) Read/Write Operations” in *Table 2*. Note that this rule shall not be applicable to other Operations (e.g., I3C CCC Transfers). As defined in the I3C specification, the I3C Target sets to 1'b1 in the Protocol Error Flag of GETSTATUS (defined in the I3C specification) when the I3C Target detects an error.

Clearing the INDEX due to S2 and S6 errors in the CCI (I3C SDR) Write Operations (Single Write to Random Location, Sequential Write Starting from Random Location) is described below:

- If an S2 error occurs in the CCI (I3C SDR) Write Operations, the CCI Target cannot count up the INDEX because the CCI Target cannot receive the correct write data. As a result, the INDEX in the CCI Target may be different from the INDEX value that the Controller is expecting. In order to avoid this situation, the CCI Target shall clear the INDEX value.
- When the I3C Controller doesn't have the collision detector and the I3C Target has it, the INDEX in the CCI Target may be different from the INDEX value that the Controller is expecting in case of an S6 error. This is because the CCI Controller assumes the INDEX counter in the Target to be counting up, but the CCI Target stops the counter. In order to avoid this situation, the CCI Target shall clear the INDEX value.

Clearing the INDEX due to an S2 error in the CCI (I3C SDR) Read Operations (Single/Sequential Read to Random Location) is described below:

- If an S2 error occurs in the CCI (I3C SDR) Single/Sequential Read from Random Location during sub address, the CCI Target cannot update the value of INDEX because the I3C Target cannot get the correct sub address. This could cause Target to send undefined or wrong data. In order to avoid this situation, the CCI Target shall clear the INDEX value.

6.3.1.1.2 SS0 Error

The CCI Target shall detect an SS0 error if the CCI Target receives the Target address (except 7'h7E) with a Read (R/W bit is 1) correctly but it does not have the INDEX value. After detecting the SS0 error, the CCI Target shall replace ACK generated by the I3C Target with NACK during SS0 error and then wait for STOP or Repeated START. **Figure 21** illustrates how NACK is generated in CCI (I3C SDR) Sequential Read from Random Location, when SS0 error occurs during this Message.

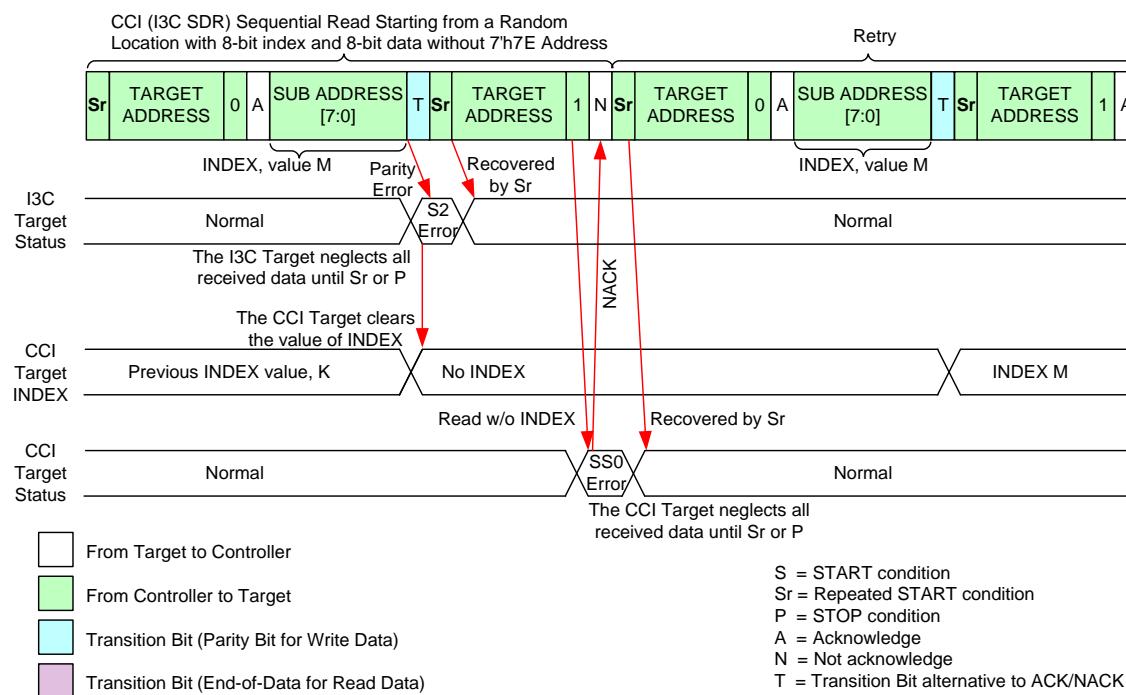


Figure 21 Example of SSO Error Detection

6.3.2 CCI (I3C DDR) Error Detection and Recovery Method

The error detection and recovery methods specified in this Section are provided in order to avoid fatal conditions when errors occur. The CCI (I3C DDR) error detection and recovery methods follow the I3C Specification. The I3C error detection and recovery method for the Target and Controller are specified in [MIPI03]. A CCI (I3C DDR) compatible device shall support both the methods defined by I3C and the methods defined in this section regarding CCI (I3C DDR) respectively.

6.3.2.1 Error Detection and Recovery Method for CCI (I3C DDR) Target Devices

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

Table 6 CCI (I3C DDR) Target Error Types

Error Type	Description	Error Detection Method	Error Recovery Method
SD0	Read without INDEX Error	Detect an error if the Target receives the DDR command Word[15] = 1 (Read) correctly, but it does not have the INDEX	Enable HDR Exit or HDR Restart detector and neglect other patterns
SD1	Write over LENGTH Error	Detect an error if the value of Preamble following LENGTH +1 bytes of the Write Data is 2'b11	Clear INDEX value. Enable HDR Exit or HDR Restart detector and neglect other patterns

6.3.2.1.1 Clearing INDEX After Detecting I3C Error

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

If a parity error occurs during the sub address in a CCI (I3C DDR) Read Operation (i.e. Sequential or Concatenated Sequential Read from Random Location), the CCI Target cannot update the value of INDEX because the I3C Target cannot get the correct sub address. This could cause Target to send undefined or wrong data. In order to avoid this situation, the CCI Target shall clear the INDEX value.

6.3.2.1.2 SD0 Error

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

629

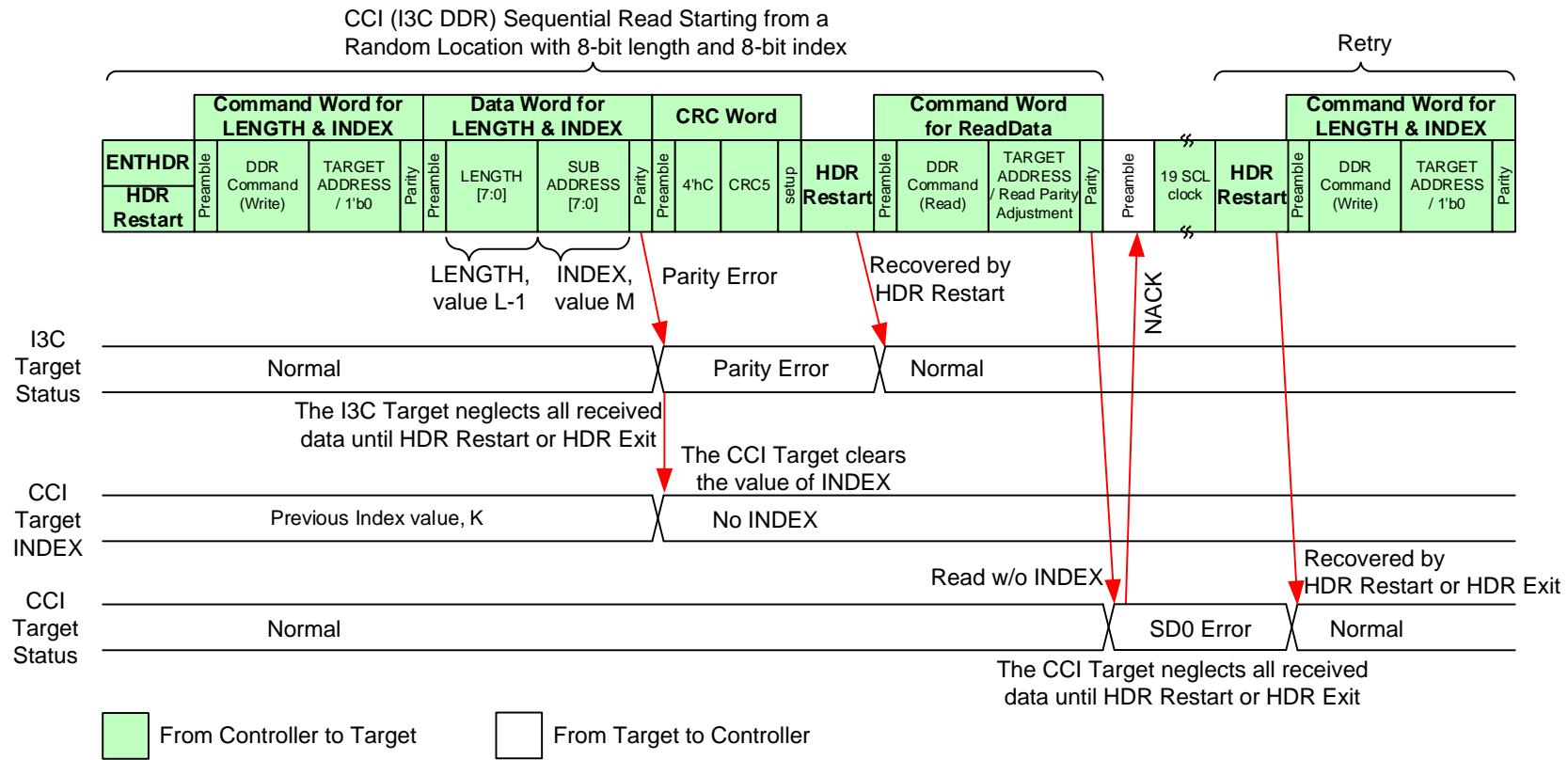


Figure 22 Example of SD0 Error Detection

6.3.2.1.3 SD1 Error

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

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

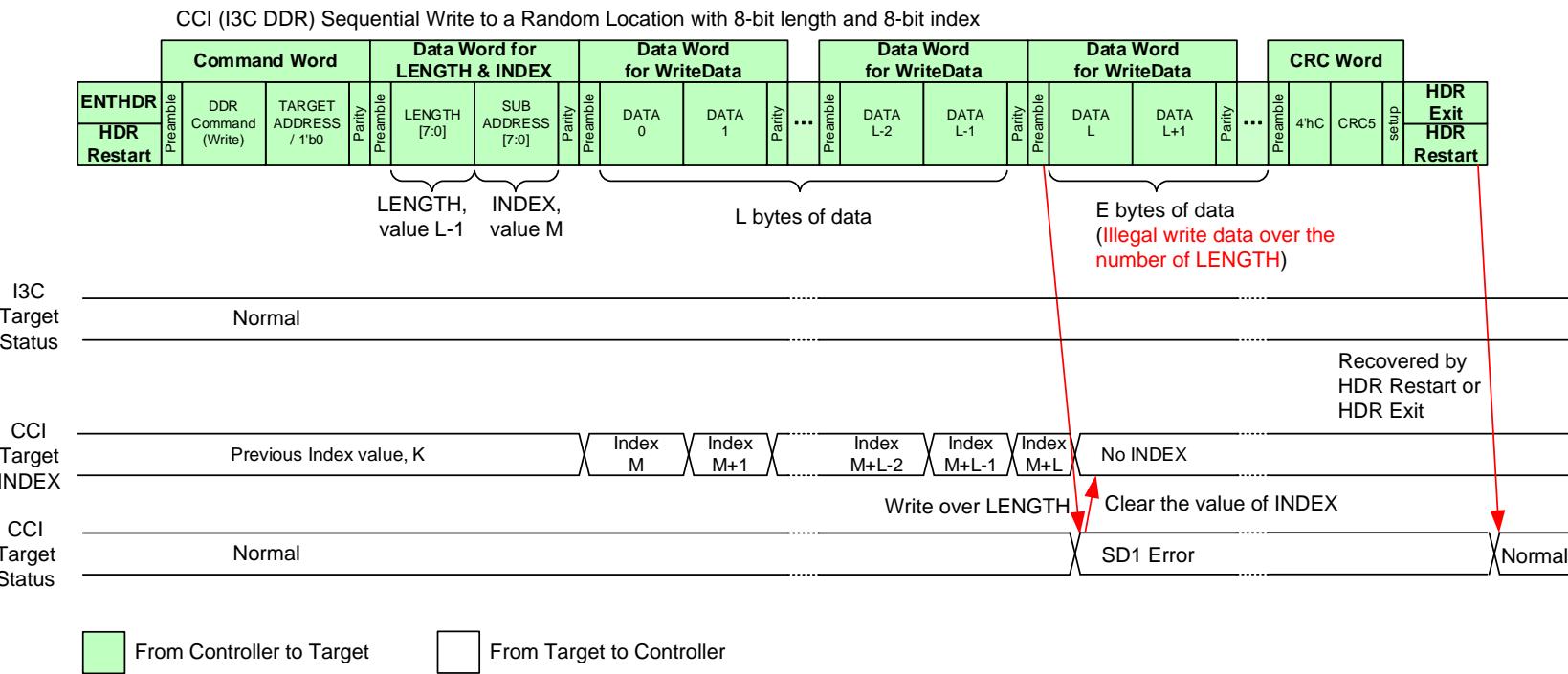


Figure 23 Example of SD1 Error Detection

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

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

Table 7 CCI (I3C DDR) Controller Error Type

Error Type	Description	Error Detection Method	Error Recovery Method
MD0 (optional)	Read over LENGTH Error	Detect an error if the value of Preamble[1] following LENGTH +1 bytes of the Read Data is 1'b1	Send Controller Abort and then HDR Exit or HDR Restart

6.3.2.2.1 MD0 Error

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

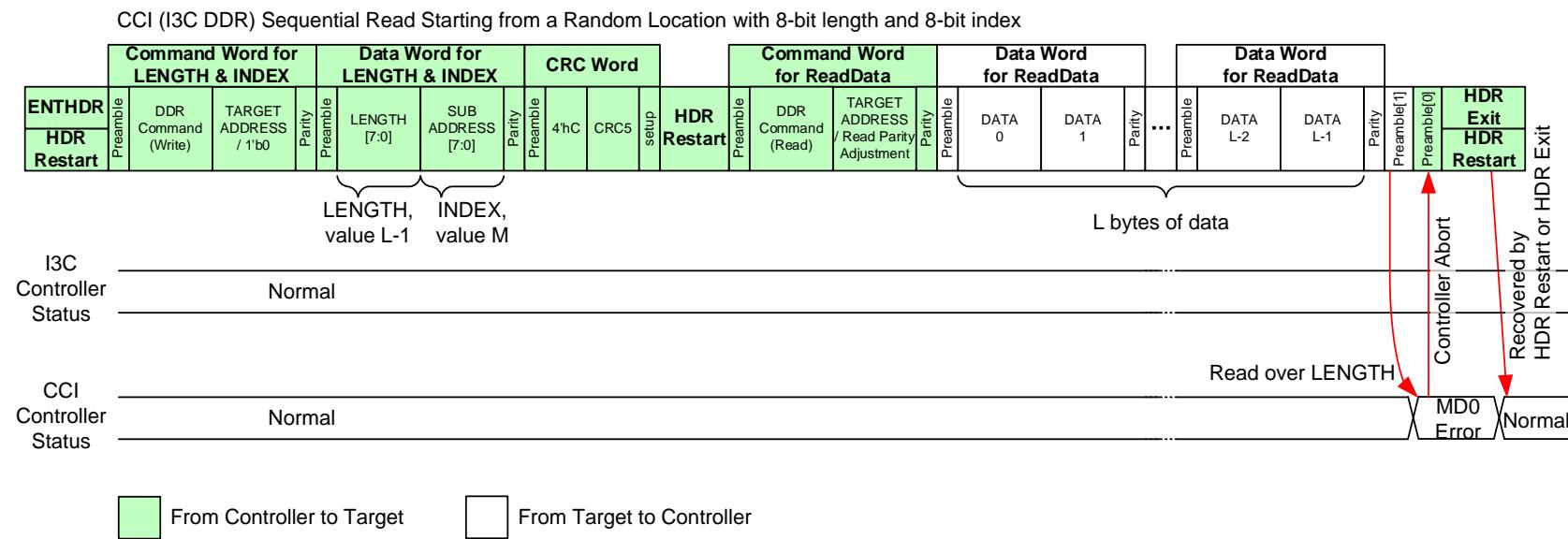


Figure 24 Example of MD0 Error Detection

6.3.3 Error Detection and Recovery for CCI (I3C) Controller Devices

In many cases, the Controller can detect an error inside the Target by receiving NACK. However, for example in case of an S2 or S6 error in the CCI (I3C SDR) Write Operations, the Controller cannot detect it by receiving NACK because there is no chance for the Target to send NACK by the end of the operation (STOP or Repeated START). Therefore if high reliability is required, the Controller may transmit GETSTATUS (defined in the I3C specification) at each important point.

Note:

E.g., the important point is that after critical CCI (I3C SDR) Write Operations, after CCI (I3C SDR) Write Operations before moving to CCI (I3C DDR), after multiple CCI (I3C SDR) Read/Write Operations before long pause if the last message is CCI (I3C SDR) Write Operations.

As a result, the Controller can detect each error by the following methods:

1. Target's error by receiving NACK
2. Target's error during CCI (I3C SDR) or CCI (I3C DDR) Write Operations by sending GETSTATUS
3. Controller's I3C SDR Error defined in the I3C specification (M0, M1 or M2 error)
4. Controller's I3C DDR Error defined in the I3C specification (including the Controller sending HDR Exit or HDR Restart pattern)

After detecting an error, the Controller should try the following error recovery method:

1. The Controller may retry sending the same CCI (I3C SDR) Read/Write Operations or CCI (I3C DDR) Read/Write Operations again.
2. The Controller may send certain other CCI (I3C SDR) Read/Write Operations or CCI (I3C DDR) Read/Write Operations, except CCI (I3C SDR) Single/Sequential Read From Current Location because the Target would generate NACK again due to an SS0 or SD0 error.

In addition to, or instead of, a retry, the Controller may read GETSTATUS, or try Escalation Handling as defined in the I3C specification.

6.4 CCI (I²C) Target Addresses

For camera modules having only raw Bayer output the 7-bit Target address should be 7'b011011X, where X = either 1'b0 or 1'b1. For all other camera modules the 7-bit Target address should be 7'b011110X.

6.5 CCI (I3C) Target Addresses

All camera modules shall use their own Dynamic Address as assigned by the I3C Controller.

6.6 CCI Multi-Byte Registers

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

6.6.1 Overview

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

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

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

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

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

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

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

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

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

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

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

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

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

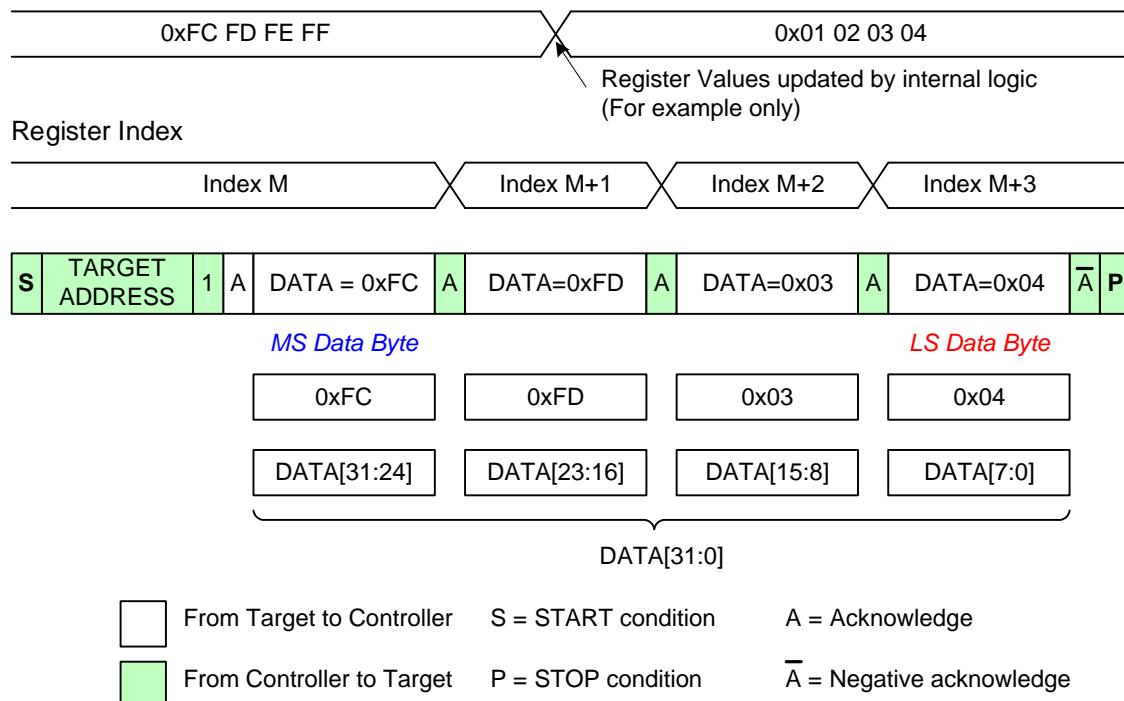


Figure 25 Corruption of 32-bit Register During Read Message

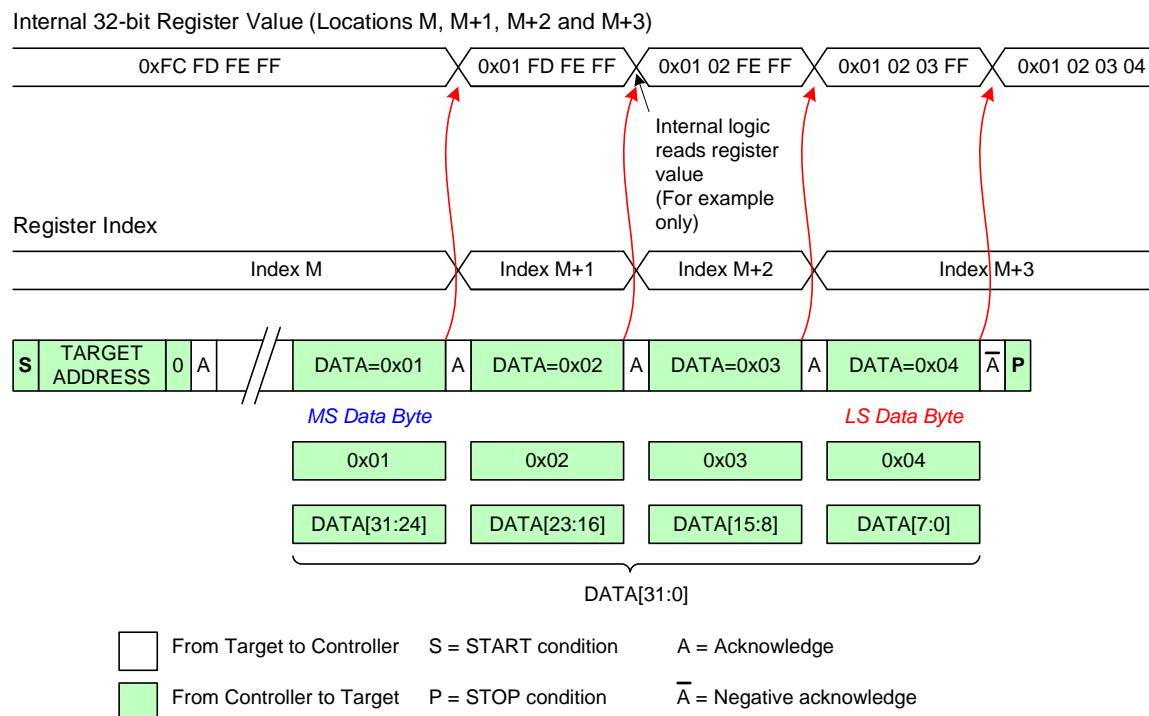
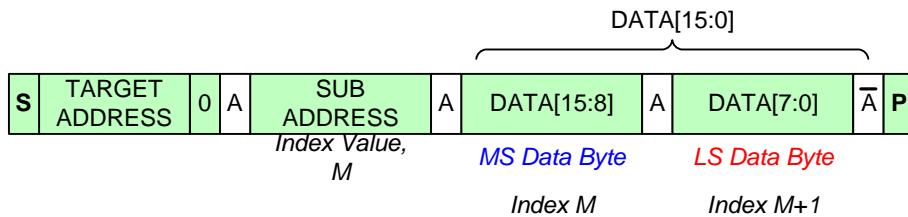


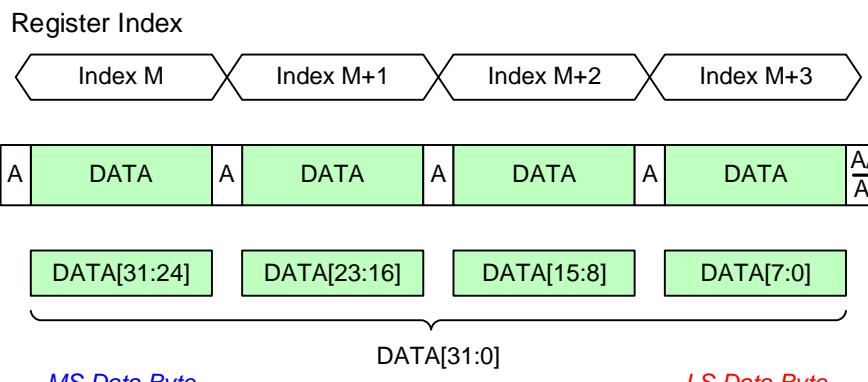
Figure 26 Corruption of 32-bit Register During Write Message

6.6.2 Transmission Byte Order for Multi-Byte Register Values

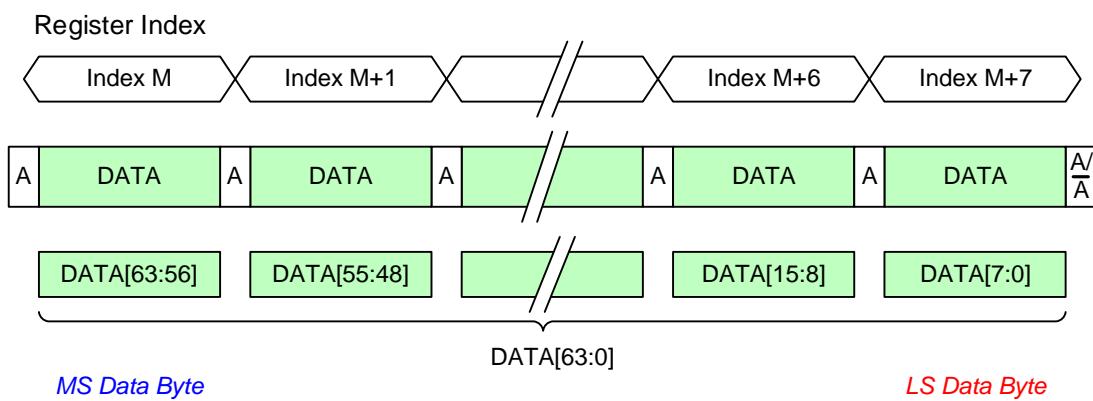
714 **Figure 27, Figure 28, and Figure 29** illustrate the requirement that the first byte of a CCI Message shall
 715 always be the MS byte of a multi-byte register, and the last byte of the CCI Message shall always be the LS
 716 byte of the multi-byte register.



717

Figure 27 Example 16-bit Register Write

718

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

719

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

6.6.3 Multi-Byte Register Protocol (Informative)

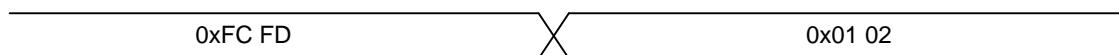
Each device may have both single-byte registers and multi-byte registers. Internally a device must understand what addresses correspond to the different register widths.

6.6.3.1 Reading Multi-Byte Registers

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

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

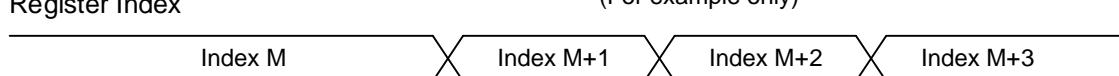
Internal 16-bit Register Value (Locations M and M+1)



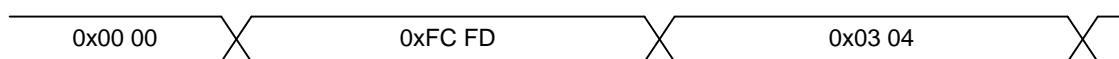
Internal 16-bit Register Value (Locations M+2 and M+3)



Register Values updated by internal logic
(For example only)

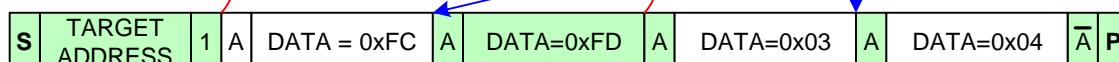


Temporary Buffer



A read from MS byte of the register causes the whole register value to be transferred into a temporary buffer

Incremental read within the same multi-byte register.
Temporary Buffer not updated



MS Data Byte

LS Data Byte

MS Data Byte

LS Data Byte

0xFC

0xFD

0x03

0x04

DATA[15:8]

DATA[7:0]

DATA[15:8]

DATA[7:0]

DATA[15:0]

DATA[15:0]



From Target to Controller

S = START condition

A = Acknowledge



From Controller to Target

P = STOP condition

\bar{A} = Negative acknowledge

Figure 30 Example 16-bit Register Read

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

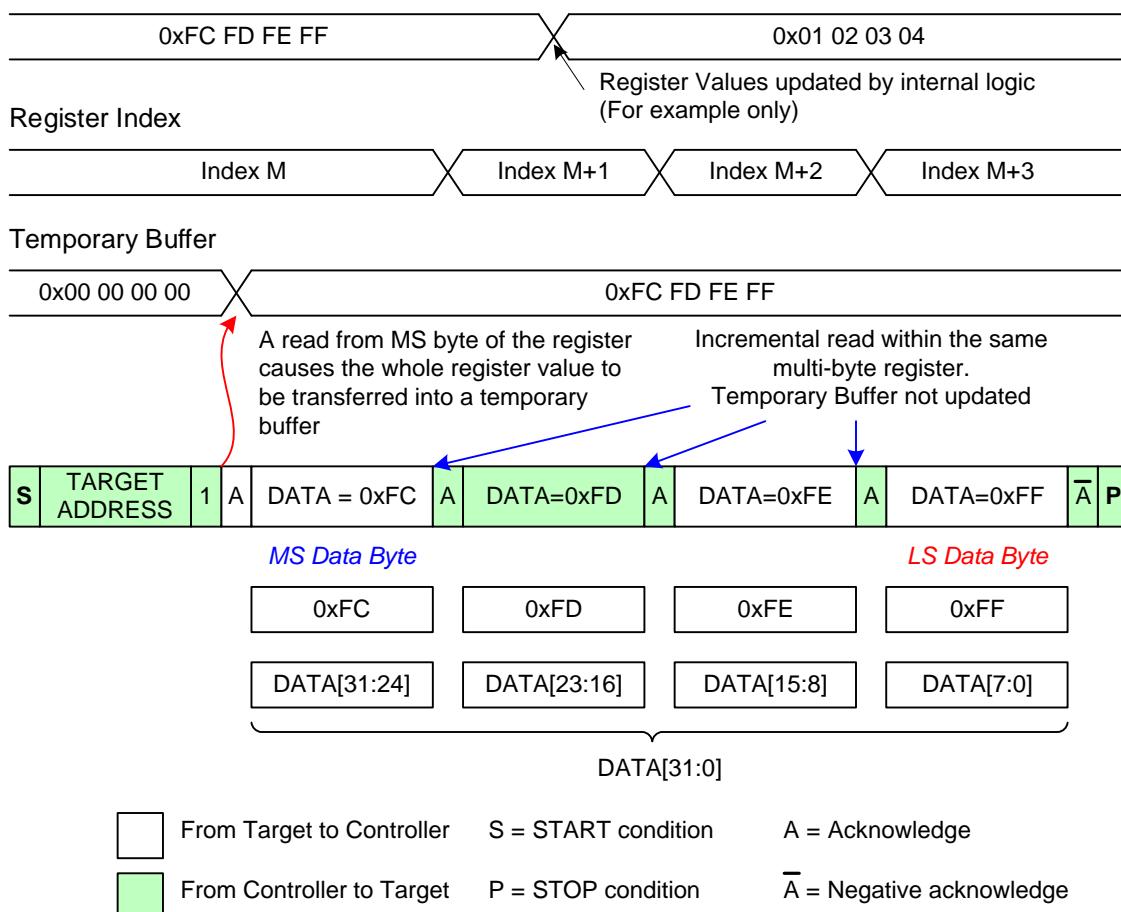
732 Examples of when the temporary buffer is updated include:

- 733 • When the MS byte of a register is accessed
 734 • When the INDEX has crossed a multi-byte register boundary
 735 • Successive single-byte reads from the same INDEX location
 736 • When the INDEX value for the byte about to be read is \leq the previous INDEX

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

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

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



740 **Figure 31 Example 32-bit Register Read**

6.6.3.2 Writing Multi-Byte Registers

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

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

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

Internal 16-bit Register Value (Locations M and M+1)

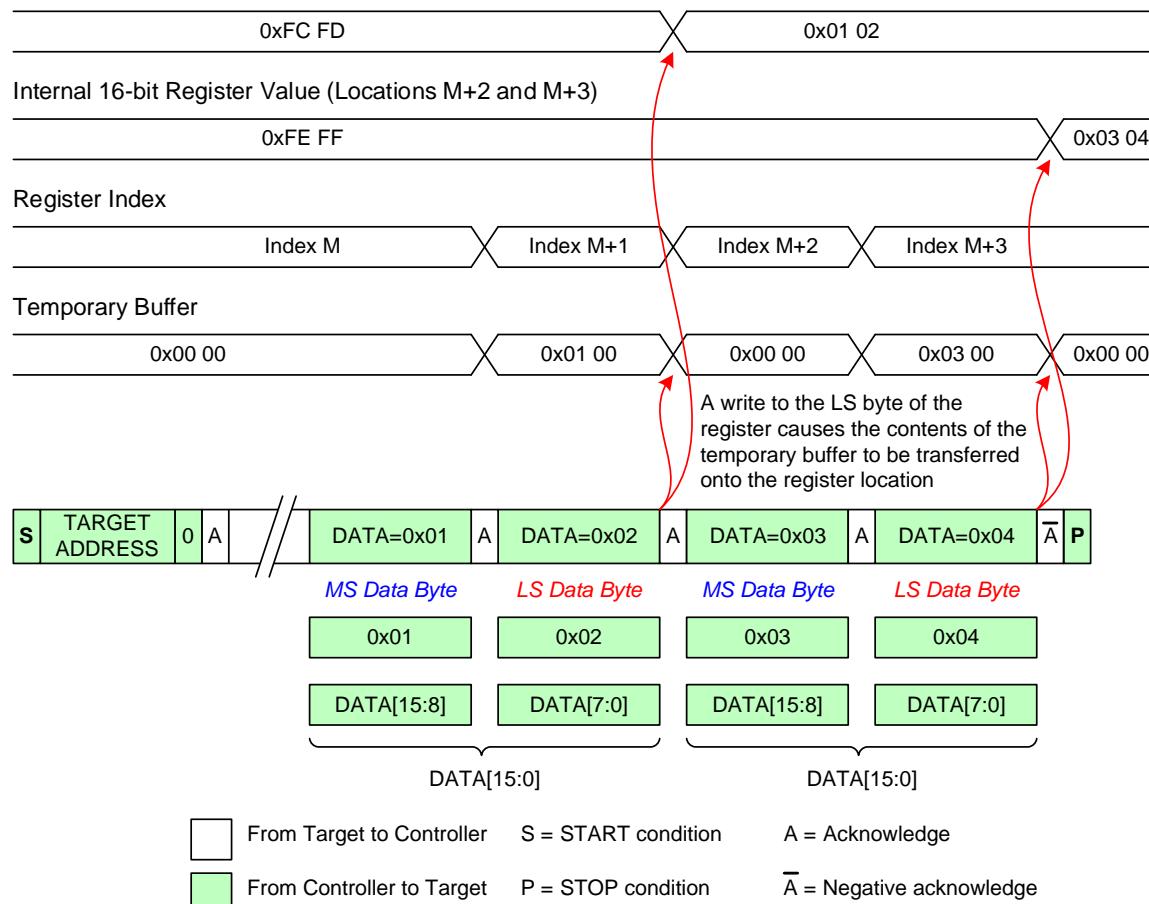
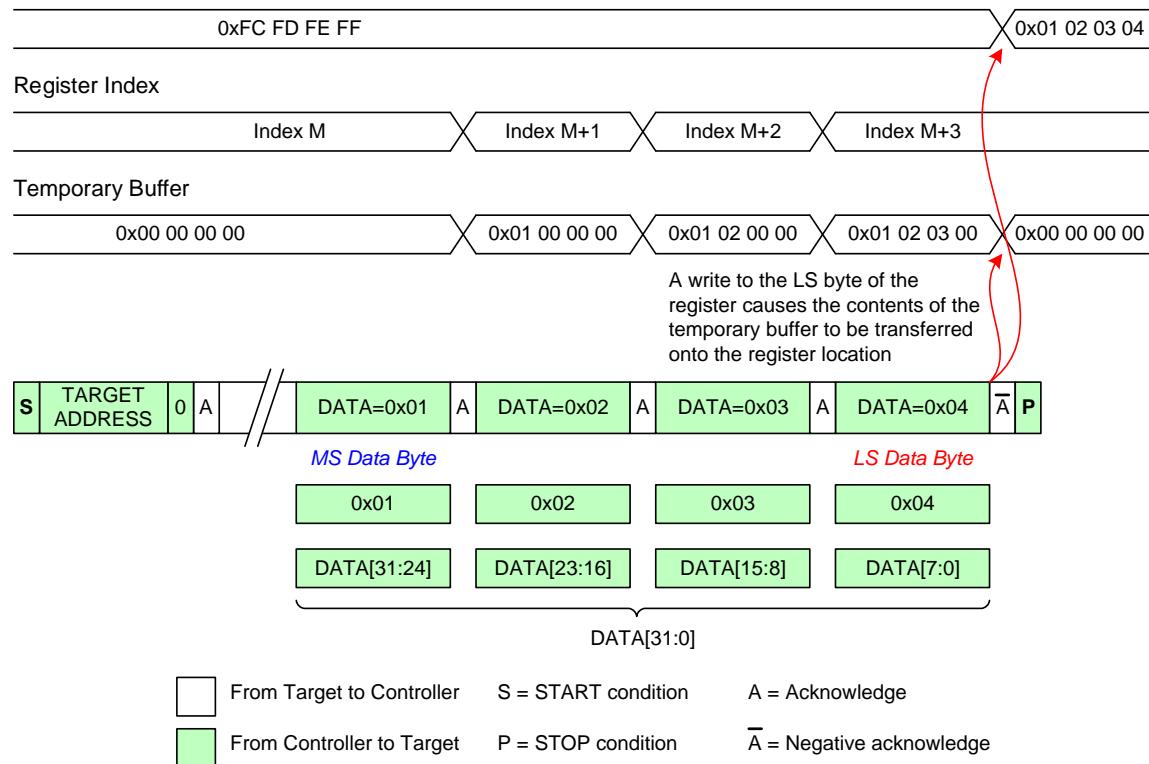


Figure 32 Example 16-bit Register Write

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



748

Figure 33 Example 32-bit Register Write

6.7 CCI I/O Electrical and Timing Specifications

The CCI I/O stages electrical specifications (*Table 8*) and timing specifications (*Table 9*) conform to I²C Fast-mode and Fast-mode Plus devices. Information presented in *Table 8* is from *[NXP01]*.

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

Table 8 CCI I/O Electrical Specifications

Parameter	Symbol	Fast-mode		Fast-mode Plus		Unit
		Min.	Max.	Min.	Max.	
LOW level input voltage	V _I L	-0.5	0.3 V _{DD}	-0.5	0.3 V _{DD}	V
HIGH level input voltage	V _I H	0.7V _{DD}	Note 1	0.7V _{DD}	Note 1	V
Hysteresis of Schmitt trigger inputs	V _{HYS}	0.05V _{DD}	—	0.05V _{DD}	—	V
LOW level output voltage (open drain) at 2mA sink current V _{DD} > 2V V _{DD} < 2V	V _{OL} 1 V _{OL} 3	0 0	0.4 0.2V _{DD}	0 0	0.4 0.2V _{DD}	V
Output fall time from V _I Hmin to V _I Lmax with bus capacitance from 10 pF to 400 pF	t _{OF}	20 x(V _{DD} / 5.5 V)	250	20 x (V _{DD} / 5.5 V)	120	ns
Pulse width of spikes which shall be suppressed by the input filter	t _{SP}	0	50	0	50	ns
Input current each I/O pin with an input voltage between 0.1 V _{DD} and 0.9 V _{DD}	I _I	-10 Note 2	10 Note 2	-10 Note 2	10 Note 2	µA
Input/Output capacitance (SDA)	C _{I/O}	—	10	—	10	pF
Input capacitance (SCL)	C _I	—	10	—	10	pF

Note:

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

Table 9 CCI I/O Timing Specifications

Parameter	Symbol	Fast-mode		Fast-mode Plus		Unit
		Min.	Max.	Min.	Max.	
SCL clock frequency	f_{SCL}	0	400	0	1000	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	0.6	—	0.26	—	μs
LOW period of the SCL clock	t_{LOW}	1.3	—	0.5	—	μs
HIGH period of the SCL clock	t_{HIGH}	0.6	—	0.26	—	μs
Setup time for a repeated START condition	$t_{SU;STA}$	0.6	—	0.26	—	μs
Data hold time	$t_{HD;DAT}$	0 Note 2	— Note 3	0	—	μs
Data set-up time	$t_{SU;DAT}$	100 Note 4	—	50	—	ns
Rise time of both SDA and SCL signals	t_R	20	300	—	120	ns
Fall time of both SDA and SCL signals	t_F	$20 \times (V_{DD} / 5.5 \text{ V})$	300	$20 \times (V_{DD} / 5.5 \text{ V})$	120	ns
Set-up time for STOP condition	$t_{SU;STOP}$	0.6	—	0.26	—	μs
Bus free time between a STOP and START condition	t_{BUF}	1.3	—	0.5	—	μs
Capacitive load for each bus line	C_B	—	400	—	550	pF
Data valid time Note 5	$t_{VD;DAT}$	—	0.9 Note 3	—	0.45 Note 3	μs
Data valid acknowledge time Note 6	$t_{VD;ACK}$	—	0.9 Note 3	—	0.45 Note 3	μs
Noise margin at the LOW level for each connected device (including hysteresis)	V_{nL}	$0.1 \times V_{DD}$	—	$0.1 \times V_{DD}$	—	V
Noise margin at the HIGH level for each connected device (including hysteresis)	V_{nH}	$0.2 \times V_{DD}$	—	$0.2 \times V_{DD}$	—	V

Note:

1. All values referred to $V_{IH\min} = 0.7V_{DD}$ and $V_{IL\max} = 0.3V_{DD}$
2. A device shall internally provide a hold time of at least 300 ns for the SDA signal (referred to the $V_{IH\min}$ of the SCL signal) to bridge the undefined region of the falling edge of SCL
3. The maximum $t_{HD;DAT}$ could be 0.9 μs and 0.45 μs for Fast-mode and Fast-mode Plus, but must be less than the maximum of $t_{VD;DAT}$ or $t_{VD;ACK}$ by a transition time. This maximum must only be met if the device does not stretch the LOW period (t_{LOW}) of the SCL signal. If the clock stretches the SCL, then the data must be valid by the set-up time before it releases the clock.
4. A Fast-mode I²C-bus device can be used in a Standard-mode I²C-bus system, but the requirement $t_{SU;DAT} \geq 250$ ns shall be then met. This will be automatically the case if the device does not stretch the LOW period of the SCL signal. If such device does stretch the low period of SCL signal, it shall output the next data bit to the SDA line $t_{MAX} + t_{SU;DAT} = 1000 + 250 = 1250$ ns (according to the Standard-mode I²C Bus specification [**NXP01**]) before the SCL line is released.
5. $t_{VD;DAT}$ = time for data signal from SCL LOW to SDA output (HIGH or LOW, whichever is worse).
6. $t_{VD;ACK}$ = time for Acknowledgement signal from SCL LOW to SDA output (HIGH or LOW, whichever is worse)

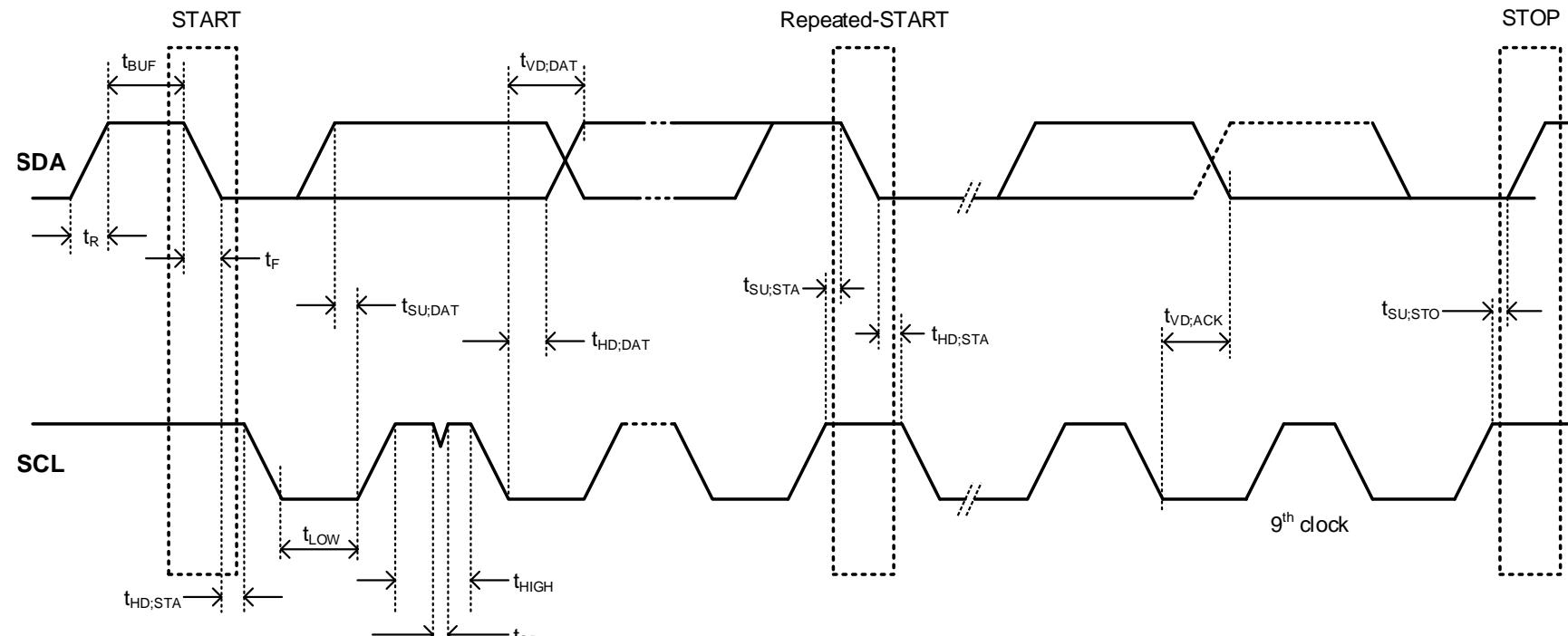


Figure 34 CCI I/O Timing

754

7 Physical Layer

The CSI-2 specification shall always be used in combination with one or more MIPI physical layer specifications, such as A-PHY [[MIPI06](#)], C-PHY [[MIPI02](#)], or D-PHY [[MIPI01](#)]. The CSI-2 specification shall not be used in combination with non-MIPI physical layers, unless expressly authorized by the MIPI Alliance Board of Directors. Any other use of the CSI-2 specification is strictly prohibited unless approved in advance by the MIPI Alliance Board of Directors.

Note:

In this specification version, D-PHY “CIL-Mxxx” Lane type descriptors have been replaced with “CIL-Pxxx” in anticipation of them being similarly updated in a forthcoming D-PHY specification.

7.1 D-PHY Physical Layer Option

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

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

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

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

- Data Lane Module: Unidirectional Primary, HS-TX, LP-TX and a CIL-PFEN function
- Clock Lane Module: Unidirectional Primary, HS-TX, LP-TX and a CIL-PCNN function

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

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

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

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

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

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

For any transmitter frequency changes the deskew calibration shall be re-run.

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

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

The D-PHY v2.5 physical layer [[MIPI01](#)] provides both Alternate Low Power (ALP) Mode and Low Voltage Low Power (LVLP) signaling, either of which may optionally replace the legacy Low Power State (LPS). Use of ALP Mode or LVLP signaling can help alleviate current leakage and electrical overstress issues with image sensors and applications processors. ALP Mode can also help achieve longer reach for CSI-2 imaging

796 interface channels, and is also central to the CSI-2 Unified Serial Link (USL) feature described in
797 **Section 9.12**. USL D-PHY support requirements are described in **Section 7.3.1**.

7.2 C-PHY Physical Layer Option

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

800 The minimum C-PHY physical layer requirement for a CSI-2 transmitter Lane module is:

- Unidirectional Primary, HS-TX, LP-TX and a CIL-PFEN function
- Support for Sync Word insertion during data payload transmission

803 The minimum C-PHY physical layer requirement for a CSI-2 receiver Lane module is:

- Unidirectional Secondary, HS-RX, LP-RX, and a CIL-SFEN function
- Support for Sync Word detection during data payload reception

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

808 The C-PHY Physical Layer provides both Alternate Low Power (ALP) Mode and Low Voltage Low Power
809 (LVLP) signaling, either of which may optionally replace the legacy Low Power State (LPS). Use of ALP
810 Mode or LVLP signaling can help alleviate current leakage and electrical overstress issues with image sensors
811 and applications processors. ALP Mode (which replaces LVLP or legacy LP signaling through the use of
812 high-speed embedded codes) can also help achieve longer reach for CSI-2 imaging interface channels before
813 re-drivers and re-timers become necessary. ALP Mode is also central to the CSI-2 Unified Serial Link (USL)
814 feature described in **Section 9.12**. USL C-PHY support requirements are described in **Section 7.3.2**.

7.3 PHY Support for the CSI-2 Unified Serial Link (USL) Feature

The CSI-2 USL feature, as described in [Section 9.12](#), requires the D-PHY and C-PHY physical layers to support bidirectional data communications on Lane 1 plus additional features as described below in [Section 7.3.1](#) and [Section 7.3.2](#), respectively. In case of any conflict between this [Section 7.3](#) and [Section 7.1](#) or [Section 7.2](#), this section takes precedence. The physical layers of all USL implementations shall support PHY LP and/or LVLP Mode signaling and should support ALP Mode signaling.

7.3.1 D-PHY Support Requirements for USL Feature

The D-PHY physical layer for a CSI-2 USL implementation is composed of one bidirectional data Lane (i.e., data Lane 1), zero or more unidirectional data Lanes, and one clock Lane. All CSI-2 transmitters and receivers implementing the D-PHY physical layer for the USL feature shall support continuous clock behavior on the clock Lane, and optionally may support non-continuous clock behavior.

For continuous clock behavior, the clock Lane remains in high-speed mode, generating active clock signals between data packet transmissions.

For non-continuous clock behavior, the clock Lane may enter the Stop state between data packet transmissions as determined by the image sensor and controlled by the host processor.

The minimum D-PHY LP/LVLP Mode physical layer requirement for a USL image sensor is:

- **Clock Lane Module:** Unidirectional Primary, HS-TX, LP-TX, and CIL-PCNN function
- **Data Lane 1 Module:** Bidirectional Primary, HS-TX, LP-TX, LP-RX, LP-CD, and CIL-PFAA function; Escape Mode LPDT shall be supported in both the forward and reverse direction
- **Data Lane n Module (for n > 1):** Unidirectional Primary, HS-TX, LP-TX, and CIL-PFEN function

The minimum D-PHY LP/LVLP Mode physical layer requirement for a USL host is:

- **Clock Lane Module:** Unidirectional Secondary, HS-RX, LP-RX, and CIL-SCNN function
- **Data Lane 1 Module:** Bidirectional Secondary, HS-RX, LP-TX, LP-RX, LP-CD, and CIL-SFAA function; Escape Mode LPDT shall be supported in both the forward and reverse direction
- **Data Lane n Module (for n > 1):** Unidirectional Secondary, HS-RX, LP-RX, and CIL-SFEN function

For USL implemented using D-PHY LP/LVLP Mode, forward direction Escape Mode LPDT transmissions shall use data Lane 1 only, and all reverse direction transmissions shall only use data Lane 1 and LPDT. The USL host shall be capable of receiving both LPDT and High-Speed (HS) transmissions. Note that transmission bandwidth is substantially reduced when transmitting using LPDT.

The minimum D-PHY ALP Mode physical layer requirement for a USL image sensor is:

- **Clock Lane Module:** Unidirectional Primary, HS-TX and CIL-PCNN function
- **Data Lane 1 Module:** Bidirectional Primary, HS-TX, HS-RX, ALP-ED, and CIL-PREN function (CIL-PREE with ALP-ULPS in both forward and reverse direction is recommended)
- **Data Lane n Module (for n > 1):** Unidirectional Primary, HS-TX and CIL-PFEN function

The minimum D-PHY ALP Mode physical layer requirement for a USL host is:

- **Clock Lane Module:** Unidirectional Secondary, HS-RX, ALP-ED, and CIL-SCNN function
- **Data Lane 1 Module:** Bidirectional Secondary, HS-TX, HS-RX, ALP-ED, and CIL-SREN function (CIL-SREE with ALP-ULPS in both forward and reverse direction is recommended)
- **Data Lane n Module (for n > 1):** Unidirectional Secondary, HS-RX, ALP-ED, and CIL-SFEN function

Note that D-PHY ALP Mode does not define a contention detection function for bidirectional Lane modules.

All USL implementations supporting the D-PHY physical layer option shall support forward direction ULPS on all data Lanes. For data Lane 1, support for both reverse direction ULPS and the reverse direction ALP-wake pulse transmission is recommended; see [Section 9.12.5.6](#) and [Section 9.12.5.8](#) for additional guidance.

7.3.2 C-PHY Support Requirements for USL Feature

The C-PHY physical layer for a CSI-2 USL implementation is composed of one bidirectional Lane (i.e., Lane 1) and zero or more unidirectional Lanes.

The minimum C-PHY LP/LVLP Mode physical layer requirement for a USL image sensor is:

- **Lane 1 Module:** Bidirectional Primary, HS-TX, LP-TX, LP-RX, LP-CD, and CIL-PFAA function;
Escape Mode LPDT shall be supported in both the forward and reverse direction
- **Lane n Module (for n > 1):** Unidirectional Primary, HS-TX, LP-TX, and CIL-PFEN function

The minimum C-PHY LP/LVLP Mode physical layer requirement for a USL host is:

- **Lane 1 Module:** Bidirectional Secondary, HS-RX, LP-TX, LP-RX, LP-CD, and CIL-SFAA
function; Escape Mode LPDT shall be supported in both the forward and reverse direction
- **Lane n Module (for n > 1):** Unidirectional Secondary, HS-RX, LP-RX, and CIL-SFEN function

For USL implemented using C-PHY LP/LVLP Mode, forward direction Escape Mode LPDT transmissions shall use Lane 1 only, and all reverse direction transmissions shall only use Lane 1 and LPDT. The USL host shall be capable of receiving both LPDT and High-Speed (HS) transmissions. Note that transmission bandwidth is substantially reduced when transmitting using LPDT.

The minimum C-PHY ALP Mode physical layer requirement for a USL image sensor is:

- **Lane 1 Module:** Bidirectional Primary, HS-TX, HS-RX, and CIL-PREN function (CIL-PREE
with ALP-ULPS in both forward and reverse direction is recommended)
- **Lane n Module (for n > 1):** Unidirectional Primary, HS-TX and CIL-PFEN function

The minimum C-PHY ALP Mode physical layer requirement for a USL host is:

- **Lane 1 Module:** Bidirectional Secondary, HS-TX, HS-RX, and CIL-SREN function (CIL-SREE
with ALP-ULPS in both forward and reverse direction is recommended)
- **Lane n Module (for n > 1):** Unidirectional Secondary, HS-RX and CIL-SFEN function

Note that C-PHY ALP Mode does not define a contention detection function for bidirectional Lane modules.

All CSI-2 USL implementations supporting the C-PHY physical layer option shall support forward direction ULPS on all Lanes. For Lane 1, additional C-PHY ALP Mode support for reverse direction ULPS is recommended; see **Section 9.12.5.8** for additional guidance.

8 Multi-Lane Distribution and Merging

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

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

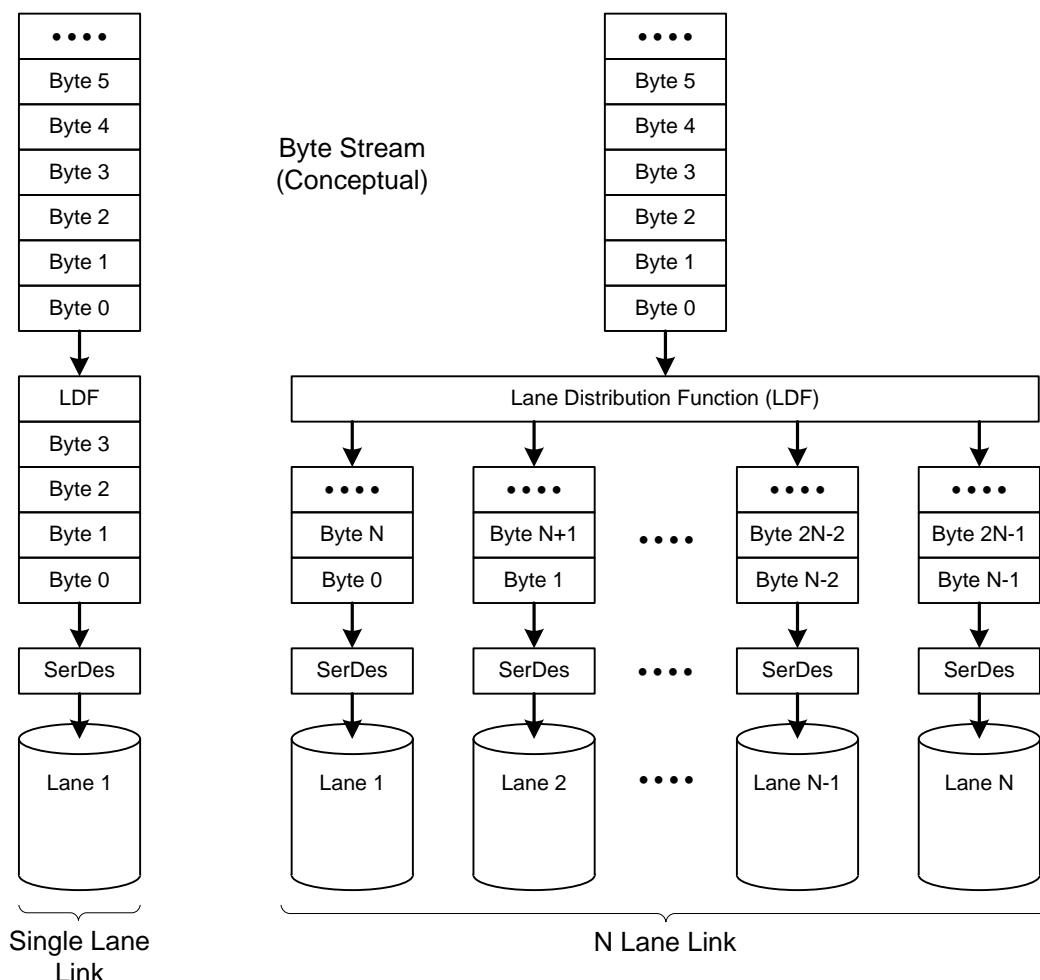
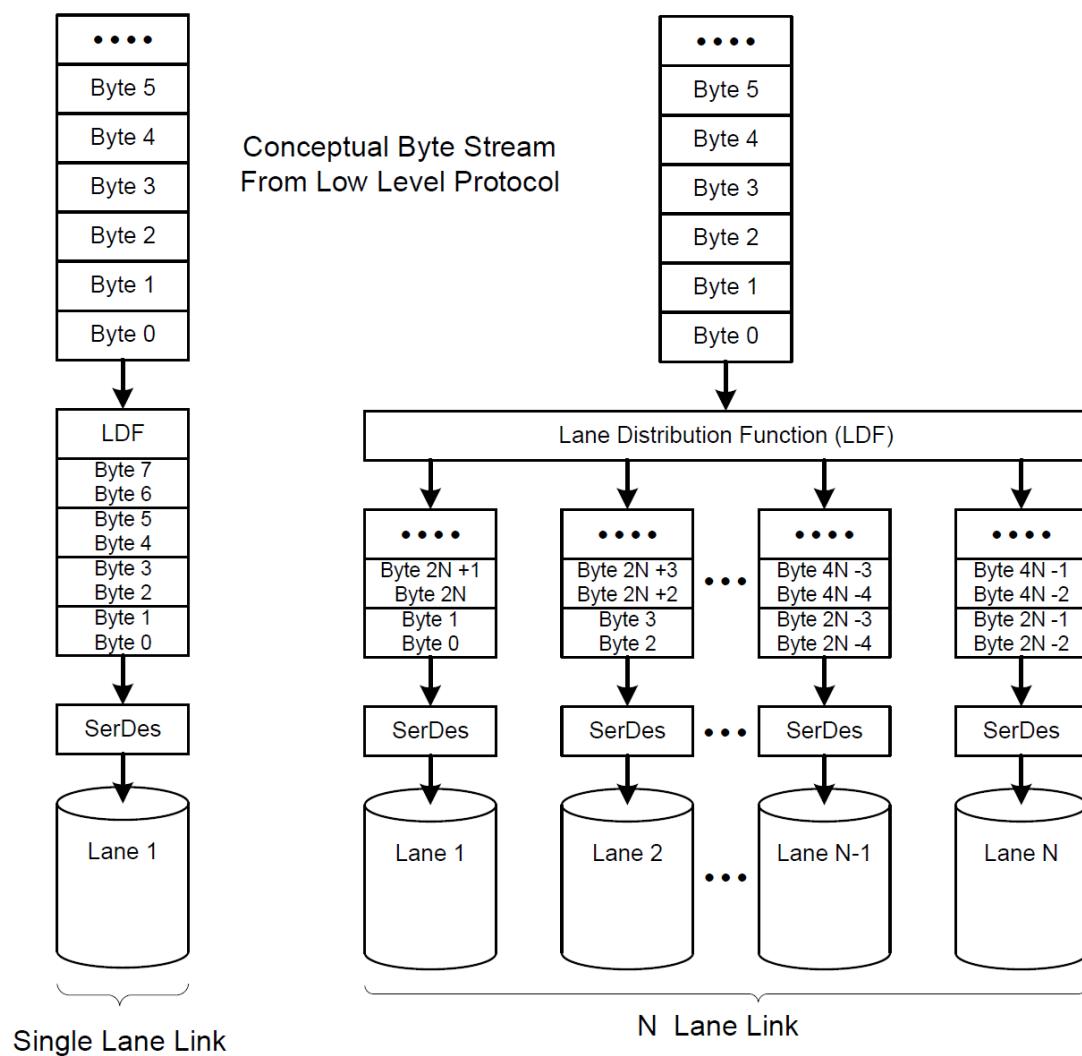
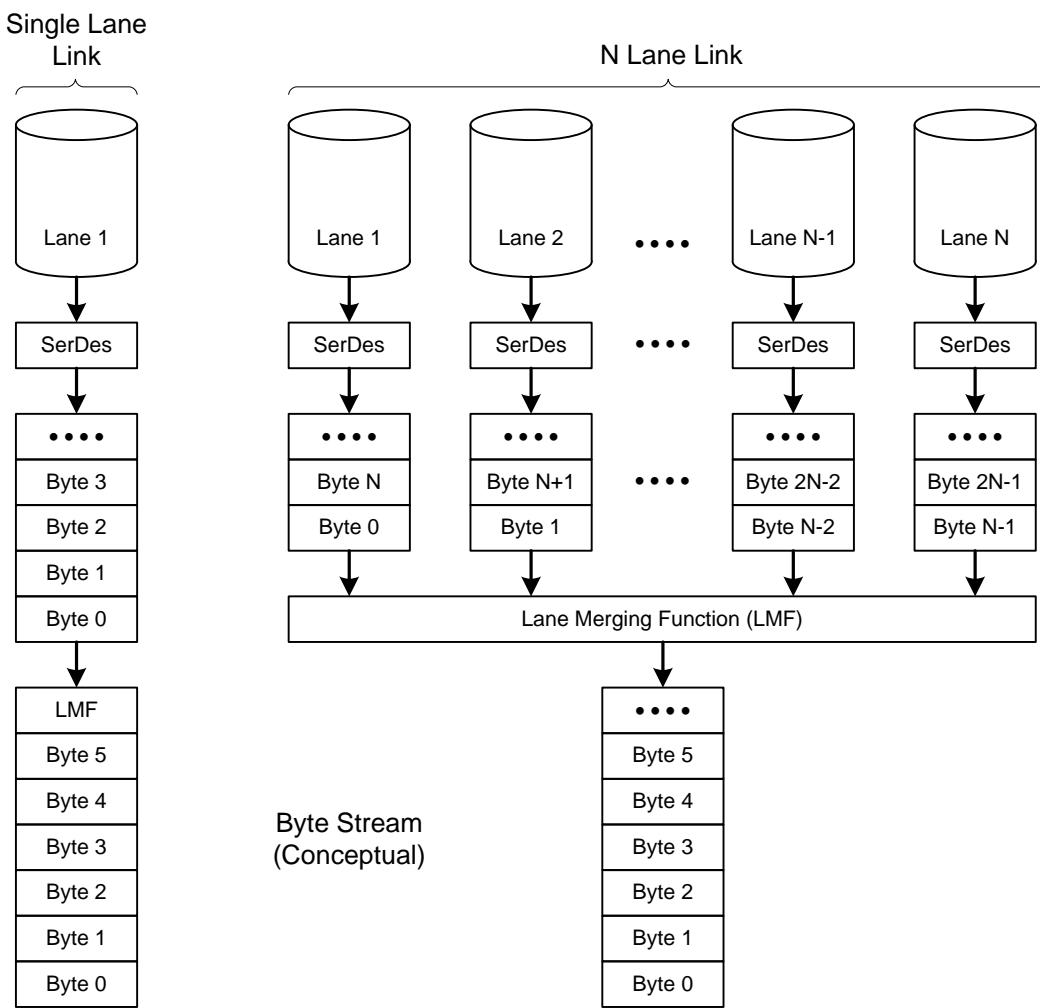


Figure 35 Conceptual Overview of the Lane Distributor Function for D-PHY

899

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



900

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

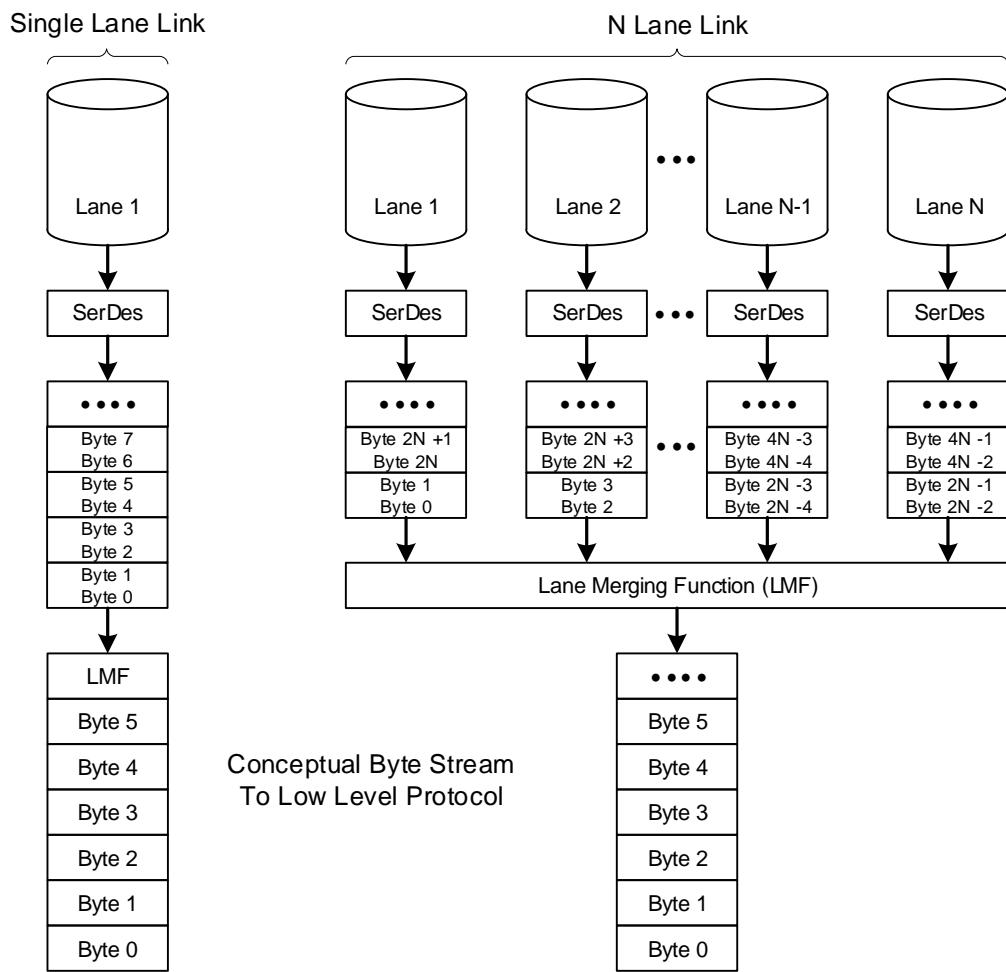


Figure 38 Conceptual Overview of the Lane Merging Function for C-PHY

The Lane distributor takes a transmission of arbitrary byte length, buffers up $N \cdot b$ bytes (where $N = \text{number of Lanes}$ and $b = 1$ or 2 for the D-PHY or C-PHY physical layer option, respectively), and then sends groups of $N \cdot b$ bytes in parallel across N Lanes with each Lane receiving b bytes. Before sending data, all Lanes perform the SoT sequence in parallel to indicate to their corresponding receiving units that the first byte of a packet is beginning. After SoT, the Lanes send groups of successive bytes from the first packet in parallel, following a round-robin process.

8.1 Lane Distribution for the D-PHY Physical Layer Option

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

- 2-Lane system (**Figure 39**): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to Lane 1, byte 3 goes to Lane 2, byte 4 goes to Lane 1, and so on.
- 3-Lane system (**Figure 40**): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to Lane 3, byte 3 goes to Lane 1, byte 4 goes to Lane 2, and so on.
- N-Lane system (**Figure 41**): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte N-1 goes to Lane N, byte N goes to Lane 1, byte N+1 goes to Lane 2, and so on.
- N-lane system (**Figure 42**) with N>4 short packet (4 bytes) transmission: byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 goes to Lane 3, byte 3 goes to Lane 4, and Lanes 5 to N do not receive bytes and stay in LPS state.

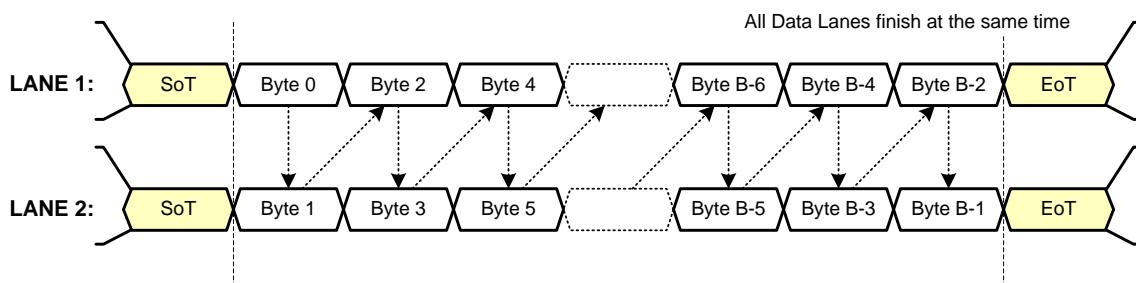
At the end of the transmission, there may be “extra” bytes since the total byte count may not be an integer multiple of the number of Lanes, N. One or more Lanes may send their last bytes before the others. The Lane distributor, as it buffers up the final set of less-than-N bytes in parallel for sending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

Each D-PHY data Lane operates autonomously.

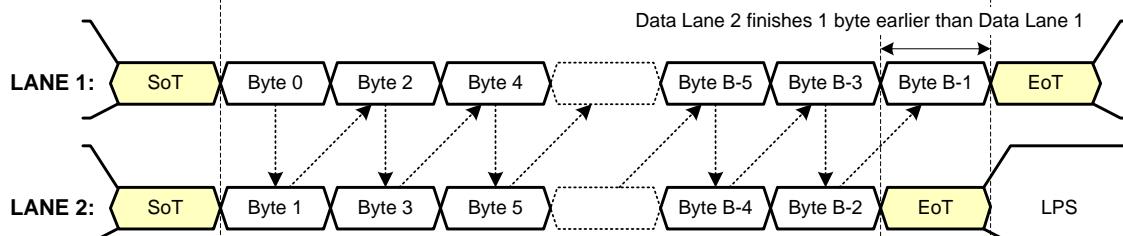
Although multiple Lanes all start simultaneously with parallel “start packet” codes, they may complete the transaction at different times, sending “end packet” codes one cycle (byte) apart.

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

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



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



KEY:

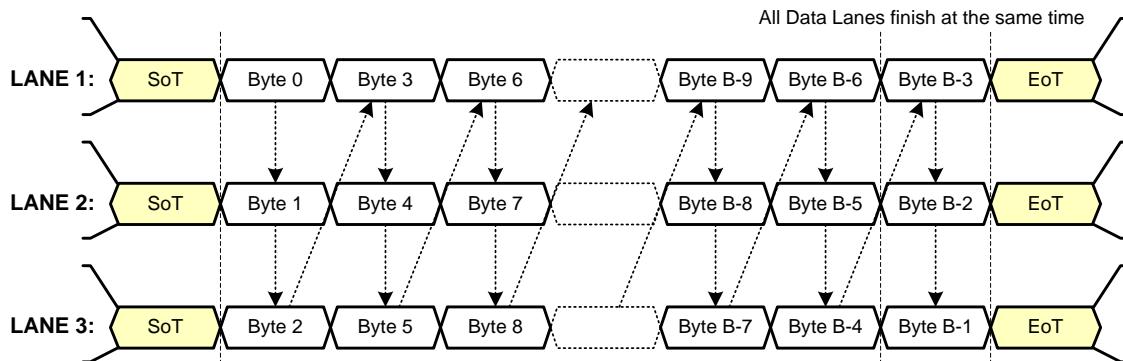
LPS – Low Power State

SoT – Start of Transmission

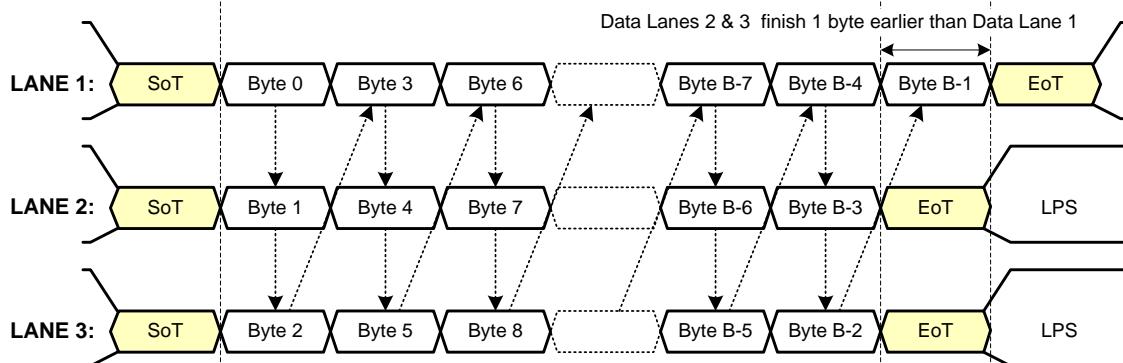
EoT – End of Transmission

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

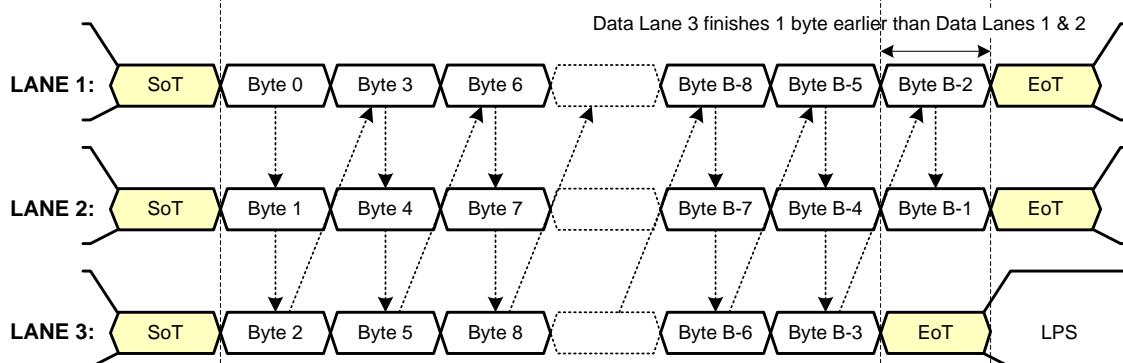
Number of Bytes, B, transmitted is an integer multiple of the number of lanes:



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



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



KEY:

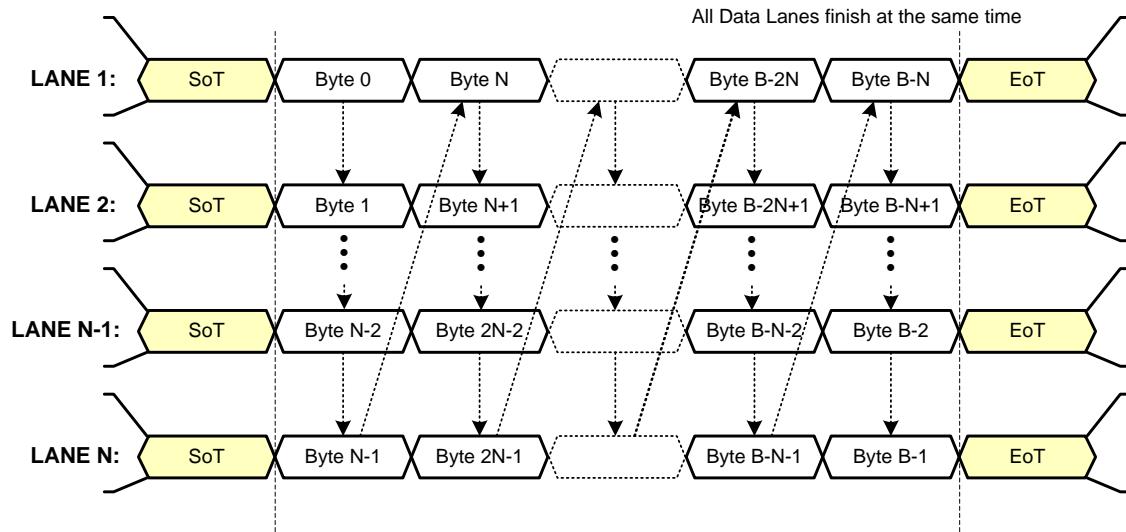
LPS – Low Power State

SoT – Start of Transmission

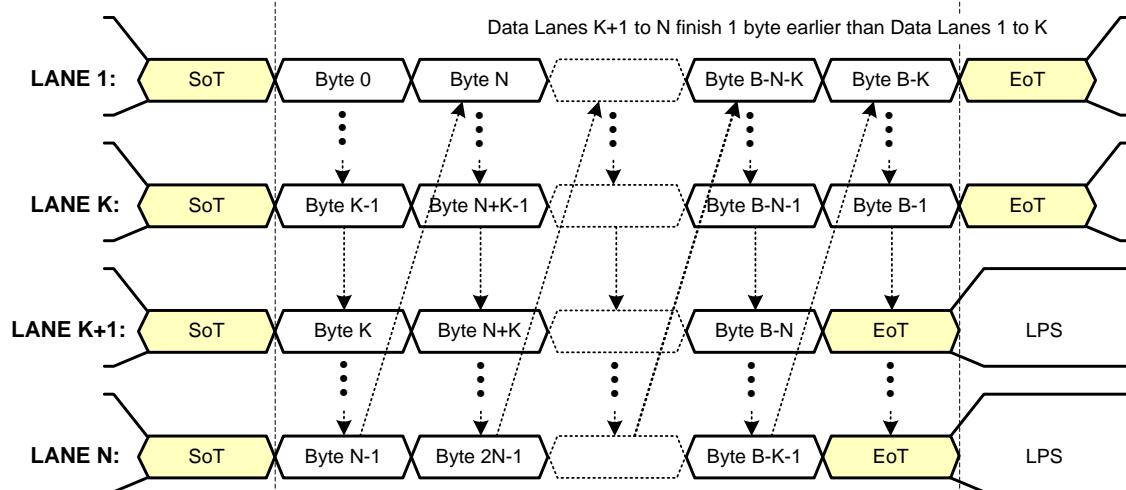
EoT – End of Transmission

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

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

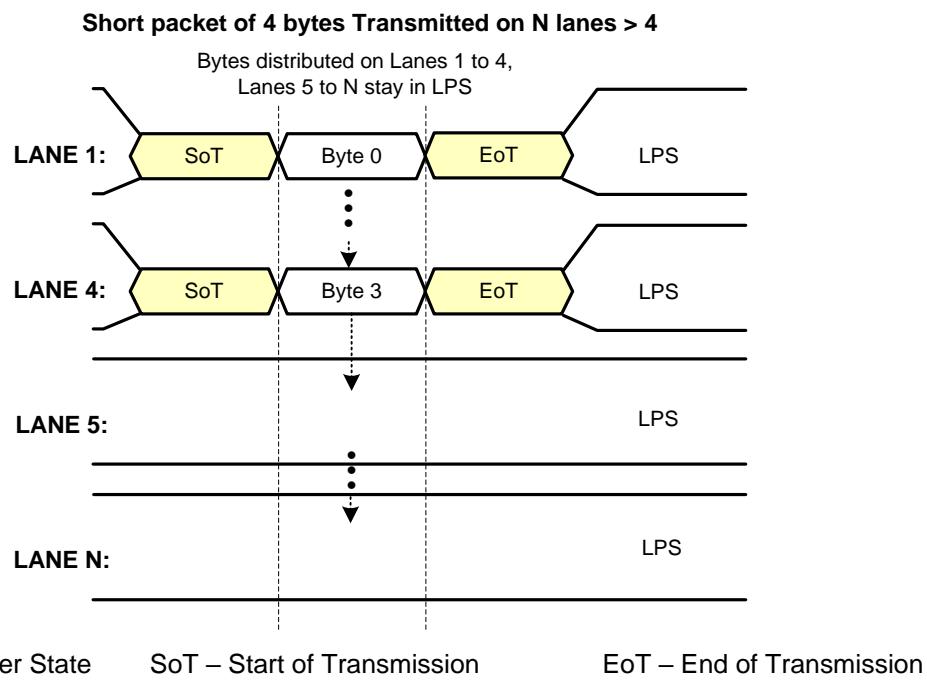


Figure 42 N-Lane Multi-Lane Example for D-PHY Short Packet Transmission

8.2 Lane Distribution for the C-PHY Physical Layer Option

934 Examples are shown in **Figure 43** and **Figure 44**:

- 935 • **2-Lane system (Figure 43):** Bytes 1 and 0 of the packet are sent as a 16-bit word to the Lane 1
 936 C-PHY module, bytes 3 and 2 are sent to Lane 2, bytes 5 and 4 are sent to Lane 1, bytes 7 and 6
 937 are sent to Lane 2, bytes 9 and 8 are sent to Lane 1, and so on.
- 938 • **3-Lane system (Figure 44):** Bytes 1 and 0 of the packet are sent as a 16-bit word to the Lane 1
 939 C-PHY module, bytes 3 and 2 are sent to Lane 2, bytes 5 and 4 are sent to Lane 3, bytes 7 and 6
 940 are sent to Lane 1, bytes 9 and 8 are sent to Lane 2, and so on.

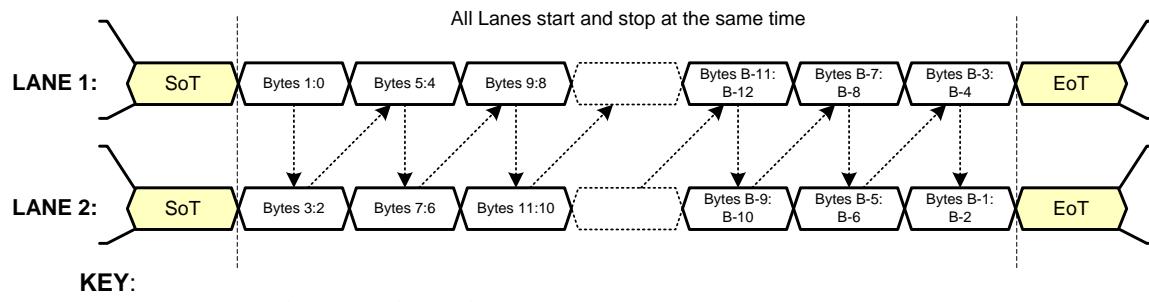
941 **Figure 45** illustrates normative behavior for an N-Lane system where $N \geq 1$: bytes 1 and 0 of the packet are
 942 sent as a 16-bit word to the Lane 1 C-PHY module, bytes 3 and 2 are sent to Lane 2, bytes $2N-1$ and $2N-2$
 943 are sent to Lane N, bytes $2N+1$ and $2N$ are sent to Lane 1, and so on. The last two bytes $B-1$ and $B-2$ are sent
 944 to Lane N, where B is the total number of bytes in the packet.

945 For an N-Lane transmitter, the C-PHY module for Lane n ($1 \leq n \leq N$) shall transmit the following sequence
 946 of $\{ms\ byte : ls\ byte\}$ byte pairs from a B-byte packet generated by the low level protocol layer: $\{\text{Byte}$
 947 $2*(k*N+n)-1 : \text{Byte } 2*(k*N+n)-2\}$, for $k = 0, 1, 2, \dots, B/(2N) - 1$, where Byte 0 is the first byte in the packet.
 948 The low level protocol shall guarantee that B is an integer multiple of $2N$.

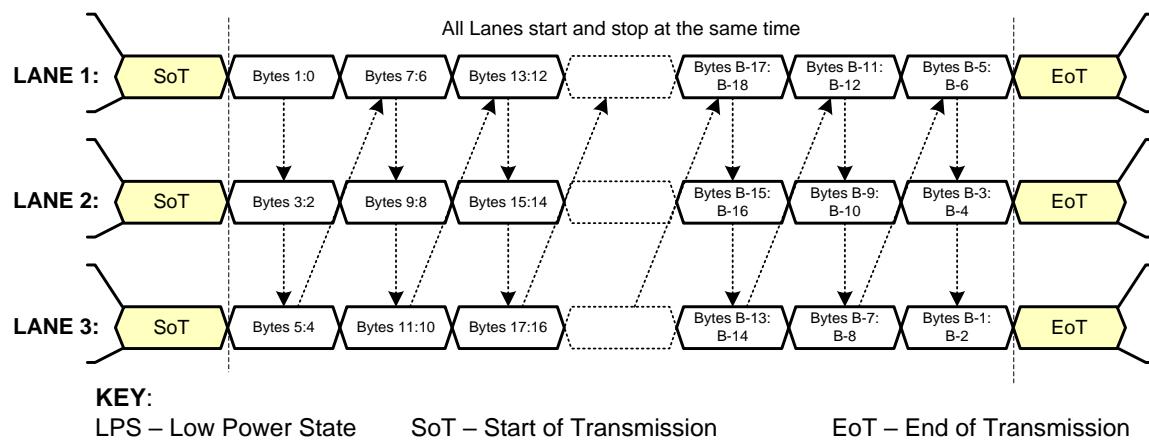
949 That is, at the end of the packet transmission, there shall be no “extra” bytes since the total byte count is
 950 always an even multiple of the number of Lanes, N. The Lane distributor, after sending the final set of $2N$
 951 bytes in parallel to the N Lanes, simultaneously de-asserts its “valid data” signal to all Lanes, signaling to
 952 each C-PHY Lane module that it may start its EoT sequence.

953 Each C-PHY Lane module operates autonomously, but packet data transmission starts and stops at the same
 954 time on all Lanes.

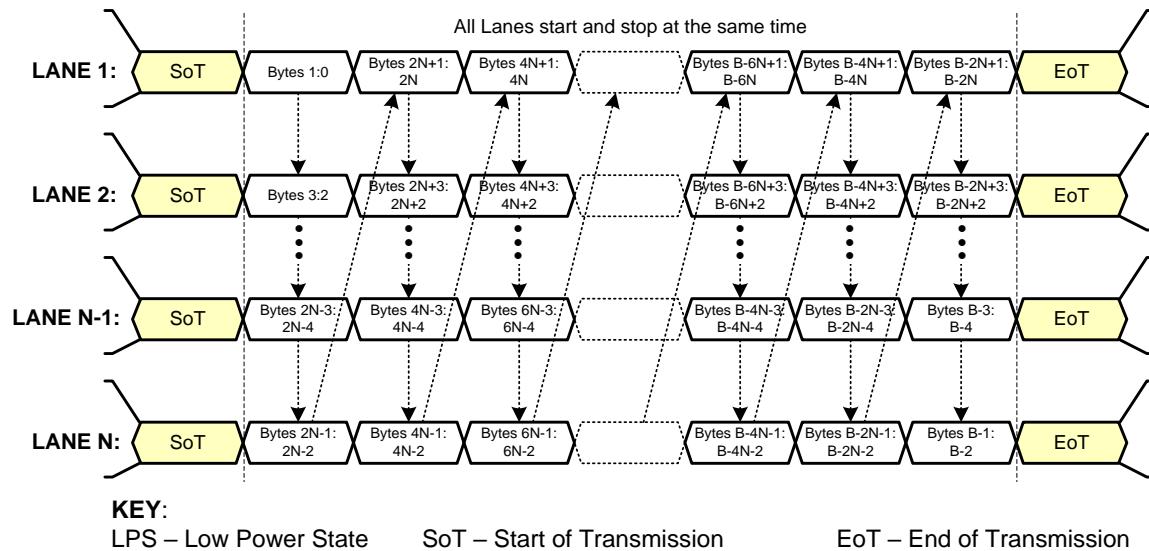
955 The N C-PHY receiver modules on the receiving end of the link collect byte pairs in parallel, and feed them
 956 into the Lane-merging layer. This reconstitutes the original sequence of bytes in the transmission, which can
 957 then be partitioned into individual packets for the packet decoder layers.



958

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

959

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

960

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

8.3 Multi-Lane Interoperability

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

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

Two cases:

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

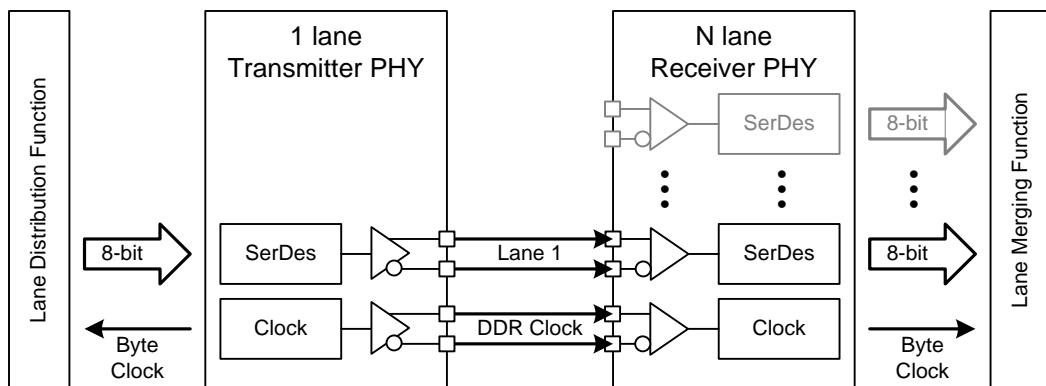


Figure 46 One Lane Transmitter and N-Lane Receiver Example for D-PHY

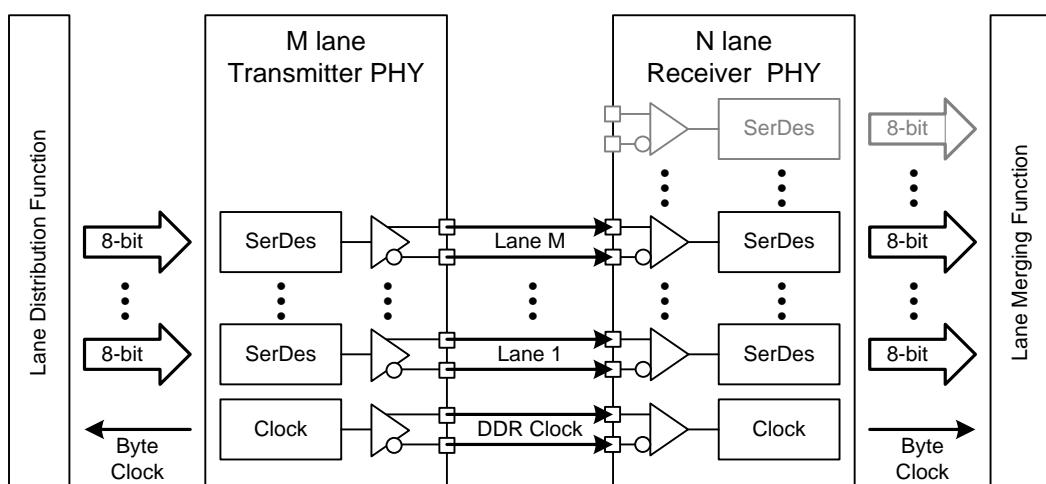
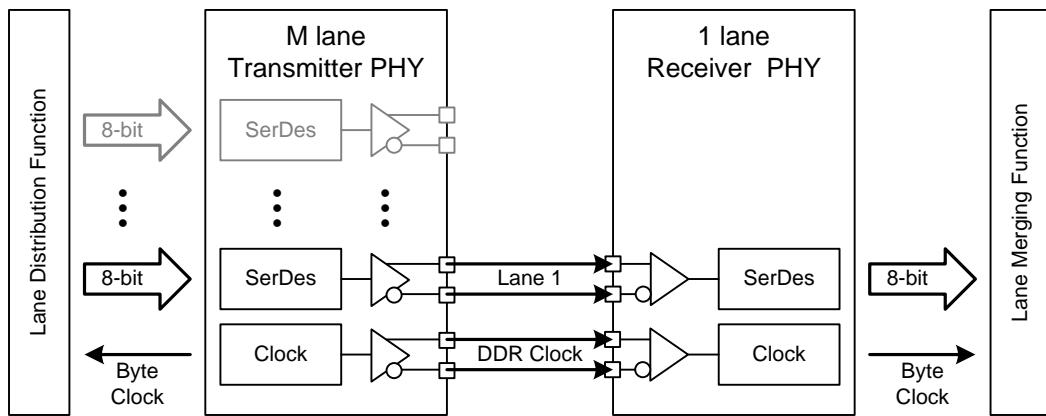
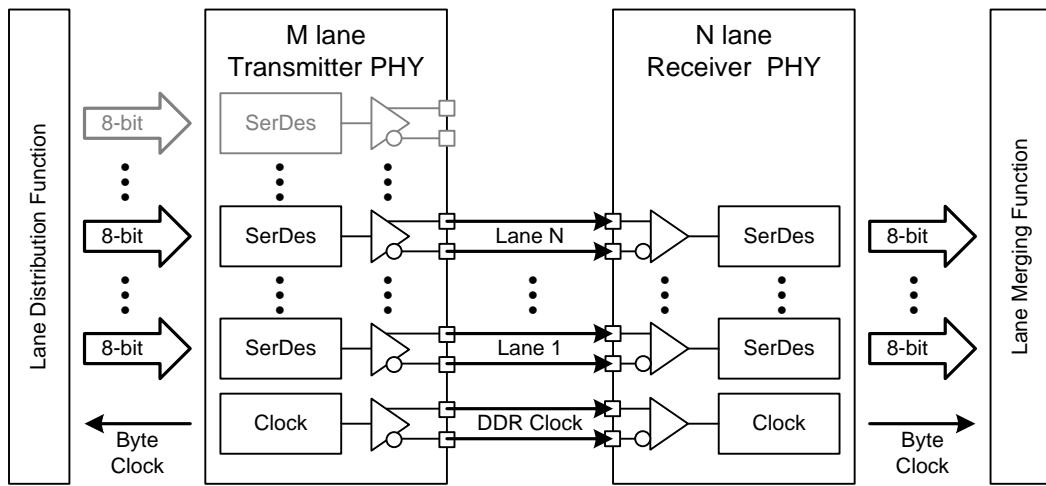


Figure 47 M-Lane Transmitter and N-Lane Receiver Example (M<N) for D-PHY



977

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



978

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

8.3.1 C-PHY Lane Deskew

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

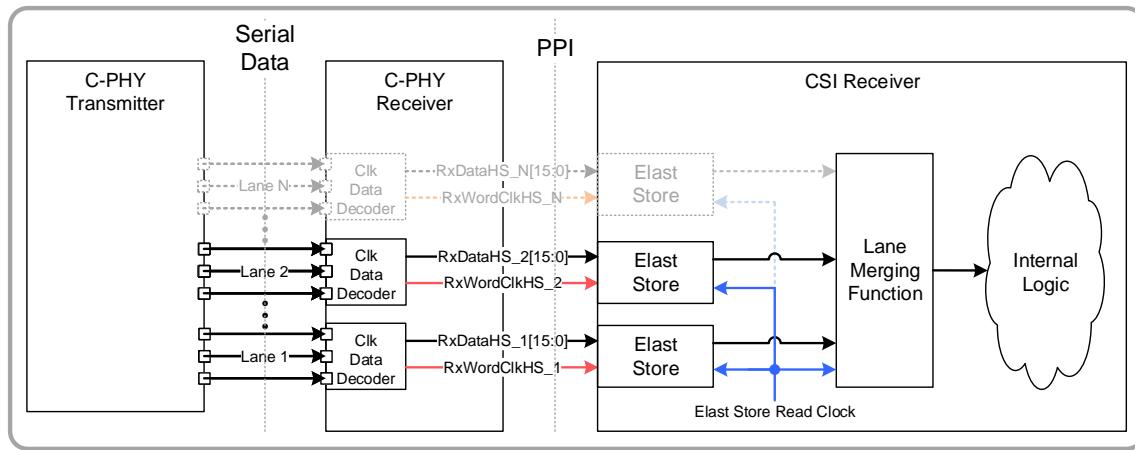


Figure 50 Example of Digital Logic to Align All RxDataHS

9 Low Level Protocol

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

Basic Low Level Protocol Features:

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

DATA:

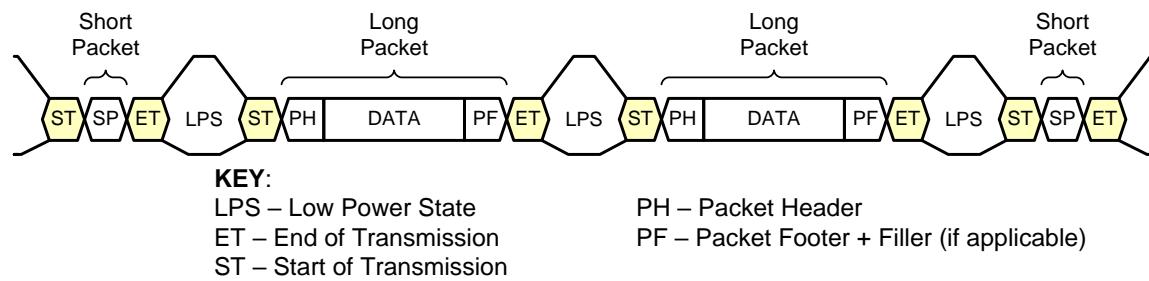


Figure 51 Low Level Protocol Packet Overview

9.1 Low Level Protocol Packet Format

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

9.1.1 Low Level Protocol Long Packet Format

Figure 52 shows the structure of the Low Level Protocol Long Packet for the D-PHY physical layer option. A Long Packet shall be identified by Data Types 0x10 to 0x38. See *Table 10* for a description of the Data Types. A Long Packet for the D-PHY physical layer option shall consist of three elements: a 32-bit Packet Header (PH), an application specific Data Payload with a variable number of 8-bit data words, and a 16-bit Packet Footer (PF). The Packet Header is further composed of four elements: an 8-bit Data Identifier, a 16-bit Word Count field, a 2-bit Virtual Channel Extension field, and a 6-bit ECC. The Packet footer has one element, a 16-bit checksum (CRC). See *Section 9.2* through *Section 9.5* for further descriptions of the packet elements.

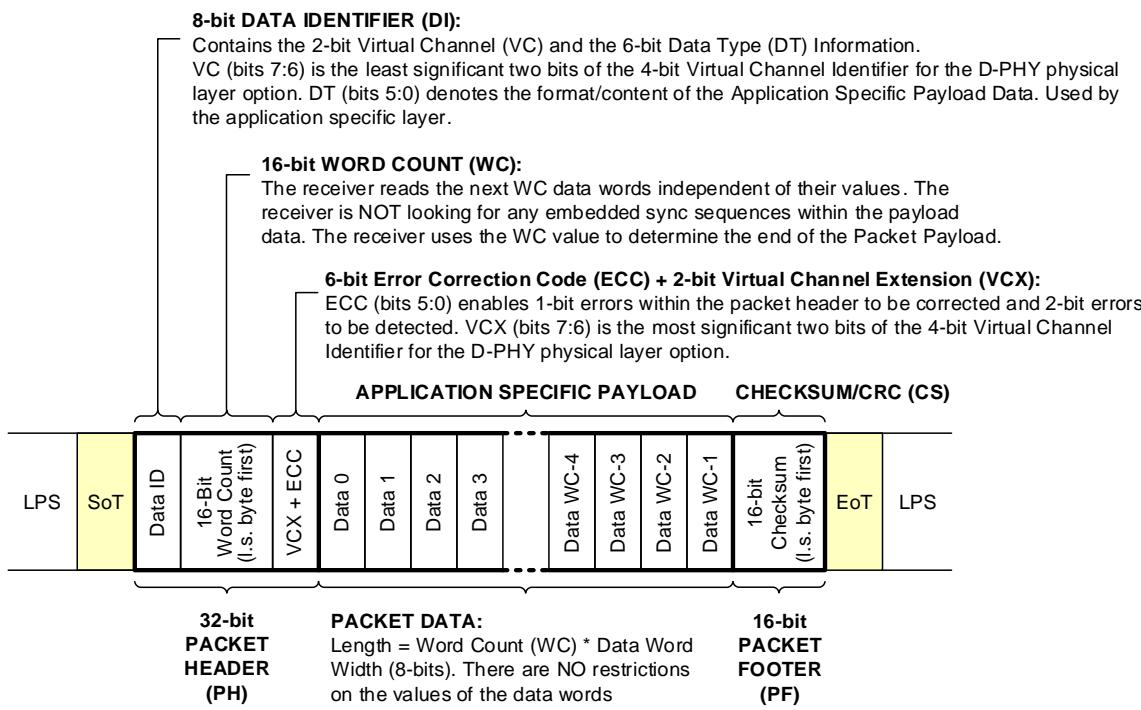
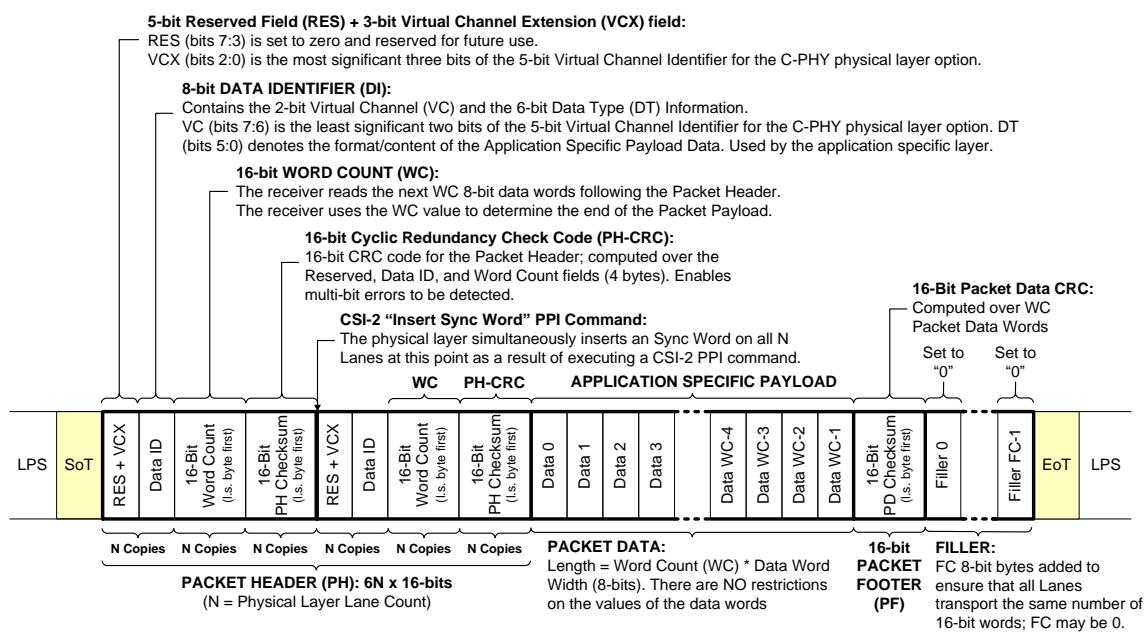


Figure 52 Long Packet Structure for D-PHY Physical Layer Option

1011
 1012 **Figure 53** shows the Long Packet structure for the C-PHY physical layer option; it shall consist of four
 1013 elements: a Packet Header (PH), an application specific Data Payload with a variable number of 8-bit data
 1014 words, a 16-bit Packet Footer (PF), and zero or more Filler bytes (FILLER). The Packet Header is $6N \times 16$ -
 1015 bits long, where N is the number of C-PHY physical layer Lanes. As shown in **Figure 53**, the Packet Header
 1016 consists of two identical $6N$ -byte halves, where each half consists of N sequential copies of each of the
 1017 following fields: a 16-bit field containing five Reserved bits, a 3-bit Virtual Channel Extension (VCX) field,
 1018 and the 8-bit Data Identifier (DI); the 16-bit Packet Data Word Count (WC); and a 16-bit Packet Header
 1019 checksum (PH-CRC) which is computed over the previous four bytes. The value of each Reserved bit shall
 1020 be zero. The Packet Footer consists of a 16-bit checksum (CRC) computed over the Packet Data using the
 1021 same CRC polynomial as the Packet Header CRC and the Packet Footer used in the D-PHY physical layer
 1022 option. Packet Filler bytes are inserted after the Packet Footer, if needed, to ensure that the Packet Footer
 1023 ends on a 16-bit word boundary and that each C-PHY physical layer Lane transports the same number of 16-
 bit words (i.e. byte pairs).



1024
Figure 53 Long Packet Structure for C-PHY Physical Layer Option

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

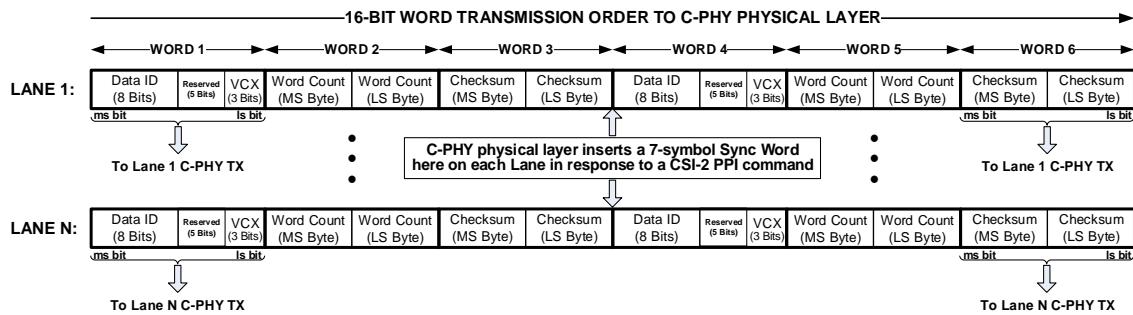


Figure 54 Packet Header Lane Distribution for C-PHY Physical Layer Option

For both physical layer options, the 8-bit Data Identifier field defines the 2-bit Virtual Channel (VC) and the Data Type for the application specific payload data. The Virtual Channel Extension (VCX) field is also common to both options, but is a 2-bit field for D-PHY and a 3-bit field for C-PHY. Together, the VC and VCX fields comprise the 4- or 5-bit Virtual Channel Identifier field which determines the Virtual Channel number associated with the packet (see **Section 9.3**).

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

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

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

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

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

Filler bytes are only inserted by the CSI-2 transmitter's low level protocol layer in conjunction with the C-PHY physical layer option. The value of any Filler byte shall be zero. If the Packet Data Word Count (WC) is an odd number (i.e. LSB is "1"), the CSI-2 transmitter shall insert one Packet Filler byte after the Packet Footer to ensure that the Packet Footer ends on a 16-bit word boundary. The CSI-2 transmitter shall also

1063 insert additional Filler bytes, if needed, to ensure that each C-PHY Lane transports the same number of 16-bit words.
1064 The latter rules require the total number of Filler bytes, FC, to be greater than or equal to $(WC \bmod 2) + \{\{N - ((WC + 2 + (WC \bmod 2)) / 2) \bmod N\} \bmod N\} * 2$, where N is the number of Lanes. Note
1065 that it is possible for FC to be zero.
1066

1067 **Figure 55** illustrates the Lane distribution of the minimal number of Filler bytes required for packets of
1068 various lengths transmitted over three C-PHY Lanes. The total number of Filler bytes required per packet
1069 ranges from 0 to 5, depending on the value of the Packet Data Word Count (WC). In general, the minimal
1070 number of Filler bytes required per packet ranges from 0 to $2N-1$ for an N-Lane C-PHY system.

1071 For the D-PHY physical layer option, the CSI-2 Lane Distributor function shall pass each byte to the physical
1072 layer which then serially transmits its least significant bit first.

1073 For the C-PHY physical layer option, the Lane Distributor function shall group each pair of consecutive bytes
1074 $2n$ and $2n+1$ (for $n \geq 0$) received from the Low Level Protocol into a 16-bit word (whose least significant
1075 byte is byte $2n$) and then pass this word to a physical layer Lane module. The C-PHY Lane module maps
1076 each 16-bit word into a 7-symbol word which it then serially transmits least significant symbol first.

1077 For both physical layer options, payload data may be presented to the Lane Distributor function in any byte
1078 order restricted only by data format requirements. Multi-byte protocol elements such as Word Count,
1079 Checksum and the Short packet 16-bit Data Field shall be presented to the Lane Distributor function least
1080 significant byte first.

1081 After the EoT sequence the receiver begins looking for the next SoT sequence.



9.1.2 Low Level Protocol Short Packet Format

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

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

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

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

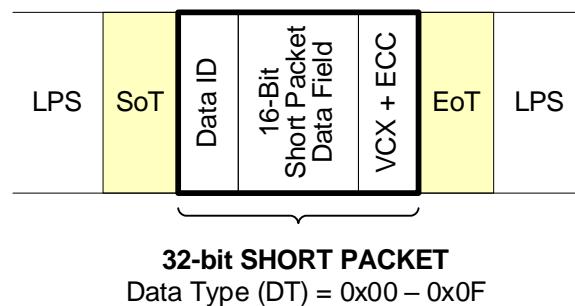


Figure 56 Short Packet Structure for D-PHY Physical Layer Option

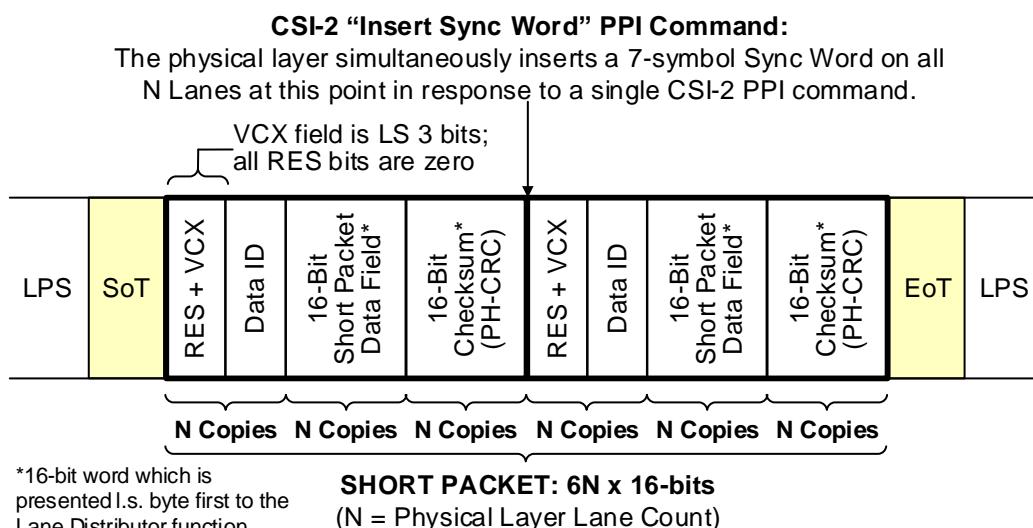


Figure 57 Short Packet Structure for C-PHY Physical Layer Option

9.2 Data Identifier (DI)

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

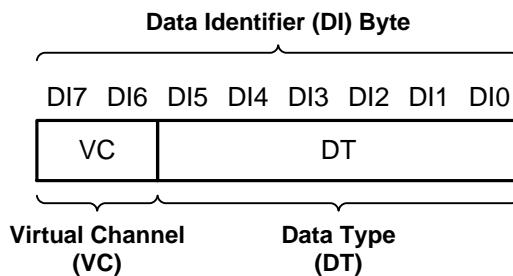


Figure 58 Data Identifier Byte

9.3 Virtual Channel Identifier

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

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

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

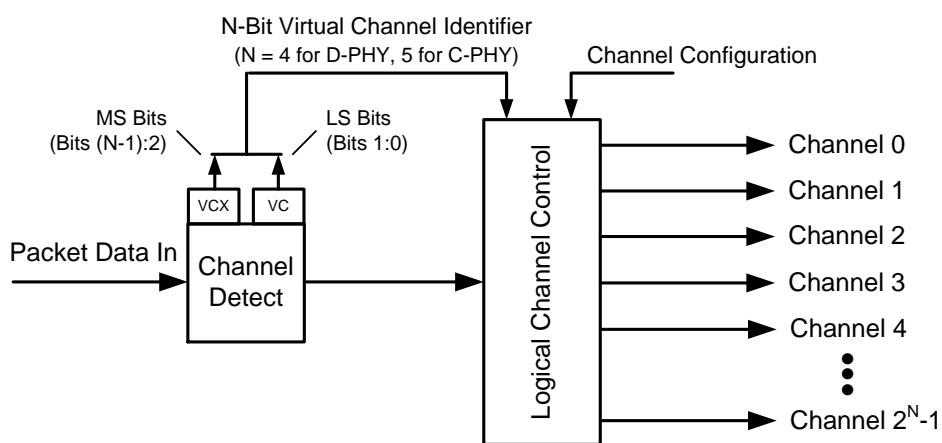
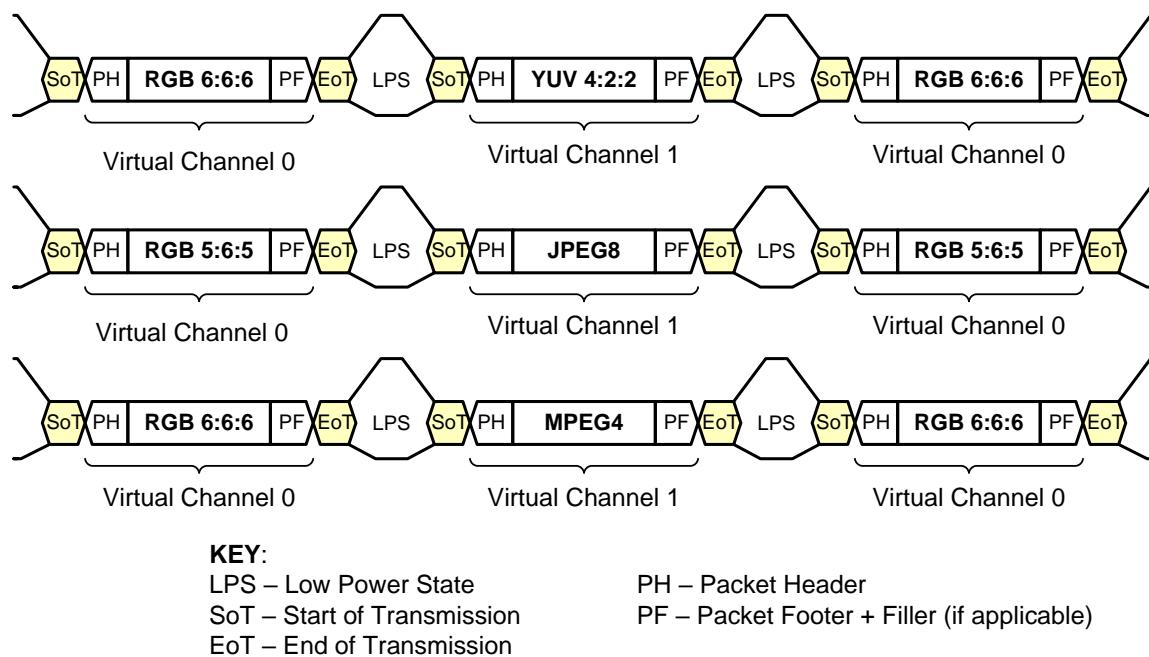


Figure 59 Logical Channel Block Diagram (Receiver)

1120

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



1121

Figure 60 Interleaved Video Data Streams Examples

9.4 Data Type (DT)

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

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

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

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

Table 10 Data Type Classes

Data Type	Description
0x00 to 0x07	Synchronization Short Packet Data Types
0x08 to 0x0F	Generic Short Packet Data Types
0x10 to 0x17	Generic Long Packet Data Types
0x18 to 0x1F	YUV Data
0x20 to 0x25	RGB Data
0x26 to 0x2F	RAW Data
0x30 to 0x37	User Defined Byte-based Data
0x38	USL Commands (See Section 9.12)
0x39	Reserved for MIPI CSE Specification [MIPI05]
0x3A to 0x3D	Reserved for future use
0x3E	Service Extension Packet as described in MIPI CSE Specification [MIPI05]
0x3F	For CSI-2 over C-PHY: Reserved for future use For CSI-2 over D-PHY: Unavailable (0x3F is used for LRTE EPD Spacer)

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

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

The Data Identifier field DI[7:0] shall map to D[7:0] of the ECC input, the Word Count LS Byte (WC[7:0]) to D[15:8], the Word Count MS Byte (WC[15:8]) to D[23:16], and the Virtual Channel Extension (VCX) field to D[25:24]. This mapping is shown in [Figure 61](#), which also serves as an ECC calculation example.

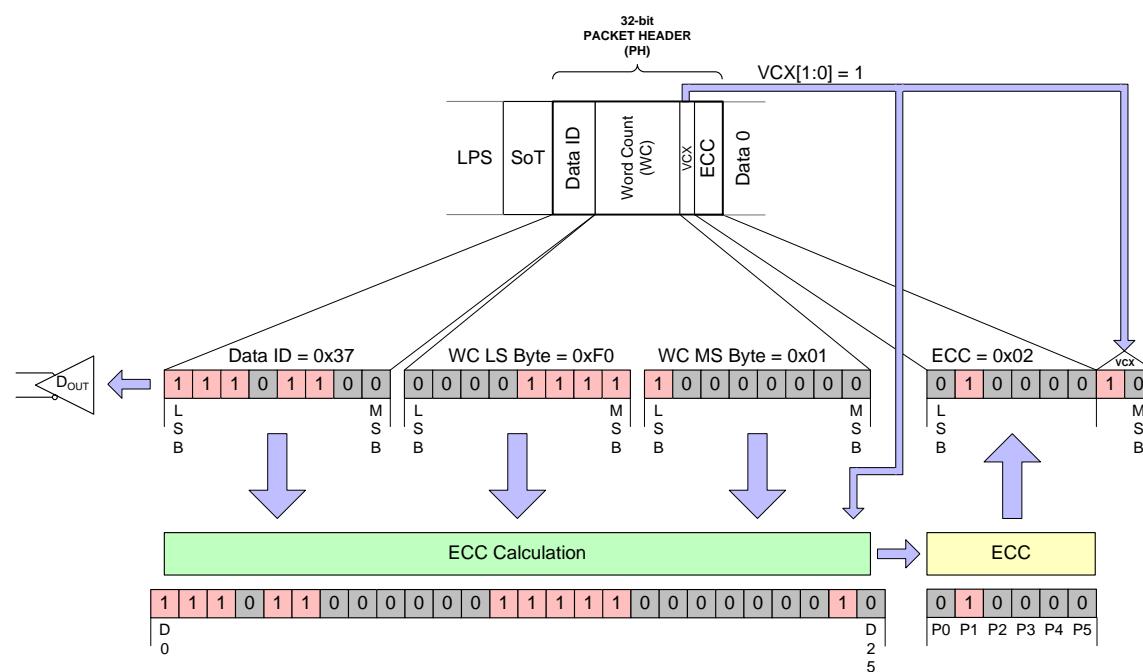


Figure 61 26-bit ECC Generation Example

9.5.1 General Hamming Code Applied to Packet Header

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

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

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

1149 $\mathbf{G} = [\mathbf{I} | \mathbf{A}]$ 1150 The packet header plus the ECC code can be obtained as: $\text{PH} = p^* \mathbf{G}$ where p represents the header (26 or 64
1151 bits) and \mathbf{G} is the corresponding generator matrix.1152 Validating the received code word r , involves multiplying it by a parity check to form s , the syndrome or
1153 parity check vector: $s = \mathbf{H}^* \text{PH}$ where PH is the received packet header and \mathbf{H} is the parity check matrix:1154 $\mathbf{H} = [\mathbf{A}^T | \mathbf{I}]$ 1155 If all elements of s are zero, the code word was received correctly. If s contains non-zero elements, then at
1156 least one error is present. If a single bit error is encountered then the syndrome s is one of the elements of \mathbf{H}
1157 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
1158 syndrome will be one of the elements on \mathbf{I} , else it will be the data bit identified by the position of the syndrome
1159 in \mathbf{A}^T .

9.5.2 Hamming-Modified Code

1160 The error correcting code used is a 7+1 bits Hamming-modified code (72,64) and the subset of it is 5+1 bits
1161 or (32,26). Hamming codes use parity to correct one error or detect two errors, but they are not capable of
1162 doing both simultaneously, thus one extra parity bit is added. The code used allows the same 6-bit syndromes
1163 to correct the first 26-bits of a 64-bit sequence. To specify a compact encoding of parity and decoding of
1164 syndromes, the matrix shown in *Table 11* is used:1165 **Table 11 ECC Syndrome Association Matrix**

		d2d1d0							
d5d4d3		0b000	0b001	0b010	0b011	0b100	0b101	0b110	0b111
0b000		0x07	0x0B	0x0D	0x0E	0x13	0x15	0x16	0x19
0b001		0x1A	0x1C	0x23	0x25	0x26	0x29	0x2A	0x2C
0b010		0x31	0x32	0x34	0x38	0x1F	0x2F	0x37	0x3B
0b011		0x3D	0x3E	0x46	0x49	0x4A	0x4C	0x51	0x52
0b100		0x54	0x58	0x61	0x62	0x64	0x68	0x70	0x83
0b101		0x85	0x86	0x89	0x8A	0x43	0x45	0x4F	0x57
0b110		0x8C	0x91	0x92	0x94	0x98	0xA1	0xA2	0xA4
0b111		0xA8	0xB0	0xC1	0xC2	0xC4	0xC8	0xD0	0xE0

1166 Each cell in the matrix represents a syndrome, and the first 26 cells (the orange cells) use the first three or
1167 five bits to build the syndrome. Each syndrome in the matrix is MSB left aligned:

1168 e.g. 0x07 = 0b0000_0111 = P7 P6 P5 P4 P3 P2 P1 P0

1169 The top row defines the three LSB of data position bit, and the left column defines the three MSB of data
1170 position bit (there are 64-bit positions in total).

1171 e.g. 37th bit position is encoded 0b100_101 and has the syndrome 0x68.

1172 To derive the parity P0 for 26-bits, the P0's in the orange cells will define whether the corresponding bit
1173 position is used in P0 parity or not.

1174 e.g. $P_{0\text{24-bits}} = D0 \wedge D1 \wedge D2 \wedge D4 \wedge D5 \wedge D7 \wedge D10 \wedge D11 \wedge D13 \wedge D16 \wedge D20 \wedge D21 \wedge D22 \wedge D23 \wedge D24$

1175 Similarly, to derive the parity P0 for 64-bits, all P0's in **Table 12** will define the corresponding bit positions
1176 to be used.

1177 To correct a single data bit error, the syndrome must be one of the syndromes in **Table 11**. These syndromes
1178 identify the bit position in error. The syndrome is calculated as:

1179 $S = P_{\text{SEND}} \wedge P_{\text{RECEIVED}}$, where P_{SEND} is the 8/6-bit ECC field in the header and P_{RECEIVED} is the
1180 calculated parity of the received header.

1181 **Table 12** represents the same information as the matrix in **Table 11**, organized so as to provide better insight
1182 into the way in which parity bits are formed out of data bits. The orange area of the table is used to form the
1183 ECC needed to protect a 26-bit header, whereas the whole table must be used to protect a 64-bit header.

1184 Previous CSI-2 specification versions not supporting the Virtual Channel Extension (VCX) field utilize a 30-
1185 bit Hamming-modified code word with 24 data bits and 5+1 parity bits based on the first 24 bit positions of
1186 **Table 12** [i.e. a (30,24) ECC]. Packet Header bits 24 and 25 are set to zero by transmitters, and ignored by
1187 receivers conforming to such Specifications.

1188 When receiving Packet Headers with a (30,24) ECC, receivers conforming to this CSI-2 Specification version
1189 shall ignore the contents of bits 24 and 25 in such Packet Headers. The intent is for such receivers to ignore
1190 any errors occurring at these bit positions, in order to match the behavior of previous receivers. (See **Section**
1191 **9.5.4** for implementation recommendations.)

1192

Table 12 ECC Parity Generation Rules

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
0	0	0	0	0	0	1	1	1	0x07
1	0	0	0	0	1	0	1	1	0x0B
2	0	0	0	0	1	1	0	1	0x0D
3	0	0	0	0	1	1	1	0	0x0E
4	0	0	0	1	0	0	1	1	0x13
5	0	0	0	1	0	1	0	1	0x15
6	0	0	0	1	0	1	1	0	0x16
7	0	0	0	1	1	0	0	1	0x19
8	0	0	0	1	1	0	1	0	0x1A
9	0	0	0	1	1	1	0	0	0x1C
10	0	0	1	0	0	0	1	1	0x23
11	0	0	1	0	0	1	0	1	0x25
12	0	0	1	0	0	1	1	0	0x26
13	0	0	1	0	1	0	0	1	0x29
14	0	0	1	0	1	0	1	0	0x2A
15	0	0	1	0	1	1	0	0	0x2C
16	0	0	1	1	0	0	0	1	0x31
17	0	0	1	1	0	0	1	0	0x32
18	0	0	1	1	0	1	0	0	0x34
19	0	0	1	1	1	0	0	0	0x38
20	0	0	0	1	1	1	1	1	0x1F
21	0	0	1	0	1	1	1	1	0x2F
22	0	0	1	1	0	1	1	1	0x37
23	0	0	1	1	1	0	1	1	0x3B
24	0	0	1	1	1	1	0	1	0x3D
25	0	0	1	1	1	1	1	0	0x3E
26	0	1	0	0	0	1	1	0	0x46
27	0	1	0	0	1	0	0	1	0x49
28	0	1	0	0	1	0	1	0	0x4A
29	0	1	0	0	1	1	0	0	0x4C
30	0	1	0	1	0	0	0	1	0x51
31	0	1	0	1	0	0	1	0	0x52

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
32	0	1	0	1	0	1	0	0	0x54
33	0	1	0	1	1	0	0	0	0x58
34	0	1	1	0	0	0	0	1	0x61
35	0	1	1	0	0	0	1	0	0x62
36	0	1	1	0	0	1	0	0	0x64
37	0	1	1	0	1	0	0	0	0x68
38	0	1	1	1	0	0	0	0	0x70
39	1	0	0	0	0	0	1	1	0x83
40	1	0	0	0	0	1	0	1	0x85
41	1	0	0	0	0	1	1	0	0x86
42	1	0	0	0	1	0	0	1	0x89
43	1	0	0	0	1	0	1	0	0x8A
44	0	1	0	0	0	0	1	1	0x43
45	0	1	0	0	0	1	0	1	0x45
46	0	1	0	0	1	1	1	1	0x4F
47	0	1	0	1	0	1	1	1	0x57
48	1	0	0	0	1	1	0	0	0x8C
49	1	0	0	1	0	0	0	1	0x91
50	1	0	0	1	0	0	1	0	0x92
51	1	0	0	1	0	1	0	0	0x94
52	1	0	0	1	1	0	0	0	0x98
53	1	0	1	0	0	0	0	1	0xA1
54	1	0	1	0	0	0	1	0	0xA2
55	1	0	1	0	0	1	0	0	0xA4
56	1	0	1	0	1	0	0	0	0xA8
57	1	0	1	1	0	0	0	0	0xB0
58	1	1	0	0	0	0	0	1	0xC1
59	1	1	0	0	0	0	1	0	0xC2
60	1	1	0	0	0	1	0	0	0xC4
61	1	1	0	0	1	0	0	0	0xC8
62	1	1	0	1	0	0	0	0	0xD0
63	1	1	1	0	0	0	0	0	0xE0

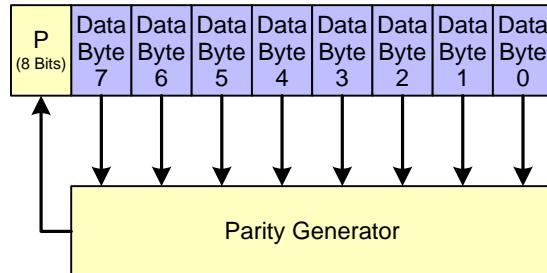
9.5.3 ECC Generation on TX Side

1193

This is an informative section.

1194

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

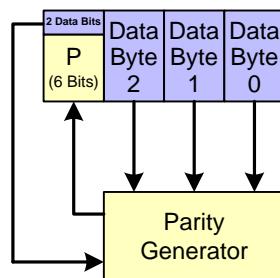


1195

Figure 62 64-bit ECC Generation on TX Side

1196

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



1197

Figure 63 26-bit ECC Generation on TX Side

1198

The parity generators are based on *Table 12*.

1199

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

1200

For backwards-compatibility, transmitters conforming to this CSI-2 Specification version should always set Packet Header bits 24 and 25 (the VCX field) to zero in any packets sent to receivers conforming to previous CSI-2 Specification versions incorporating a (30,24) ECC.

1201

1202

9.5.4 Applying ECC on RX Side (Informative)

Applying ECC on RX side involves generating a new ECC for the received Packet Header, computing the syndrome using the new ECC and the received ECC, decoding the syndrome to find if a single-error has occurred, and if so, correcting it. **Figure 64** depicts ECC processing for 64 received Packet Header data bits, using 8 parity bits.

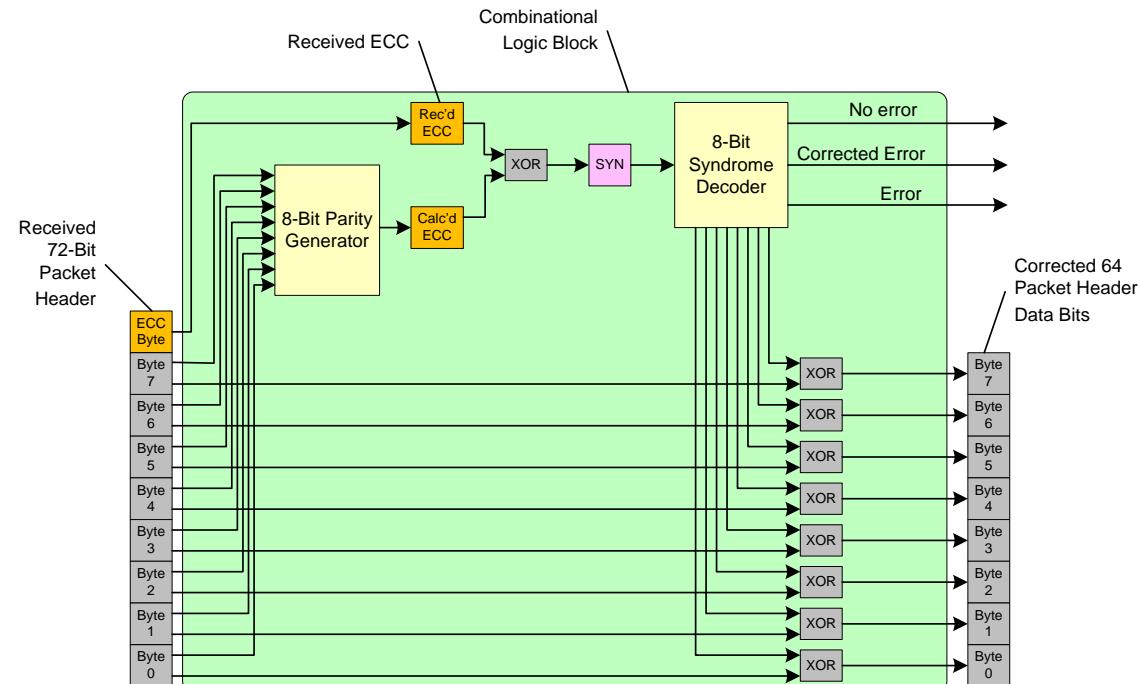


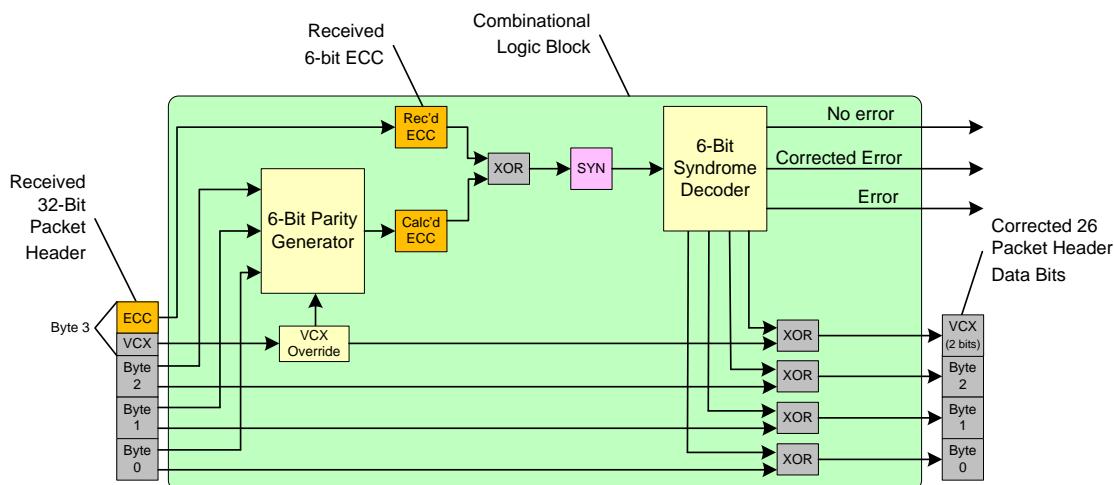
Figure 64 64-bit ECC on RX Side Including Error Correction

Decoding the syndrome has four possible outcomes:

1. If the syndrome is 0, no errors are present.
2. If the syndrome matches one of the matrix entries in the **Table 11**, then a single bit error has occurred and the corresponding bit position may be corrected by inverting it (e.g. by XORing with '1').
3. If the syndrome has only one bit set, then a single bit error has occurred at the parity bit located at that syndrome bit position, and the rest of the received packet header bits are error-free.
4. If the syndrome does not fit any of the other outcomes, then an uncorrectable error has occurred, and an error flag should be set (indicating that the Packet Header is corrupted).

The 26-bit implementation shown in **Figure 65** uses fewer terms to calculate the parity, and thus the syndrome decoding block is much simpler than the 64-bit implementation.

Receivers conforming to this CSI-2 Specification version that receive Packet Headers from transmitters without the VCX field should forcibly set received bits 24 and 25 to zero in such Packet Headers prior to any parity generation or syndrome decoding (this is the function of the "VCX Override" block shown in **Figure 65**). This guarantees that the receiver will properly ignore any errors occurring at bit positions 24 and 25, in order to match the behavior of receivers conforming to previous versions of this Specification.



1224

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

9.6 Checksum Generation

To detect possible errors in transmission, a checksum is calculated over the WC bytes composing the Packet Data of every Long Packet; a similar checksum is calculated over the four bytes composing the Reserved, Virtual Channel Extension, Data Identifier, and Word Count fields of every Packet Header for the C-PHY physical layer option. In all cases, the checksum is realized as 16-bit CRC based on the generator polynomial $x^{16}+x^{12}+x^5+x^0$ and is computed over bytes in the order in which they are presented to the Lane Distributor function by the low level protocol layer as shown in *Figure 52*, *Figure 53*, and *Figure 57*.

The order in which the checksum bytes are presented to the Lane Distributor function is illustrated in *Figure 66*.

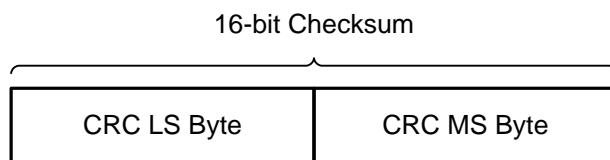


Figure 66 Checksum Transmission Byte Order

When computed over the Packet Data words of a Long Packet, the 16-bit checksum sequence is transmitted as part of the Packet Footer. When the Word Count is zero, the CRC shall be 0xFFFF. When computed over the Reserved, Virtual Channel Extension, Data Identifier, and Word Count fields of a Packet Header for the C-PHY physical layer option, the 16-bit checksum sequence is transmitted as part of the Packet Header CRC (PH-CRC) field.

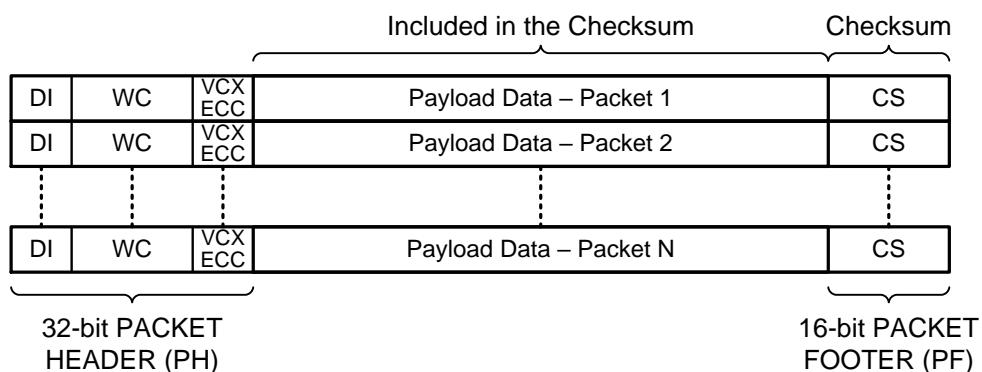
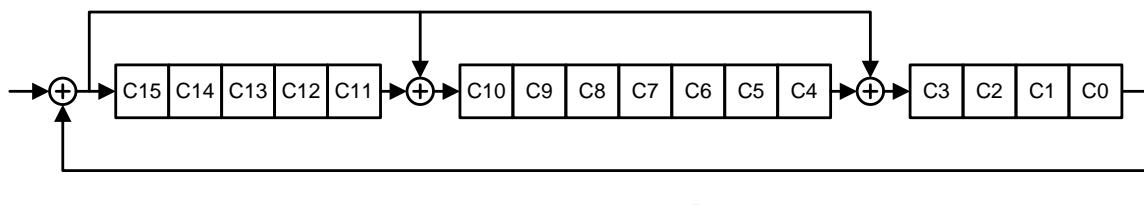


Figure 67 Checksum Generation for Long Packet Payload Data

The definition of a serial CRC implementation is presented in *Figure 68*. The CRC implementation shall be functionally equivalent with the C code presented in *Figure 69*. The CRC shift register is initialized to 0xFFFF at the beginning of each packet. Note that for the C-PHY physical layer option, if the same circuitry is used to compute both the Packet Header and Packet Footer CRC, the CRC shift register shall be initialized twice per packet, i.e. once at the beginning of the packet and then again following the computation of the Packet Header CRC. After all payload data has passed through the CRC circuitry, the CRC circuitry contains the checksum. The 16-bit checksum produced by the C code in *Figure 69* equals the final contents of the C[15:0] shift register shown in *Figure 68*. The checksum is then transmitted by the CSI-2 physical layer to the CSI-2 receiver to verify that no errors have occurred in the transmission.



1249

Figure 68 Definition of 16-bit CRC Shift Register

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

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

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

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

    return (unsigned short)(crc);
}
```

1250

Figure 69 16-bit CRC Software Implementation Example

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

1254

CRC Computation Examples:

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

Input Data Bytes:
FF 00 00 00 1E F0 1E C7 4F 82 78 C5 82 E0 8C 70 D2 3C 78 E9 FF 00 00 01
Checksum LS byte and MS byte:
69 E5

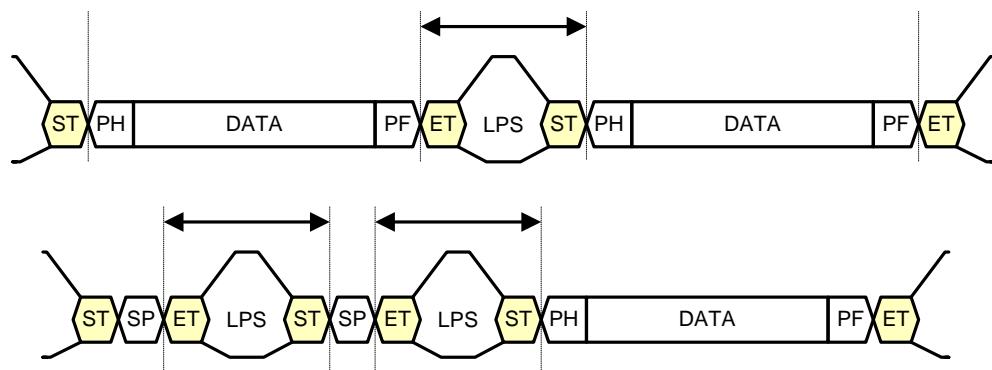
9.7 Packet Spacing

All CSI-2 implementations shall support a transition into and out of the Low Power State (LPS) between Low Level Protocol packets; however, implementations may optionally remain in the High-Speed State between packets as described in [Section 9.11](#). [Figure 70](#) illustrates the packet spacing with the LPS.

The packet spacing illustrated in [Figure 70](#) does not have to be a multiple of 8-bit data words, as the receiver will resynchronize to the correct byte boundary during the SoT sequence prior to the Packet Header of the next packet.

SHORT / LONG PACKET SPACING:

Variable - always a LPS between packets



KEY:

LPS – Low Power State

ST – Start of Transmission

ET – End of Transmission

PH – Packet Header

PF – Packet Footer + Filler (if applicable)

SP – Short Packet

Figure 70 Packet Spacing

9.8 Synchronization Short Packet Data Type Codes

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

Table 13 Synchronization Short Packet Data Type Codes

Data Type	Description
0x00	Frame Start Code
0x01	Frame End Code
0x02	Line Start Code (Optional)
0x03	Line End Code (Optional)
0x04	End of Transmission Code (Optional) See <i>Section 9.11.1.2.5</i> for description.
0x05 to 0x07	Reserved

9.8.1 Frame Synchronization Packets

Each image frame shall begin with a Frame Start (FS) Packet containing the Frame Start Code. The FS Packet shall be followed by one or more long packets containing image data and zero or more short packets containing synchronization codes. Each image frame shall end with a Frame End (FE) Packet containing the Frame End Code. See *Table 13* for a description of the synchronization code data types.

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

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

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

- Frame number is always zero – frame number is inoperative.
- Frame number increments by 1 or 2 for every FS packet with the same Virtual Channel and is periodically reset to one; e.g. 1, 2, 1, 2, 1, 2, 1, 2 or 1, 2, 3, 4, 1, 2, 3, 4 or 1, 3, 5, 1, 3, 5 or 1, 2, 4, 1, 3, 4. Frame number may be incremented by 2 only when an image frame is masked (i.e. not transmitted) due to corruption. To accommodate such cases, increments by 1 or 2 may be freely intermixed within a sequence of frame numbers as needed.

9.8.2 Line Synchronization Packets

Line synchronization short packets are optional on a per-image-frame basis. If an image frame includes Line Start (LS) and Line End (LE) line synchronization short packets with one long packet having a given data type and virtual channel number, then it shall include LS and LE short packets with all long packets having that same data type and virtual channel number within the same image frame.

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

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

The behavior of the 16-bit line number within the same Data Type and Virtual Channel shall be one of the following.

Either:

1. Line number is always zero – line number is inoperative.

Or:

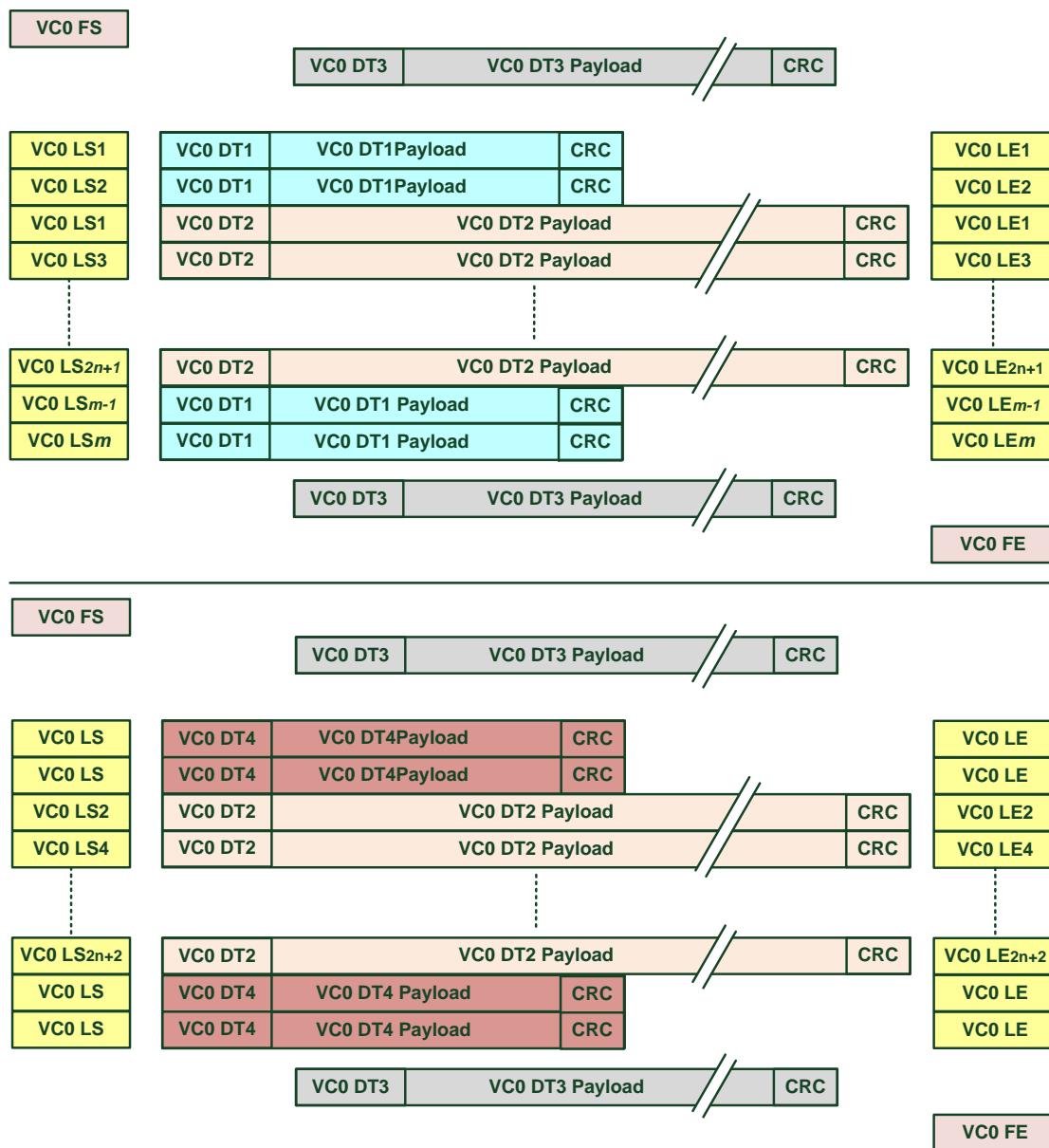
2. Line number increments by one for every LS packet within the same Virtual Channel and the same Data Type. The line number is periodically reset to one for the first LS packet after a FS packet. The intended usage is for progressive scan (non-interlaced) video data streams. The line number must be a non-zero value.

Or:

3. Line number increments by the same arbitrary step value greater than one for every LS packet within the same Virtual Channel and the same Data Type. The line number is periodically reset to a non-zero arbitrary start value for the first LS packet after a FS packet. The arbitrary start value may be different between successive frames. The intended usage is for interlaced video data streams.

Figure 71 contains examples for the use of optional LS/LE packets within an interlaced frame with pixel data and additional embedded types. The Figure illustrates the use cases:

1. VC0 DT2 Interlaced frame with line counting incrementing by two. Frame1 starting at 1 and Frame2 starting at 2.
2. VC0 DT1 Progressive scan frame with line counting.
3. VC0 DT4 Progressive scan frame with non-operative line counting.
4. VC0 DT3 No LS/LE operation.

**Note:**

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

1321

Figure 71 Example Interlaced Frame Using LS/LE Short Packet and Line Counting

9.9 Generic Short Packet Data Type Codes

1322

Table 14 lists the Generic Short Packet Data Types.

1323

Table 14 Generic Short Packet Data Type Codes

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

1324

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.

1325

1326

1327

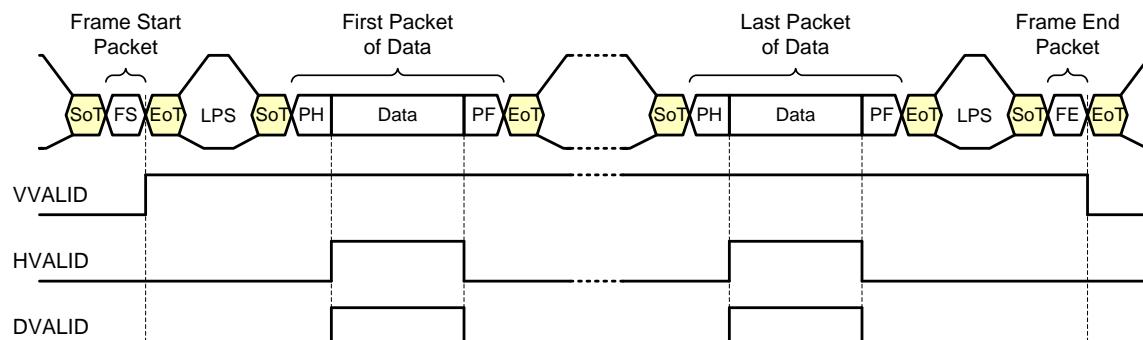
1328

9.10 Packet Spacing Examples Using the Low Power State

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

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

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

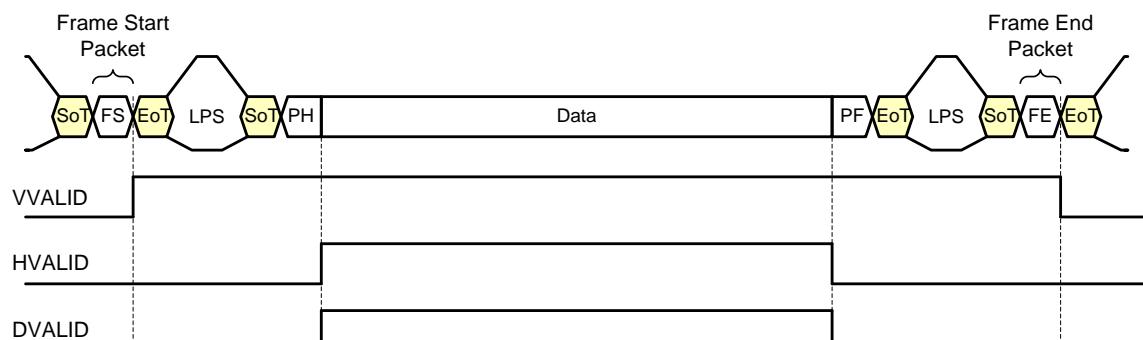


KEY:

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

EoT – End of Transmission LPS – Low Power State
PF – Packet Footer + Filler (if applicable)
FE – Frame End
LE – Line End

Figure 72 Multiple Packet Example

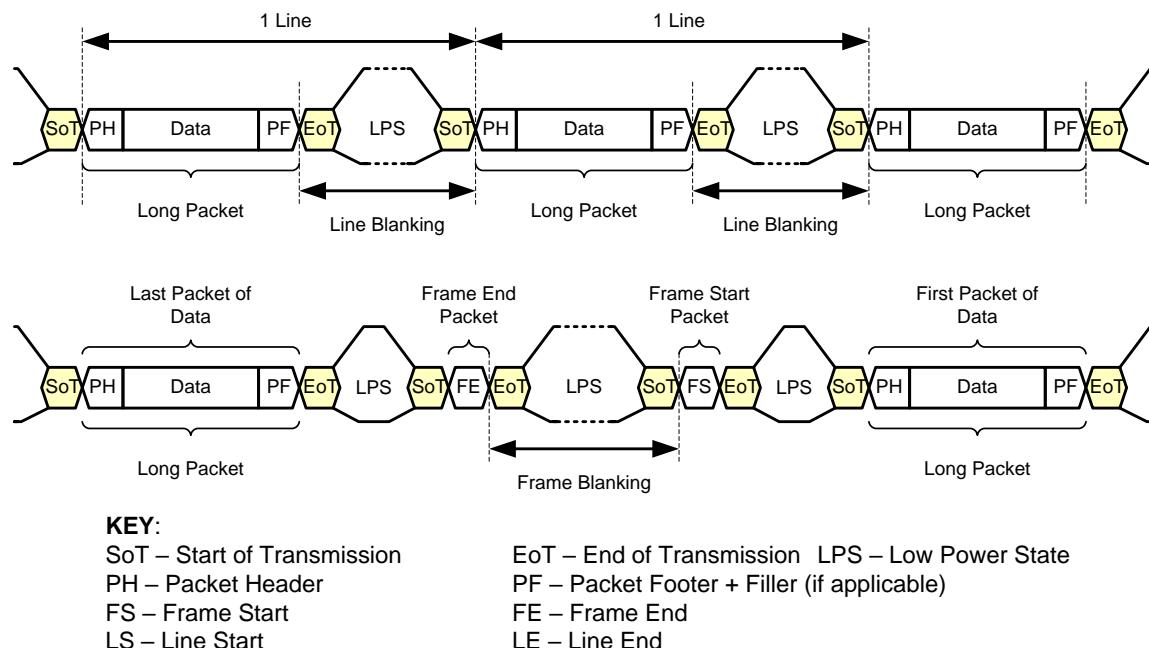


KEY:

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

EoT – End of Transmission LPS – Low Power State
PF – Packet Footer + Filler (if applicable)
FE – Frame End
LE – Line End

Figure 73 Single Packet Example

**Figure 74 Line and Frame Blanking Definitions**

The period between the end of the Packet Footer (or the Packet Filler, if present) of one long packet and the Packet Header of the next long packet is called the Line Blanking Period.

The period between the Frame End packet in frame N and the Frame Start packet in frame N+1 is called the Frame Blanking Period (**Figure 74**).

The Line Blanking Period is not fixed and may vary in length. The receiver should be able to cope with a near zero Line Blanking Period as defined by the minimum inter-packet spacing defined in [\[MIPI01\]](#) or [\[MIPI02\]](#), as appropriate. The transmitter defines the minimum time for the Frame Blanking Period. The Frame Blanking Period duration should be programmable in the transmitter.

Frame Start and Frame End packets shall be used.

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

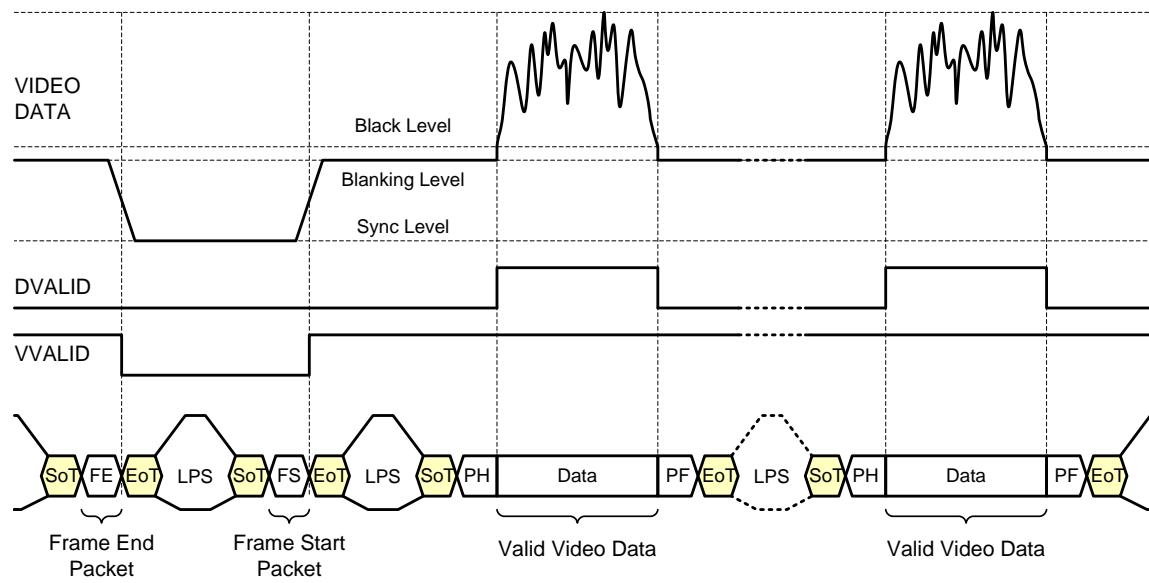
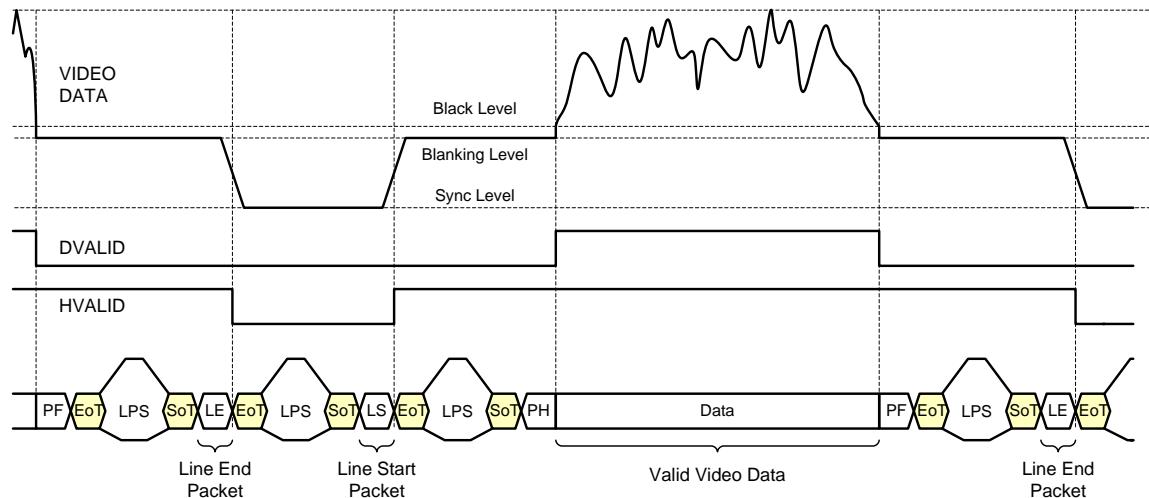
- The Frame Start packet to first data packet spacing should be as close as possible to the minimum packet spacing
- The last data packet to Frame End packet spacing should be as close as possible to the minimum packet spacing

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

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

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

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

**Figure 75 Vertical Sync Example****Figure 76 Horizontal Sync Example**

9.11 Latency Reduction and Transport Efficiency (LRTE)

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

LRTE has two parts, further detailed in this Section:

- Interpacket Latency Reduction (ILR)
- Enhanced Transport Efficiency

9.11.1 Interpacket Latency Reduction (ILR)

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

As shown in [Figure 77](#), interpacket latency reduction may be used to replace legacy EoT, LPS, and SoT packet delimiters with a more Efficient Packet Delimiter (EPD) signaling mechanism that avoids the need for HS-LPS-HS transitions. An EPD consists of PHY layer and/or protocol layer elements. The PHY-generated EPD element is referred to as “Packet Delimiter Quick” (PDQ). Protocol-generated EPD elements are called Spacers and may optionally precede PDQs.

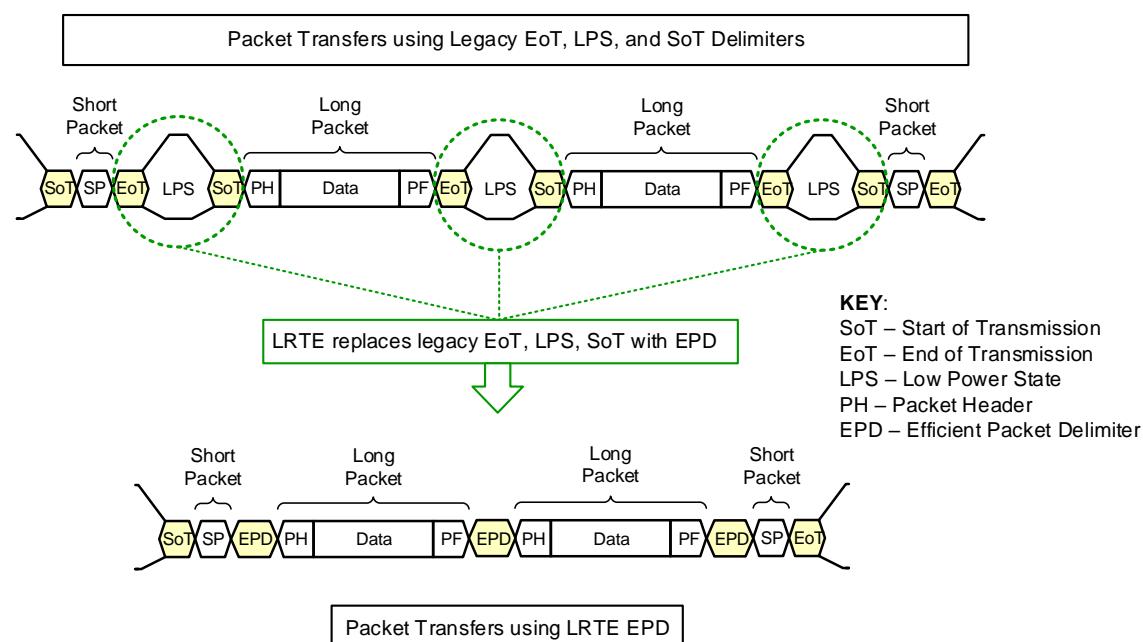


Figure 77 Interpacket Latency Reduction Using LRTE EPD

As shown in [Figure 77](#), LRTE requires an EPD to be inserted between adjacent CSI-2 packets during PHY high-speed signaling, but does not permit an EPD to be inserted after a CSI-2 packet just prior to PHY EoT. However, as described later in this section, it is possible under certain conditions to insert Spacers, but without PHY-generated PDQ signaling, after a CSI-2 packet just prior to PHY EoT.

9.11.1.1 EPD for C-PHY Physical Layer Option

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

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

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

1. TX_REG_CSI_EPD_EN_SSP (EPD Enable and Short Packet Spacer) Register

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

2. TX_REG_CSI_EPD_OP_SLP (Long Packet Spacer) Register

- The MS bit of this register is reserved for future use.
- If C-PHY EPD is enabled, the remaining 15 bits of this register (bits [14:0]) shall specify the minimum number (up to 32,767) of Spacer Word insertions per Lane following CSI-2 Long Packets.

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

- A register value of 15'd0 generates zero or more Spacers.
- A register value of 15'd5 generates at least 5 Spacers, resulting in a minimum duration of 5 x 7 UI.
- The maximum register value of 15'd32,767 generates at least 32,767 Spacers, resulting in a minimum duration of 32,767 x 7 UI.

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

Spacer Words without PDQ signaling may be inserted after CSI-2 packets just prior to C-PHY EoT only if C-PHY EPD is enabled and bit 7 of the TX_REG_CSI_EPD_MISC_OPTIONS register is set to 1; see **Table 16**. If this register bit is not implemented by an image sensor, its contents shall be treated as 0 by this specification. The minimum number of Spacer Word insertions just prior to C-PHY EoT is determined by the fifteen least significant bits of the TX_REG_CSI_EPD_EN_SSP and TX_REG_CSI_EPD_OP_SLP registers.

Note that C-PHY EPDs and Spacer Words without PDQ signaling are completely compatible with C-PHY ALP Mode high-speed burst transmissions as described in **[MIPI02]**.



KEY:

SoT – Start of Transmission

EoT – End of Transmission

EPD – Efficient Packet Delimiter contains optional Spacer

SYN – Sync Word

PH – Packet Header

PDQ (Sync Word). An EPD consisting of single Spacer followed by one PDQ shown for illustration

Figure 78 LRTE Efficient Packet Delimiter Example for CSI-2 Over C-PHY (2 Lanes)

9.11.1.2 EPD for D-PHY Physical Layer Option

1427 There are two EPD options for CSI-2 over the D-PHY physical layer option, as detailed in the following sub-
1428 sections.

1429 When EPD is enabled, CSI-2 over the D-PHY physical layer option shall align all Lanes corresponding to a
1430 Link using the minimum number of Filler byte(s) for both options. The value of the Filler byte shall be 0x00.
1431 The process of aligning Lanes within a Link through the use of Filler bytes is similar to native EOT alignment
1432 of CSI-2 over C-PHY.

9.11.1.2.1 D-PHY EPD Option 1

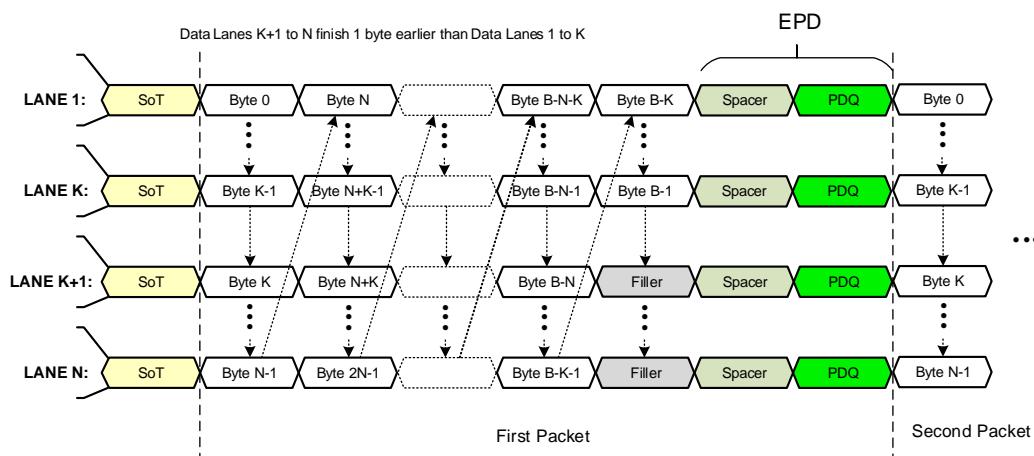
1433 D-PHY EPD Option 1 consists of PHY-generated and PHY-consumed D-PHY HS-Idle Packet Delimiter
1434 Quick (PDQ) signaling, optionally preceded by CSI-2 protocol-generated and protocol-consumed Filler bytes
1435 (as needed) plus zero or more Spacer bytes. The value of the Spacer byte for CSI-2 over D-PHY shall be
1436 0xFF, and when present, Spacer bytes shall be generated across all Lanes within a Link. The PDQ is generated
1437 and consumed by the transmitter and receiver physical layers, respectively, and as a result serves as a robust
1438 CSI-2 packet delimiter. D-PHY receivers can benefit from protocol-generated and protocol-consumed
1439 Spacer(s), because additional clock cycles might be needed to flush the payload content through the pipelines
1440 before the forwarded clock is disabled for PDQ signaling. Note that D-PHY HS-Idle is not supported by ALP
1441 mode in *[MIPI01]*.

1442 Under D-PHY Option 1, Filler bytes shall be inserted as needed following all short and long packets
1443 transmitted between D-PHY SoT and EoT sequences. For example, if a short packet is the last packet
1444 transmitted before EoT in an 8-Lane link, then a Filler byte is transmitted on each of Lanes 4 through 8 for
1445 that packet.

1446 Under D-PHY Option 1, an EPD may not be inserted after a CSI-2 packet just prior to D-PHY EoT, but
1447 Spacer Bytes without PDQ signaling may be inserted after such packets if bit 7 of the
1448 **TX_REG_CSI_EPDS_MISC_OPTIONS** register is 1. The minimum number of Spacer byte insertions just prior
1449 to D-PHY EoT is determined by the fifteen least significant bits of the **TX_REG_CSI_EPDS_EN_SSP** and
1450 **TX_REG_CSI_EPDS_OP_SLP** registers. See *Table 17*.

1451 The image sensor should use “TxHSIdleClkHS” at the PPI in order to generate the PDQ sequence by the
1452 D-PHY transmitter. Upon reception of the PDQ sequence by the D-PHY receiver, an application processor
1453 should use “RXSyncHS” at the PPI to notify the CSI-2 protocol layer. Additionally, “RxClkActiveHS” may
1454 also be used to provide an advance indication of the EPD.

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



KEY:

LPS – Low Power State

SoT – Start of Transmission

EoT – End of Transmission

PDQ – PHY generated and consumed Packet Delimiter Quick

EPD – Efficient Packet Delimiter

1455

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

9.11.1.2.2 D-PHY EPD Option 2

D-PHY EPD Option 2 is limited to the insertion of CSI-2 protocol-generated and CSI-2 protocol-consumed Filler bytes (as needed) plus zero or more Spacer bytes for use between multiple back-to-back packet transfers within the same D-PHY high-speed burst transfer (i.e., there is no use of PHY-generated and PHY-consumed PDQ). Option 2 is primarily intended for use with D-PHYs supporting legacy LP or LVLP mode that don't support Option 1; however, it may also be used with ALP Mode as described in [\[MIPI01\]](#). Depending on the use case (i.e., the sizes and number of CSI-2 packets being concatenated), the lack of D-PHY-generated and D-PHY-consumed packet delimiters could compromise CSI-2 link integrity. Option 2 is not intended to completely eliminate D-PHY LP, LVLP, or ALP Mode packet delimiters. It is also recommended that one or more Spacers be included following a Short Packet or a Long Packet when using D-PHY EPD Option 2.

D-PHY EPD Option 2 may also be applied to packets transmitted using C-PHY or D-PHY Escape Mode LPDT in connection with the Unified Serial Link (USL) feature described in [Section 9.12](#).

Under D-PHY Option 2, Filler bytes shall be inserted as needed following all short and long packets transmitted between D-PHY SoT and EoT sequences except for possibly the last packet before EoT. For the last packet, Filler bytes are required if Spacer bytes are inserted before EoT, and optional otherwise. For example, if a short packet is the last packet transmitted before EoT in an 8-Lane link, and no Spacer bytes are inserted before EoT, then Filler bytes are not required for Lanes 4 through 8 and EoT may start earlier on these Lanes.

Under D-PHY Option 2, an EPD (i.e., one or more Spacers) shall not be inserted immediately after the last CSI-2 short or long packet in any high-speed burst or LPDT payload, including after the EoTp short packet described in [Section 9.11.1.2.5](#).

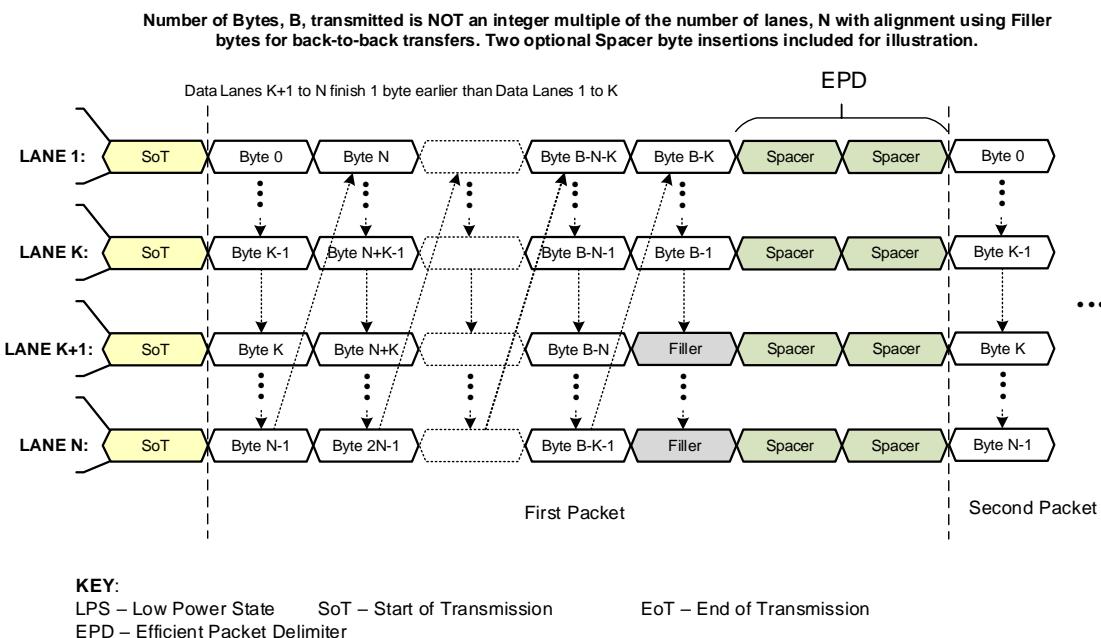


Figure 80 Example of LRTE EPD for CSI-2 Over D-PHY – Option 2

9.11.1.2.3 D-PHY EPD Specifications (for EPD Options 1 and 2)

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

1. TX_REG_CSI_EPD_EN_SSP (EPD Enable and Short Packet Spacer) Register

- The MS bit of this register shall be used to enable EPD insertion between two CSI-2 packets.
 - 1'b0: D-PHY legacy EoT, LPS, SoT Packet Delimiter
 - 1'b1: Enable D-PHY EPD (Efficient Packet Delimiter)

• **Variable-length Spacer insertions following Short Packets:**

If D-PHY EPD is enabled and either Option 1 is selected (i.e., bit 15 of register

TX_REG_CSI_EPD_OP_SLP is 0) or Option 2 is selected with bit 5 or bit 4 of register

TX_REG_CSI_EPD_MISC_OPTIONS in *Table 17* set to 1, then bits [14:0] of register

TX_REG_CSI_EPD_EN_SSP shall specify the minimum number (up to 32,767) of Spacers

insertions per Lane following CSI-2 Short Packets. The number of Spacers actually inserted per

Lane may vary from one Short Packet to another. For D-PHY EPD Option 2, both the contents of register **TX_REG_CSI_EPD_EN_SSP** and the actual number of Spacers inserted per Lane shall be

multiples of the value selected in bits [5:4] of register **TX_REG_CSI_EPD_MISC_OPTIONS**.

• **Fixed-length Spacer Insertions following Short Packets:**

If D-PHY EPD is enabled, and Option 2 is selected (i.e., bit 15 of register

TX_REG_CSI_EPD_OP_SLP is 1) with bit 5 and bit 4 of register

TX_REG_CSI_EPD_MISC_OPTIONS set to 0, then bits [14:0] of register

TX_REG_CSI_EPD_EN_SSP shall specify the exact number (up to 32,767) of Spacer insertions

per Lane following CSI-2 Short Packets. The number of Spacers inserted shall not vary from one

Short Packet to another.

2. TX_REG_CSI_EPD_OP_SLP (EPD Option and Long Packet Spacer) Register

- The MS bit of this register shall be used to select the D-PHY EPD option.

- 1'b0: D-PHY EPD Option 1

- 1'b1: D-PHY EPD Option 2

• **Variable-length Spacer insertions following Long Packets:**

If D-PHY EPD is enabled and either Option 1 is selected (i.e., bit 15 of register

TX_REG_CSI_EPD_OP_SLP is 0) or Option 2 is selected with bit 5 or bit 4 of register

TX_REG_CSI_EPD_MISC_OPTIONS set to 1, then bits [14:0] of register

TX_REG_CSI_EPD_OP_SLP shall specify the minimum number (up to 32,767) of Spacers

insertions per Lane following CSI-2 Long Packets. The number of Spacers actually inserted per

Lane may vary from one Long Packet to another. For D-PHY EPD Option 2, both the contents of register **TX_REG_CSI_EPD_EN_SLP** and the actual number of Spacers inserted per Lane shall be

multiples of the value selected in bits [5:4] of register **TX_REG_CSI_EPD_MISC_OPTIONS**.

• **Fixed-length Spacer insertions following Long Packets:**

If D-PHY EPD is enabled, and Option 2 is selected (i.e., bit 15 of register

TX_REG_CSI_EPD_OP_SLP is 1), with bit 5 and bit 4 of register

TX_REG_CSI_EPD_MISC_OPTIONS set to 0, then bits [14:0] of register

TX_REG_CSI_EPD_OP_SLP shall specify the exact number (up to 32,767) of Spacer insertions

per Lane following CSI-2 Long Packets. The number of Spacers inserted shall not vary from one

Long Packet to another.

1519 The following examples apply to the least significant fifteen bits of the two EPD registers:

1520 • **For variable-length Spacer insertions:**

- 1521 • A register value of 15'd0 generates zero or more Spacers.
1522 • A register value of 15'd5 generates at least 5 Spacers.
1523 • A register value of 15'd32,767 generates at least 32,767 Spacers.

1524 • **For fixed-length Spacer insertions:**

- 1525 • A register value of 15'd0 generates no Spacers.
1526 • A register value of 15'd5 generates exactly 5 Spacers.
1527 • A register value of 15'32,767 generates exactly 32,767 Spacers.

1528 The transmitter shall support at least one non-zero value of the Spacer insertion count field in each of the
1529 **TX_REG_CSI_EPD_EN_SSP** and **TX_REG_CSI_EPD_OP_SLP** registers. The duration of the PDQ
1530 sequence is directly proportional to the D-PHY Link rate, and is configured using the HS-Idle timing
1531 parameters defined in *[MIPI01]* for the D-PHY physical layer option.

1532 For D-PHY EPD, the **TX_REG_CSI_EPD_MISC_OPTIONS** register is required for image sensors
1533 (transmitters) supporting the insertion of Spacers-without-PDQ under Option 1 or the insertion of EoTp (see
1534 *Section 9.11.1.2.5*) or a variable number of Spacer bytes under Option 2. If this register is not implemented,
1535 then its value shall be treated as zero by this specification.

1536 Note that registers **TX_REG_CSI_EPD_EN_SSP**, **TX_REG_CSI_EPD_OP_SLP**, and
1537 **TX_REG_CSI_EPD_MISC_OPTIONS** are not intended to control image sensor LRTE when applied to USL
1538 packets transmitted using Escape Mode LPDT; see *Section 9.12*.

9.11.1.2.4 Robust Variable-Length Spacer Detection under D-PHY EPD Option 2 (Informative)

CSI-2 transmitters inserting a variable number of Spacer bytes per Lane between packets under D-PHY EPD Option 2 require CSI-2 receivers to examine, rather than simply count, potential Spacer bytes in order not to confuse them with packet header bytes. Since Spacer bytes are required to be distributed across all data Lanes beginning with Lane 1, CSI-2 receivers may detect them by scanning for bytes with the value 0xFF between packets only on Lane 1. However, truly reliable detection also requires these bytes to be error-free because a bit error in an intended Spacer byte (with value 0xFF) on Lane 1 can cause it to appear as a packet header Data Identifier byte. Conversely, a bit error in an intended packet header Data Identifier byte can cause it to appear as a Spacer byte.

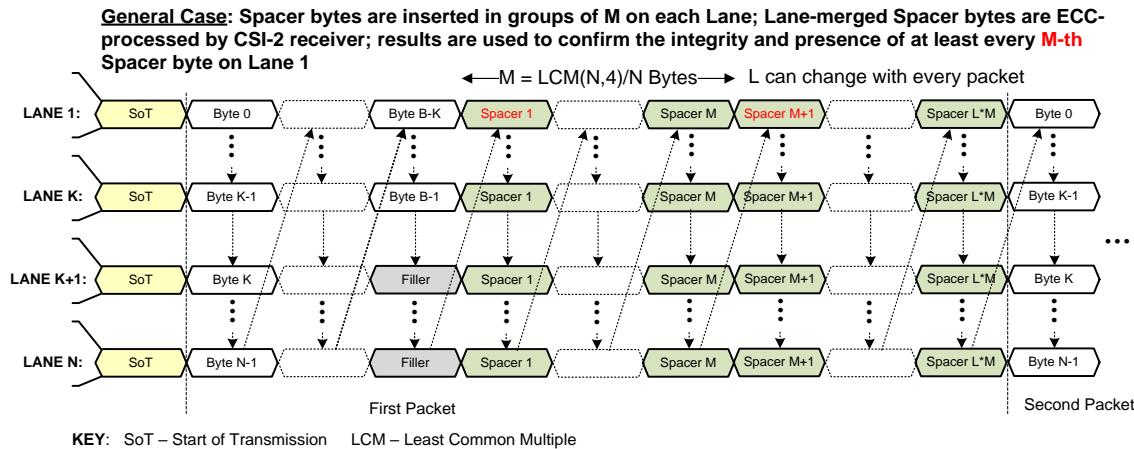
If Lane-merged groups of four sequential Spacer bytes are processed as potential 32-bit packet headers by a CSI-2 receiver, then the results of the ECC calculation can be used to detect and correct errors in the Spacer bytes just as it does in packet header bytes. This is possible because the value 0x3F in the least significant six bits of each transmitted group of four Spacer bytes also happens to be the 6-bit ECC of the value 0xFFFFFFFF in the most significant 26 bits.

Spacer byte insertion and detection schemes leveraging the latter principle may vary, depending upon the total number of data Lanes (N) in the link. Requiring the CSI-2 transmitter to insert Spacers in multiples of $M = \text{LCM}(N, 4) / N$ bytes per Lane, where LCM is the least common multiple function, always guarantees that the CSI-2 receiver will observe an integer multiple of four Spacer bytes between packets. As shown in **Table 15**, only values of M equal to 1, 2, and 4 are possible and programmable on the CSI-2 transmitter using bits 5:4 of the **TX_REG_CSI_EPD_MISC_OPTIONS** register shown in **Table 17**.

Table 15 Minimum Spacer Bytes per Lane for ECC Calculation

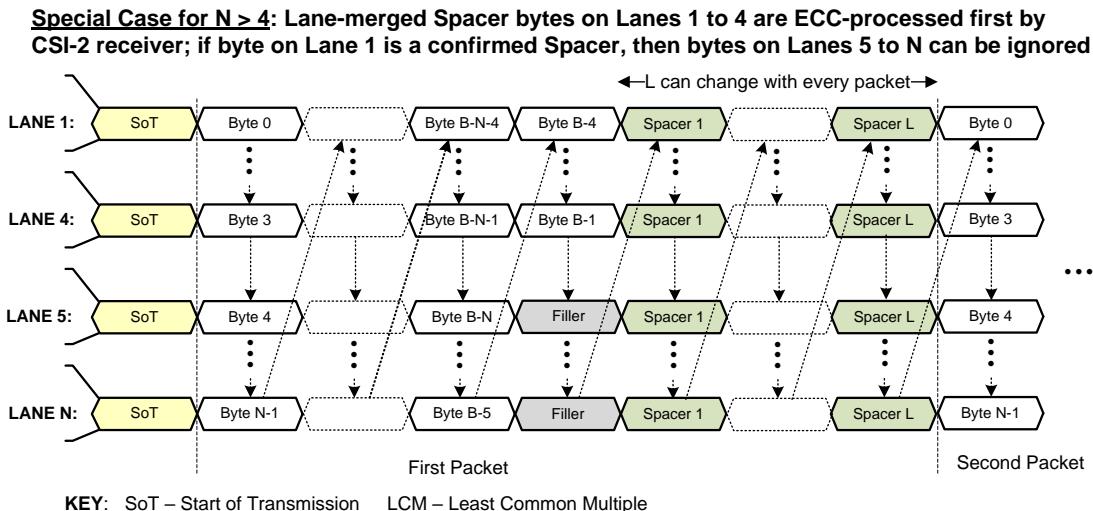
Number of Data Lanes (N)	Minimum Spacer Bytes per Lane $M = \text{LCM}(N, 4) / N$
1 + 4n	4
2 + 4n	2
3 + 4n	4
4 + 4n	1
Note: $n = 0, 1, 2, \dots$	

1559
1560
1561
1562 Note that the CSI-2 receiver only needs to check the data integrity of one out of every M potential Spacer bytes on Lane 1; i.e., once the first byte in a group of M bytes on Lane 1 has been confirmed as a Spacer byte, then the remaining M-1 bytes can be safely ignored by the CSI-2 receiver because it knows in advance that Spacer bytes are being inserted in groups of M. *See Figure 81.*



1563 **Figure 81 Enabling Robust Spacer Byte Detection: General Case**

1564 However, for $N > 4$, it is possible to program the CSI-2 transmitter to always insert an arbitrary number of
1565 Spacer bytes per Lane (i.e., to set $M = 1$) if the CSI-2 receiver examines bytes from the first four Lanes and,
1566 upon confirming a Spacer byte in Lane 1, ignores bytes from the remaining Lanes. See *Figure 82.*



1567 **Figure 82 Enabling Robust Spacer Byte Detection: Special Case**

9.11.1.2.5 End-of-Transmission Short Packet (EoTp)

When multiple CSI-2 packets are transmitted in a single D-PHY high-speed (HS) burst payload using D-PHY EPD Option 2, proper operation requires the host to be able to reliably detect the last CSI-2 packet in the burst under all circumstances. In many implementations, this requires the CSI-2 host protocol receiver to perform additional End-of-Transmission (EoT) processing on HS trailer and other bits passed to it from the D-PHY physical layer receiver.

Such EoT processing can be even more complex if HS trailer bits are followed by a small but potentially unknown number of bits with indeterminate values sampled, for instance, during the D-PHY HS to LP or LVLP voltage transition.

For D-PHY EPD Option 2, the End-of-Transmission short packet (EoTp) removes the need for the CSI-2 protocol receiver to perform EoT processing by unambiguously signaling the last CSI-2 packet in a D-PHY high-speed burst payload.

EoTp has the following short packet field definitions:

- DT shall be set to 0x04.
- All VC, VCX, and WC bits shall be set to zero.

Other rules pertaining to EoTp are as follows:

- EoTp generation or detection is mandatory for all devices conforming to this version of the CSI-2 specification that also support D-PHY EPD Option 2 for HS transmissions. EoTp shall not be used in conjunction with C-PHY EPD, D-PHY EPD Option 1, or packet transmissions using Escape Mode LPDT (see [Section 9.12](#)).
- Devices conforming to CSI-2 specification v2.1 or earlier do not support EoTp. In order to ensure interoperability with earlier devices, EoTp-supporting devices shall provide a means to enable or disable EoTp generation or detection; for image sensors, this capability shall be supplied via bit 6 of the **TX_REG_CSI_EPD_MISC_OPTIONS** register shown in [Table 17](#). This permits the EoTp feature to be effectively disabled by the system designer whenever a device on either side of the Link does not support EoTp.
- When the EoTp feature is enabled, exactly one EoTp shall be present in every high-speed burst payload as the last CSI-2 packet following all other short and/or long packets, including payloads with only one CSI-2 short or long packet in addition to the EoTp short packet itself.
- Spacers are not permitted after an EoTp because this packet is always immediately followed by D-PHY EoT, and never by another CSI-2 packet.
- The contents of EoTp VC, VCX, and WC fields shall be ignored by CSI-2 protocol receivers.

See [Figure 83](#) for EoTp usage examples.

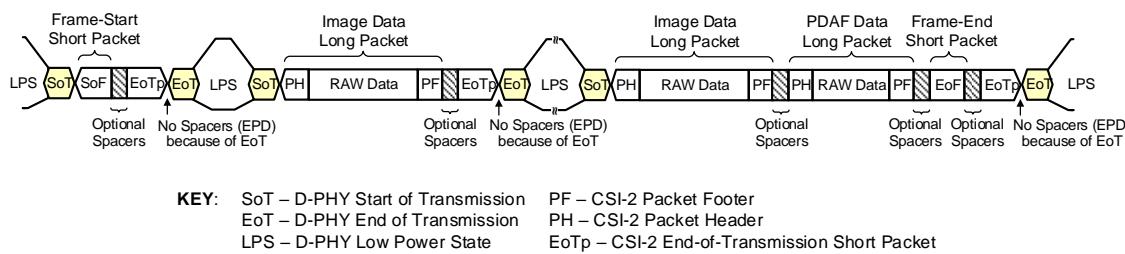
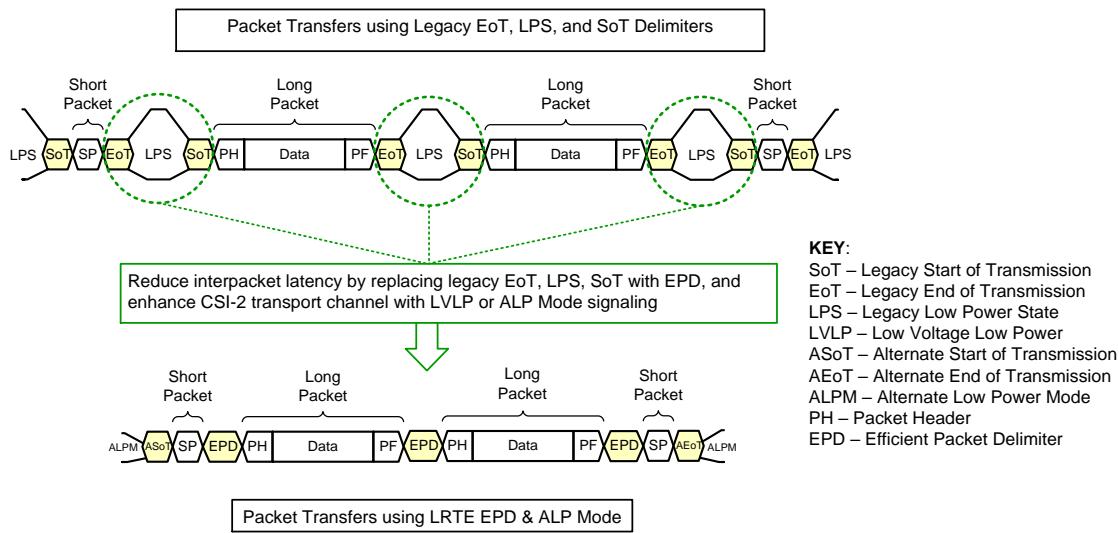


Figure 83 EoTp Usage Examples

9.11.2 Using ILR and Enhanced Transport Efficiency Together

1601
1602 EPD may be used together with LVLP or ALP Mode signaling in many imaging applications in order to benefit from CSI-2 ILR and enhanced channel transport.



1603

Figure 84 Using EPD with LVLP or ALP Mode Signaling

9.11.3 LRTE Register Tables

The CSI-2 over C-PHY Spacer Words and the CSI-2 over D-PHY Spacer Bytes shall be generated across all Lanes within a Link as specified in *Table 16* and *Table 17*.

Table 16 LRTE Transmitter Registers for CSI-2 Over C-PHY

Transmitter Register		Description
TX_REG_CSI_EPDP_EN_SSP [15:0]		Write-only. Required.
Bit [15]: Enable or disable Efficient Packet Delimiter using PHY-generated and PHY-consumed PDQ with optional minimum Spacer Insertion(s)	Value 1'b0: Disable Efficient Packet Delimiter Value 1'b1: Enable Efficient Packet Delimiter	CSI-2 over C-PHY EPD operation uses PHY-generated and PHY-consumed PDQ (7-UI Sync Word). Optional minimum Spacers may be Inserted for Short Packets and Long Packets. See <i>Figure 78</i> .
Bits [14:0]: EPD Short Packet Spacers	The minimum number of Spacer Words per Lane following a Short packet. Examples: Value 15'd0: Zero or more Spacer Words ... Value 15'd7: At least 7 Spacer Words ... Value 15'd32767: At least 32,767 Spacer Words	The Short Packet Spacers insertions are enabled by the C-PHY EPD (TX_REG_CSI_EPDP_EN_SSP[15]).
TX_REG_CSI_EPDP_OP_SLP [15:0]		Write-only. Required
Bit [15]: Reserved	Reserved	Reserved for future use
Bits [14:0]: EPD Long Packet Spacers	The minimum number of Spacer Words per Lane following a Long packet. Examples: Value 15'd0: Zero or more Spacer Words ... Value 15'd7: At least 7 Spacer Words ... Value 15'd32767: At least 32,767 Spacer Words	The Long Packet Spacers insertions are enabled by the C-PHY EPD (TX_REG_CSI_EPDP_EN_SSP[15]).
TX_REG_CSI_EPDP_MISC_OPTIONS [7:0]		Write-only. Optional.
Bit [7]: Enable insertion of Spacers-without-PDQ after CSI-2 packets just prior to C-PHY EoT.	Value 1'b0: Disable Spacers without-PDQ (default) Value 1'b1: Enable Spacers without-PDQ	Required for image sensors supporting C-PHY EPD with Spacers-without-PDQ.
Bits [6:0]: Reserved	-	-

Table 17 LRTE Transmitter Registers for CSI-2 Over D-PHY

Transmitter Register		Description
TX_REG_CSI_EDP_EN_SSP [15:0]		Write-only. Required
Bit [15]: Enable or disable EPD (Efficient Packet Delimiter) operation	Value 1'b0: Disable EPD Value 1'b1: Enable EPD	See Figure 79 . If EPD is enabled, the D-PHY EPD Options are determined by TX_REG_CSI_EDP_OP_SLP[15] .
Bits [14:0]: EPD Short Packet Spacers	For D-PHY EPD Option 1 -or- For Option 2 with Variable Spacers: Minimum number of Spacer Bytes per Lane following a Short packet. Examples: Value 15'd0: Zero or more Spacer Bytes ... Value 15'd7: At least 7 Spacer Bytes ... Value 15'd32767: At least 32,767 Spacer Bytes Otherwise, for D-PHY EPD Option 2: Fixed number of Spacer Bytes per Lane following a Short packet.	The Short Packet Spacers insertions are enabled by the D-PHY EPD (TX_REG_CSI_EDP_EN_SSP[15]). See Figure 79 and Figure 80 .
TX_REG_CSI_EDP_OP_SLP [15:0]		Write-only. Required.
Bit [15]: D-PHY EPD Option Select	Value 1'b0: D-PHY EPD Option 1 Value 1'b1: D-PHY EPD Option 2	D-PHY EPD Option 1: CSI-2 over D-PHY EPD operation using PHY-generated and PHY-consumed PDQ (using forwarded clock signaling) and optional Spacer Insertion(s). See Figure 79 . D-PHY EPD Option 2: CSI-2 over D-PHY EPD operation using optional Spacer Insertion(s). See Figure 80 .
Bits [14:0]: Long Packet Spacers	For D-PHY EPD Option 1 -or- For Option 2 with Variable Spacers: Minimum number of Spacer Bytes per Lane following a Long packet. Examples: Value 15'd0: Zero or more Spacer Bytes ... Value 15'd7: At least 7 Spacer Bytes ... Value 15'd32767: At least 32,767 Spacer Bytes Otherwise, for D-PHY EPD Option 2: Fixed number of Spacer Bytes per Lane following a Long packet.	The Long Packet Spacers insertions are enabled by the D-PHY EPD (TX_REG_CSI_EDP_EN_SSP[15]). See Figure 79 and Figure 80 .

Transmitter Register		Description
TX_REG_CSI_EPD_MISC_OPTIONS [7:0]		Write-only. Optional.
Bit [7]: For D-PHY EPD Option 1: Enable insertion of Spacers-without-PDQ after CSI-2 packets just prior to D-PHY EoT	Value 1'b0: Disable Spacers without-PDQ (default) Value 1'b1: Enable Spacers-without-PDQ	Required for image sensors supporting D-PHY EPD Option 1 with Spacers-without-PDQ.
Bit [6]: For D-PHY EPD Option 2: Enable EoTp	Value 1'b0: Disable EoTp (default) Value 1'b1: Enable EoTp	Required for image sensors supporting D-PHY EPD Option 2 with EoTp short packets.
Bit [5:4]: For D-PHY EPD Option 2: Enable variable-length Spacer insertions with a multiple of n Spacer bytes per Lane	Value 2'b00: Enable fixed-length Spacers (default) Value 2'b01: Enable variable-length Spacers with n = 1 Value 2'b10: Enable variable-length Spacers with n = 2 Value 2'b11: Enable variable-length Spacers with n = 4	Required for image sensors supporting D-PHY EPD Option 2 with variable-length Spacers.
Bits [3:0]: Reserved	—	—

9.12 Unified Serial Link (USL)

Unified Serial Link (USL, see *Figure 85*) is an optional CSI-2 feature that reduces the number of interface wires and helps to natively support longer reach.

USL builds upon the benefits provided by the LRTE features (*Section 9.11*). USL alleviates the need to use any additional I²C, I3C, and/or SPI interconnect for the Camera Command Interface (CCI), or GPIO wires for camera module control signaling. This is accomplished by using CSI-2 encapsulation with the Bus Turn Around (BTA) capabilities of the natively supported C-PHY 2.0 (or beyond) and D-PHY v2.5 (or beyond) transport layer options.

MIPI PHY bandwidths are many orders of magnitude greater than those provided by the I²C-compatible 2-wire bi-directional control bus. Moreover, there are natural blanking intervals (i.e., idle cycles) between transferring horizontal rows (i.e., horizontal blanking), and between transferring frames (i.e., vertical blanking), that can be used to update the image sensor shadow registers.

In this Section:

- The term **SNS** refers to an image sensor, or an image sensor module comprising a CMOS image sensor plus additional complementary devices (i.e., non-imaging devices).
- The term **APP** refers to an SoC application processor (AP) containing any combination of computer vision engine(s) and/or image processing unit(s), or to a host processor.

The reverse link throughput from an APP to a SNS does not require high bandwidth, which substantially relaxes timing constraints and margins for receiver implementations on the SNS transceiver. Mapping all CSI-2 transactions via MIPI C-PHY/D-PHY can reduce the number of wires, support long reach, help secure channel implementations, and reduce engineering development costs.

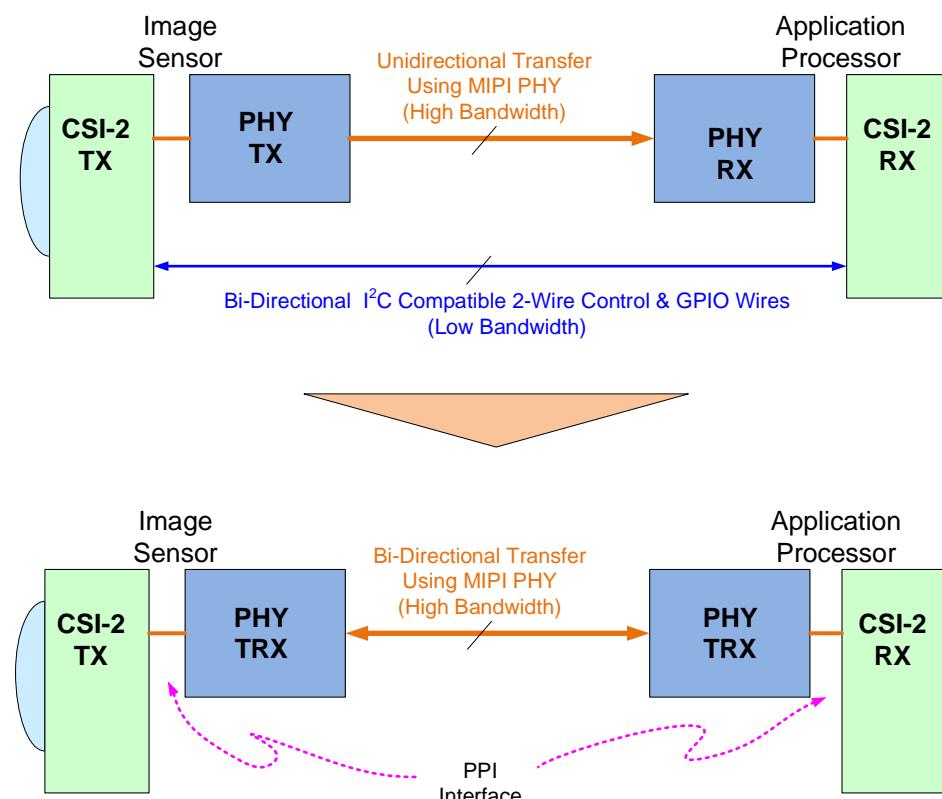


Figure 85 USL System Diagram

9.12.1 USL Technical Overview

Lane Usage: The vast majority of imaging applications using USL are expected to be mapped to a conduit with a single bi-directional lane (Lane 1). However, if a conduit requires multiple Lanes, then the first Lane (Lane 1) shall be bi-directional, and the remaining Lanes shall be uni-directional and configured as TX for image sensors. In multi-Lane conduits, all Lanes within a link are utilized for USL_FWD Mode operations, and only the first Lane (Lane 1) is utilized for USL_REV Mode operations. Transceiver functionality is always limited to one Lane (Lane 1). USL physical layer requirements are detailed in *Section 7.3*.

USL Commands and Transactions: The 0x38 PH Data Type (see *Table 10*) and the CSI-2 Long Packet Format shall be used for USL command operations and transactions. A “USL packet” is any long packet with Data Type 0x38.

Virtual Channels: In order to facilitate sensor aggregation on system platforms, the SNS shall support USL packets with Virtual Channel Identifiers ranging from 0 to 15 for D-PHY, and 0 to 31 for C-PHY. CSI-2 Frame Start and Frame End short packets shall not be transmitted for any Virtual Channel Identifiers used exclusively in USL packets. Virtual Channel Identifiers used in USL packets are permitted to be used in CSI-2 packets with Data Type codes other than 0x38 (i.e., non-USL packets). However, a pair of CSI-2 Frame Start and Frame End short packets is required to enclose all non-USL packets for each shared Virtual Channel Identifier within each image frame.

9.12.2 USL Command Payload Constructs

All USL transport operations shall use the existing CSI-2 Long Packet (LgP) constructs. The encapsulated USL payload fields (Size_Bytes, Target_Address, Sub_Address, Write_data, Read_data, USL_CTL, TSEQ) shall map LSB data to Bit0 (see *Figure 3*). All multi-byte USL payload fields shall be transmitted least significant byte first.

USL Commands are used to map register Read/Write requests from the APP to the SNS, and read completions from the SNS to the APP, using the CSI-2 LgP. The LgP Packet Header Data Type field shall be set to the value 0x38 when an APP or SNS generates USL Command register R/W requests or completions, respectively. For all C-PHY or D-PHY high-speed (HS) Mode transmissions, the encapsulated R/W requests are mapped into the LgP Packet Data Field Application Specific Payload of USL packets, per *Figure 52* for the D-PHY physical layer option, and per *Figure 53* for the C-PHY physical layer option. The Long Packet Padding and Filler insertions shall be preserved for the C/D-PHYS as defined in *Section 9.1*.

As described in *Section 7.3*, all USL implementations shall support PHY LP or LVLP Mode signaling, and should support ALP Mode signaling. USL systems are normally configured prior to power-up to support one of these signaling Modes under either D-PHY or C-PHY.

If USL is configured to use C-PHY or D-PHY LP/LVLP Mode, then all USL packets transmitted using forward or reverse direction Escape Mode LPDT have the same format as USL packets transmitted using D-PHY HS Mode; each byte is also transmitted least significant bit first. The APP shall be capable of receiving USL packets both as HS Mode transmissions distributed over all active Lanes, and as Escape Mode LPDT Transmissions driven only over Lane 1. Note that data scrambling (*Section 9.13*) does not apply to LPDT Transmissions.

During USL_REV Mode, the SNS shall support buffering for a minimum of 32 register read requests (single and contiguous) from an APP. During USL_REV Mode, the SNS shall support a minimum of 64 register write requests (single and contiguous) with a minimum of 1024 bits of write data from an APP.

If USL is configured to use C-PHY or D-PHY ALP Mode, then while in USL_REV Mode, the APP may concatenate multiple USL packets into a single HS burst transmission over Lane 1 using C-PHY EPD or D-PHY EPD Option 2, respectively, provided the applicable LRTE EPD features are supported (see *Section 9.11*). SNS HS Mode receiver LRTE capabilities may be understood from the SNS datasheet, or discovered in some other fashion. Similarly, while in USL_FWD Mode, the SNS may concatenate multiple USL and/or non-USL packets into a single HS burst transmission distributed over all active Lanes using one of the latter EPD alternatives.

If USL is configured to use C-PHY or D-PHY LP/LVLP Mode, then while in USL_FWD Mode, the SNS may concatenate multiple USL and/or non-USL packets into a single HS burst transmission distributed over all active Lanes using any supported C-PHY or D-PHY LRTE EPD feature (including D-PHY EPD Option 1). While in either USL_FWD or USL_REV Mode, LRTE D-PHY EPD Option 2 may be used to concatenate multiple USL packets transmitted using Escape Mode LPDT, but only with zero or more fixed length Spacers between packets and without the insertion of EoTp short packets. SNS Spacer generation for USL packets transmitted using LPDT may be controlled by the APP using the register in *Table 18*. SNS LP/LVLP Mode receiver LRTE capabilities may be understood from the SNS datasheet, or discovered in some other fashion.

The existing LgP 16-bit Checksum shall be used to preserve transport integrity for all USL transitions. USL Command transactions do not require the legacy CCI “S” (Start), “P” (Stop), and “A” (Acks), because the LgP transport handles these functions.

A 16-bit Size_Bytes field contains the number of sequential bytes of data to write or read. A Read_Data field contains the byte based read data.

A Write_Data field contains the byte based write data.

A 7-bit Target_Address field contains the device address where the commands are to be send or to be received. For example, CCI (I²C) Target Address e.g. 0x6C was used to communicate with an image sensor and

similarly with USL the same e.g. 0x6C would apply. This field is transmitted as the most significant 7 bits of a byte whose LSB is analogous to the CCI R/W bit.

A 16-bit Sub_Address field is used for pointing at a register inside the Target device. Product-specific imaging systems may use an 8-bit and/or 16-bit Sub_Address on an as-needed basis, since Sub_Address is device-specific and is known at platform level. Note that systems using a single USL link to communicate with multiple devices, e.g. via local I²C bus, may require using both 8-bit and 16-bit indexes, depending on the devices.

An 8-bit USL Control (USL_CTL) (**Table 19**) is used to ensure transport integrity with guaranteed delivery of commands from the APP to the SNS using ACK, and to improve transport efficiency with support for contiguous R/W operations often used for imaging and vision firmware uploads from APP to SNS.

A 16-bit Transaction Sequence (TSEQ) field contains a unique non-zero USL command identification number generated by the SNS and APP. The SNS and APP generate and clear the Transaction Sequence field upon successful reception of a USL Packet (as detailed in sections below).

Table 18 Image Sensor LPDT LRTE Control Register

Register Name	Type	RW	Comment
SNS_USL_LPDT_LRTE	8-bit unsigned integer	RW	<p>Bit 7 1: Enable image sensor LRTE for USL packets transmitted using Escape Mode LPDT</p> <p>Bits 6-0 If LRTE is enabled, specifies the fixed number of Spacers inserted between all USL packets (0 to 127)</p>

1705

Table 19 USL Transport Control (USL_CTL) Bit Description

USL_CTL[7:0]	Name	Description	Initiator
Bits [1:0]	ACK_NAK_INT Generation	Values: 2'b01: ACK generation. SNS transmits the TSEQ of the last successful reception an USL command. 2'b10: NAK generation. SNS transmits the TSEQ of the R/W command resulting in an illegal or invalid operation. 2'b00: Neither ACK nor NAK generation (i.e., Read completions) 2'b11: In-band interrupt	SNS
Bit [2]	Force Command	Values: 1'b0: The requested command from APP shall not be executed by the SNS if a prior command request was NAK'ed during USL_REV_Mode. 1'b1: Force command operation even if a prior command was NAK'ed during USL_REV_mode.	APP
Bit [3]	Sequential R/W Enable	Values: 1'b1: USL command includes sequential R/W request or response. The requested size (in bytes) shall be determined from Size_Bytes[15:0] field. 1'b0: USL command includes single R/W request or response.	APP
Bit [4]	Initiate BTA	Values: 1'b1: Command bit used to turn around the bus. Generated by the Initiator. 1'b0: NOP	APP during USL_REV Mode, SNS during USL_FWD Mode
Bits [7:5]	Reserved for future use	Reserved for future use	—

1706

Note:

1707

The USL_CTL[7:5] bits are reserved for future expansion.

9.12.3 USL Operation Procedures

For simplification, the following examples show:

- A 16-bit Sub_Address [15:0]
- A 16-bit Transaction Sequence (TSEQ) containing a unique non-zero USL command identification number generated by the APP

9.12.3.1 APP Initiated USL Transactions

This example procedure illustrates the APP initiating four valid USL Command transactions (USL_REV Mode).

1. ACK Generation

TSEQ is used by the system to ensure guaranteed delivery of USL commands. An APP shall include a 16-bit register to store the last successful TSEQ received from the SNS during USL_FWD Mode. Upon entering USL_REV Mode, the APP shall generate ACK twice immediately using the following format:

- LgP Packet Header Data ID = 0x38
- LgP Packet Data format for ACK generation:
 {USL_CTL[7:0], TSEQ[15:0]}

2. Register Write Request with Write Data

- LgP Packet Header Data ID = 0x38
- LgP Packet Data format for all writes:
 {USL_CTL[7:0], TSEQ[15:0], Size_Bytes[15:0],
 Target_Address[7:1], [W=0],
 Sub_Address[15:0],
 Write_Data [16'd Size_Bytes-1:0]}

3. Register Read Request

- LgP Packet Header Data ID = 0x38
- LgP Packet Data format for all read requests:
 {USL_CTL[7:0], TSEQ[15:0], Size_Bytes[15:0],
 Target_Address[7:1], [R=1],
 Sub_Address[15:0]}

4. Initiate BTA

Upon completing the R/W register commands, the APP shall generate the Initiate BTA packet twice to switch the link from USL_REV to USL_FWD mode.

The Initiate BTA bit in the USL_CTL shall be set to 1'b1.

- LgP Packet Header Data ID = 0x38
- LgP Packet Data format for initializing BTA, and switch from USL_REV to USL_FWD:
 {USL_CTL[7:0], TSEQ[15:0]}

9.12.3.2 SNS Initiated USL Transactions

This example procedure illustrates the SNS initiating five valid USL Command transactions (USL_FWD Mode).

1. NAK Generation

If an Image Sensor (SNS) receives an invalid or an illegal USL Command request from the Application Processor (APP), then the SNS shall generate a Negative Acknowledgement (NAK). By default, the SNS shall not execute any following R/W USL command requests from the APP after a NAK during an USL_REV_Mode, unless the USL_CTL Force Command bit is enabled by the APP. The SNS shall generate negative acknowledgement (NAK) for the first illegal or failed R/W request from an APP. The SNS may optionally generate additional NAKs resulting from “Force Command” requests. The NAK shall be transmitted twice to the APP immediately upon switching to USL_FWD_Mode using the following format:

- LgP Packet Header Data ID = 0x38
- LgP Packet Data format for NAK generation:
{USL_CTL[7:0], TSEQ[15:0]}

2. ACK Generation

TSEQ is used by the product imaging system to ensure guaranteed delivery of R/W commands from APP to SNS. An Image Sensor shall include a 16-bit register to store the last successful TSEQ received from the APP during USL_REV Mode. Upon entering USL_FWD Mode, the SNS shall generate ACK twice immediately following any NAK generation(s) using the following format:

- LgP Packet Header Data ID = 0x38
- LgP Packet Data format for ACK generation:
{USL_CTL[7:0], TSEQ[15:0]}

3. Register Read Completion

- LgP Packet Header Data ID = 0x38
- LgP Packet Data format for all read completions:
{USL_CTL[7:0], TSEQ[15:0], Size_Bytes[15:0],
Read_Data [16'd Size_Bytes-1:0]}

4. Interrupt Generation

Upon completing ACK/NAK generations, the SNS may generate in-band interrupt using USL_CTL[1:0] in the USL_FWD mode. It is strongly recommended for the SNS to prioritize and generate the interrupt notification at the earliest opportunity.

The following format shall be used to also include optional information pertaining to the interrupt:

- LgP Packet Header Data ID = 0x38
- LgP Packet Data format for In-Band Interrupt generation:
{USL_CTL[7:0], Target_Address[15:0], Interrupt_Information[15:0]}

5. Initiate BTA

The SNS shall generate the Initiate BTA packet twice prior to switching the link from USL_FWD to USL_REV mode. The Initiate BTA bit in the USL_CTL shall be set to 1'b1. The TSEQ shall be the 16-bit value from REG_USL_ACK_TSEQ.

- LgP Packet Header Data ID = 0x38
- LgP Packet Data format for initializing BTA, and switch from USL_REV to USL_FWD:
{USL_CTL[7:0], TSEQ[15:0]}

9.12.4 Monitoring USL Command Transport Integrity

This section presents a stepwise procedure for monitoring the integrity of the USL Command Transport, using TSEQ and the two SNS registers defined in *Table 20*.

1. [System Reset] [Power Cycle]

- The SNS shall reset registers **REG_USL_ACK_TSEQ[15:0]** and **REG_USL_NAK_TSEQ[15:0]** to the value 16'd0. The APP shall reset internal register **APP_REG_USL_ACK_TSEQ[15:0]**.

2. [USL_REV Mode]

- Upon entering USL_FWD Mode, APP shall generate ACK twice using the 16-bit TSEQ from internal register **APP_REG_USL_ACK_TSEQ[15:0]**.
- The APP shall generate a 16-bit, non-zero TSEQ value that increments with each USL Command R/W request it sends to the SNS.
- The SNS shall store the TSEQ of the last successful USL command into register **REG_USL_TSEQ_ACK[15:0]**.
- The SNS shall store the TSEQ of the first illegal USL command into register **REG_USL_TSEQ_NAK[15:0]**.

3. [USL_REV Mode Exit]

- The APP shall reset internal register **APP_REG_USL_ACK_TSEQ[15:0]** to the value 16'd0.

4. [USL_FWD Mode]

- Upon entering USL_FWD Mode, if any illegal operation was encountered in the prior USL_REV Mode, the SNS shall generate NAK twice using the 16-bit TSEQ from register **REG_USL_TSEQ_NAK[15:0]**.
- Next, the SNS shall generate ACK twice using the 16-bit TSEQ from register **REG_USL_TSEQ_ACK[15:0]**.
- An ACK with TSEQ value of 16'd0 shall be generated by the SNS if no USL commands were successfully received from the APP during USL_REV Mode.

Note:

SNS shall reset the internal state(s) used to block execution of subsequent command operations after a NAK.

5. [USL_FWD Mode Exit]

The SNS shall reset registers **REG_USL_ACK_TSEQ[15:0]** and **REG_USL_NAK_TSEQ[15:0]** to the value 16'd0.

Table 20 USL Transport Integrity ACK and NAK Registers with TSEQ

SNS USL Transport Integrity Registers	Description
REG_USL_ACK_TSEQ[15:0]	Required. Write / Read. See Above.
REG_USL_NAK_TSEQ[15:0]	Required. Write / Read. See Above.

9.12.5 USL Powerup / Reset, SNS Configuration, and Mode Switching

This section details the various stages of USL operations. Note that the current TX shall always initiate the BTA when switching USL modes.

9.12.5.1 USL Link Modes

The USL link is in **USL Reverse Mode** (USL_REV) when:

- SNS is configured as RX
- APP is configured as TX
- The channel is established to transfer payloads from APP to SNS

The USL link is in **USL Forward Mode** (USL_FWD) when:

- SNS is configured as TX
- APP is configured as RX
- The channel is established to transfer payloads from SNS to APP

9.12.5.2 USL Power-up / Reset

The link comes up in USL_FWD Mode after power-up or reset:

- The SNS shall initially configure each PHY Lane (including clock Lane, if present) as a transmitter (TX).
- The APP shall initially configure each PHY Lane (including clock Lane, if present) as a receiver (RX).
- The SNS and APP shall then initialize their Lane 1 PHYs following the appropriate PHY-defined procedure; all other Lanes are unidirectional and should remain uninitialized until needed.
- If USL is configured to use D-PHY ALP Mode, then the clock Lane shall also be initialized and started following its initialization in order to facilitate ALP Mode fast BTA and high-speed bidirectional data transfers over data Lane 1; see *Section 9.12.5.6* for additional guidance.
- If USL is configured to use D-PHY LP/LVLP Mode, then initialization and start-up of the clock Lane may be delayed until the SNS is first required to transmit packets to the APP using HS Mode. Note that in this case, all USL packet transmissions from the APP to the SNS over data Lane 1 use Escape Mode LPDT and don't require the clock Lane to be running.
- Once the SNS has completed internal power-up / reset calibration, the SNS shall initiate BTA on Lane 1 using the steps outlined in *Section 9.12.3.2*. The SNS may optionally send the contents of **TX_USL_SNS_BTA_ACK_TIMEOUT[15:0]** using the read completion format from *Section 9.12.3.2* prior to initiating BTA.
- Upon completion of the BTA, the link is configured as USL_REV Mode to enable the APP to configure the SNS using Lane 1.
- During SNS configuration, the APP may select the total number of active Lanes in order to enable the correct number of unidirectional Lanes to be initialized and put into service when image streaming is started.

9.12.5.3 USL SNS Configuration

The APP may configure the SNS when the link is in USL_REV Mode using steps outlined in **Section 9.12.3.1**.

- Once the APP has completed one or more SNS configuration operations, the APP shall initiate BTA using steps outlined in **Section 9.12.3.1**.
- Upon completion of the BTA, the link is configured as USL_FWD Mode.

9.12.5.4 USL Mode Switching SNS Configuration

Upon entering USL_FWD Mode, the SNS generates the USL NAK (if needed) followed by the USL ACK using the steps outlined in **Section 9.12.3.1**. The SNS generates the USL read completions as the content becomes available, and may be interleaved with non-USL payloads during USL_FWD Mode.

The SNS shall support the two 16-bit USL BTA Switch registers defined in **Table 21** and **Table 22**. The APP shall configure these registers during USL_REV Mode.

The USL BTA switch may be initiated during vertical blanking for traditional photography and video applications, or after predefined LgPs (intra-frame) for more advanced vision applications. CSI-2 Imaging and Vision systems will require fast BTA durations mapped to the PHYs. **Figure 86** illustrates the USL_REV and USL_FWD State Diagram for both the APP and the SNS.

When a system is powered up, the SNS is configured as an RX (receiver) and the APP is configured as a TX (transmitter). Five USL modes are allowed for imaging applications, along with the transition arcs as defined in **Figure 86**.

A USL SNS shall support the 16-bit Operational Register shown in **Table 25**.

A USL SNS shall support one or more 16-bit GPIO Registers, per **Table 26**.

Table 21 USL BTA Switch Registers

SNS USL BTA Registers		Description
TX_USL_REV_ENTRY [15:0]		Required. Write / Read. Enable USL_REV Mode Entry after the specified number of non-USL LgPs, non-USL Frames, or PPI Word/Byte clocks.
Bit [15:10]: Select Mode Switch Triggers	Bit[15]: Clock Counter	Increment Clock Counter using PPI Word/Byte clock in USL_FWD Mode. Switch to USL_REV mode when Clock Counter expires. Reinitialize the Clock Counter when exiting USL_FWD mode.
	Bit[14]: LgP Counter	Increment LgP Counter using non-USL LgPs in USL_FWD Mode. Switch to USL_REV mode when LgP Counter expires. Reinitialize the LgP Counter when exiting USL_FWD mode.
	Bit[13]: Frame Counter	Increment Frame Counter using non-USL FEs in USL_FWD Mode. Switch to USL_REV mode when Frame Counter expires. Reinitialize the Frame Counter and LgP Counter when exiting the USL_FWD mode.
	Bit[12]: Chronological Timer	The Chronological (duration in μ s) Timer is initiated with the first non-USL transmission in USL_FWD mode. Switch to USL_REV mode once Chronological Duration timer expires and any inflight packet has completed transmission. Reinitialize the timer when exiting USL_FWD mode.
	Bit[11]: Configure SNS	Enable SNS to switch to REV_MODE immediately after the USL transmissions are completed (i.e., read completion, ACK, NAK). Non-USL (pixel data) transmissions are not allowed during the USL_FWD Mode.
	Bit[10]: Smart SNS	Enable SNS to switch to REV_MODE autonomously based on smart features. Smart SNS capability is optional for the SNS.
	Bit[9-0]: Reserved for future use	
TX_USL_Clock Counter [15:0]		Required. Write / Read. Counter used to trigger SNS switch from USL_FWD Mode to USL_REV Mode.
TX_USL_LGP Counter [15:0]		Required. Write / Read. Counter used to trigger SNS switch from USL_FWD Mode to USL_REV Mode.
TX_USL_Frame Counter [15:0]		Required. Write / Read. Counter used to trigger SNS switch from USL_FWD Mode to USL_REV Mode.
TX_USL_Chronological Timer [15:0]		Required. Write / Read. Counter used to trigger SNS switch from USL_FWD Mode to USL_REV Mode.

1869

Table 22 Register TX_USL_REV_FWD_ENTRY

SNS USL BTA Register		Description
TX_USL_FWD_ENTRY [15:0]		<p>Required. Write / Read.</p> <p>Enable USL_FWD Mode Entry after the specified number of PPI Word/Byte clocks.</p>
Bit [15]: FWD_Switch_En	Value 1'b0: Disable Value 1'b1: Initiate switch to USL_FWD Mode once the REV_MODE PPI Word / Byte clock counts match FWD_Counter [14:0]	<p>USL_REV Mode: Upon completing the R/W requests, the APP shall initiate switching to USL_FWD mode by writing 1'b1 to FWD_Switch_En</p> <p>USL_FWD Mode or SNS Reset: The SNS shall reset FWD_Switch_En bit by writing 1'b0.</p>
Bits [14:0]: FWD_Counter	Value 15'd0: Initiate switch to USL_FWD mode immediately after APP writes 1'b1 to FWD_Switch_En. ... Value 15'd7: Increment FWD_Counter using USL_REV Mode PPI Word / Byte clock when APP writes 1'b1 to USL_FWD Switch_En. Initiate switch to USL_FWD mode when FWD_Counter[14:0] = 15'd7 Value 15'd32767: Increment FWD_Counter using USL_REV Mode PPI Word / Byte clock when APP writes 1'b1 to USL_FWD Switch_En. Initiate switch to USL_FWD mode when FWD_Counter[14:0] = 15'd32767 .	<p>USL_REV Mode: Increment FWD_Counter using the USL_REV PPI Byte / Word clock when APP writes 1'b1 to USL_FWD Switch_En.</p> <p>USL_FWD Mode or SNS Reset: The SNS may optionally reset the FWD_Counter by writing 15'd0.</p>

1870

9.12.5.5 ALP Fast BTA Timeout Support

Since the target device PHY does not provide an acknowledgement electrical signaling upon receiving an “Initiate BTA” USL command request from an initiator, the following SNS timeout registers are utilized by the APP to alleviate potential deadlocks.

Table 23 Register TX_USL_SNS_BTA_ACK_TIMEOUT[15:0]

SNS USL BTA Register	Description
TX_USL_SNS_BTA_ACK_TIMEOUT[15:0]	<p>Required. Read Only.</p> <p>Maximum time (in ns) required by SNS to send ACK in the USL_FWD Mode upon SNS receiving “Initiate BTA” USL command from APP in USL_REV Mode.</p> <p>Value of 16'd0 implies the timeout is disabled.</p> <p>Value of 16'd500 implies the SNS shall send ACK in the USL_FWD Mode within 500 ns upon reception of “Initiate BTA” USL command from APP in USL_REV Mode.</p>

Table 24 Register TX_USL_APP_BTA_ACK_TIMEOUT[15:0]

SNS USL BTA Register	Description
TX_USL_APP_BTA_ACK_TIMEOUT[15:0]	<p>Required. Write Only.</p> <p>Maximum time (in ns) required by APP to send ACK in the USL_REV Mode upon APP receiving “Initiate BTA” USL command from SNS in USL_FWD Mode.</p> <p>Value of 16'd0 implies the timeout is disabled.</p> <p>Value of 16'd500 implies the APP shall send ACK in the USL_REV Mode within 500 ns upon reception of “Initiate BTA” USL command from SNS in USL_FWD Mode.</p>

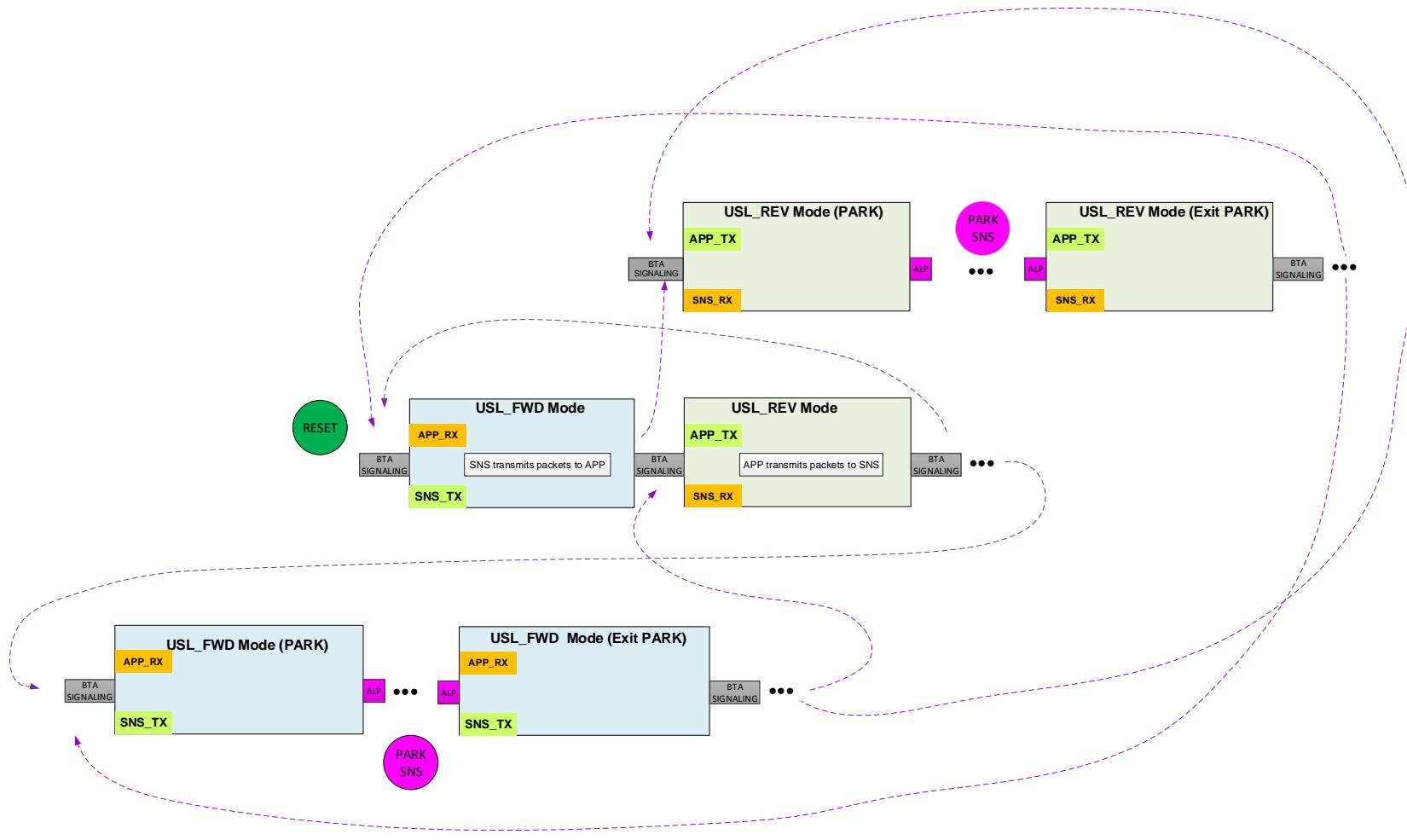


Figure 86 USL Modes Link Transitions

1877

Table 25 USL Operation Registers

SNS USL Operational Register		Description
TX_USL_Operation [15:0]		Required. Write / Read. General purpose register mapped SNS operations
Bit [0]: SNS Reset	Values: 1'b0: NOP 1'b1: Reset SNS	In-band register mapped SNS reset
Bit [15:1]: Reserved	Reserved	Reserved for future use

1878

Table 26 USL GPIO Registers

Bit [15:0]: SNS USL GPIO Register0		Description
TX_USL_GPIO [15:0]		Required. Write / Read. General purpose register mapped GPIO operations
Bit [0]: GPIO_0 Configuration	Values: 1'b0: Low 1'b1: High	In-band register mapped GPIO_0 configuration
...		
Bit [7]: GPIO_7 Configuration	Values: 1'b0: Low 1'b1: High	In-band register mapped GPIO_7 configuration
...		
Bit [15]: GPIO_15 Configuration	Values: 1'b0: Low 1'b1: High	In-band register mapped GPIO_15 configuration

9.12.5.6 USL Clock Lane Management Under D-PHY ALP Mode (Informative)

When USL is configured to use D-PHY ALP Mode as described in [MIPI01], a running high-speed clock Lane is required for any active data communications between SNS and APP. Such communications include the usual forward-direction SNS pixel streaming, as well as any bidirectional transactions on data Lane 1 related to USL packet transmissions (with DT code 0x38), or any low-level PHY fast BTA or ULPS entry commands initiated by either the SNS or APP. When USL is configured to use D-PHY LP or LVLP Mode, the clock Lane is only required to be running during SNS pixel streaming.

9.12.5.6.1 System Power-Up and/or Reset (Informative)

Following SNS ALP Mode TX and RX PHY initialization, the SNS will start up the clock Lane at some initial, implementation-dependent frequency prior to switching to USL_REV mode. For example, it may be advantageous to set this frequency equal to the SNS external reference clock frequency (or a submultiple thereof) since this avoids the need to start and lock a PLL, potentially saving hundreds of microseconds of start-up latency. Care must be taken to ensure that the selected clock Lane frequency is at least twice the minimum bit rate required by the D-PHY specification, since the D-PHY high-speed reverse-direction data transmissions used by USL occur at one-fourth the bit rate of forward-direction data transmissions. For example, a 2 MHz clock Lane frequency corresponds to a forward-direction bit rate of 4 Mbps and a reverse-direction bit rate of 1 Mbps.

Once the clock Lane is running, the SNS can use the USL protocol to switch to USL_REV mode in order to enable the APP to configure SNS CCI registers as needed, including the PLL control registers used for setting the clock Lane frequency required for image streaming. The last action taken by the APP will be to request the start of image streaming by writing to the appropriate CCI control register and then switching to USL_FWD mode; the SNS, in response, will stop the clock Lane, lock all PLLs to their configured frequencies, restart the clock Lane, and only then actually start image streaming.

9.12.5.6.2 Dynamic Clock Control (Informative)

As with legacy CSI-2 non-continuous clock mode, USL permits the clock Lane to be stopped during periods in which the system application either doesn't support or doesn't immediately anticipate data communications between SNS and APP.

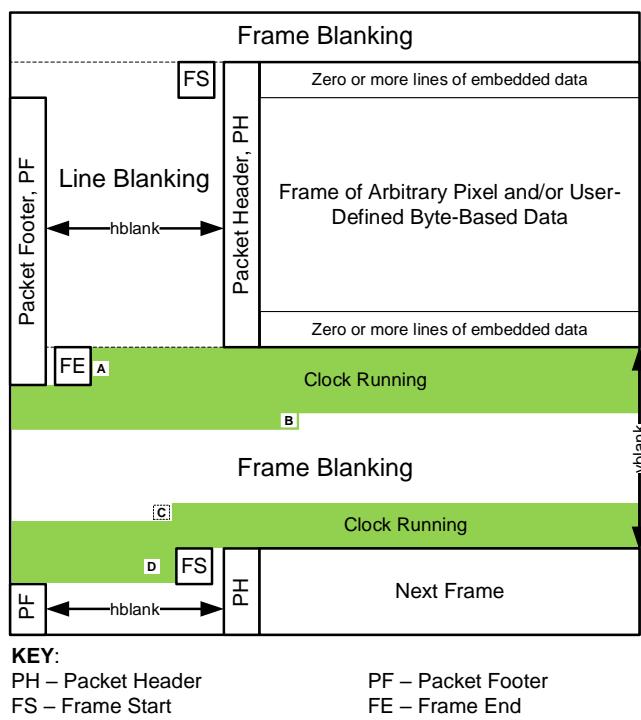
Examples of such periods include:

- One or more of the “horizontal line blanking” (hblank) periods between image lines during image streaming (assuming such periods are sufficiently long enough to include the timing overhead entailed in stopping and then restarting the clock).
- All or part of the “vertical frame blanking” (vblank) period between image frames during image streaming.
- “Hard standby” periods during which image sensor power consumption is at a minimum and no CCI register accesses are possible. Hard standby entry and exit is typically controlled using a separate control signal (e.g., the XSHUTDOWN signal defined in [\[MIPI04\]](#)).
- All or part of “soft standby” periods during which image streaming is temporarily halted to enable the APP to reconfigure CCI registers concerning fundamental SNS operating characteristics, such as output image resolution, pixel depth, frame rate, bit rate, active Lane count, etc.

Stopping and restarting the clock Lane may be performed by the SNS either automatically following conventions understood in advance by the APP, or in response to the APP triggering the SNS in an ad hoc manner. For example, when the clock Lane is running during USL_REV mode, the APP may command the SNS to stop the clock by writing to a special CCI control register on the SNS. Conversely, when the clock Lane is stopped during USL_REV mode (meaning no CCI register accesses are possible), the APP may request the SNS to restart the clock by transmitting a short D-PHY ALP Mode PHY-to-PHY Wake pulse to the SNS (as described in [\[MIPI01\]](#)).

Note that changing the clock Lane frequency requires the SNS to stop the clock Lane, internally adjust the frequency (which may involve relocking a PLL), and then restart the clock Lane in accordance with the D-PHY specification. Needless to say, the latter actions can be collectively time consuming and are best avoided when switching from USL_FWD mode to USL_REV mode and then back again during relatively short time periods such as hblank. In other words, if the APP needs to access SNS CCI registers during hblank, then the ideal situation is one in which both the SNS and APP can support reverse-direction transmissions at one-fourth of the *full* bit rate used for streaming pixel data packets, thereby avoiding the need to change the clock Lane frequency twice during hblank.

1930 **Figure 87** illustrates examples of USL ALP Mode clock Lane management during the image streaming vblank period. Beginning at time **A** in the figure, the SNS automatically keeps the clock Lane running after
 1931 transmitting the CSI-2 Frame End short packet and uses the USL protocol to enter USL_REV mode in order
 1932 to enable the APP to start accessing SNS CCI registers. At this point, the APP may also wish to start an
 1933 internal timer to warn it when the vblank period is about to end. At time **B**, when the APP is at least
 1934 temporarily finished with CCI register accesses, the last access it performs is a write to the
 1935 **TX_USL_ALP_CTRL** register (**Table 27**), for example, which commands the SNS to turn-off the clock Lane
 1936 but also to expect a request to restart it at a later time. USL_REV mode is then maintained (with all D-PHY
 1937 clock and data Lanes in the Stop state) until time **C** when the APP requests the clock Lane to be restarted by
 1938 transmitting a short ALP Mode PHY-to-PHY Wake pulse to the SNS. In response, the SNS restarts the clock
 1939 Lane and keeps it running while the APP performs more CCI accesses as needed. When finished at time **D**,
 1940 the APP finally switches back to USL_FWD mode prior to the SNS having to transmit the Frame Start short
 1941 packet at the beginning of the next image frame.
 1942



1943 **Figure 87 Examples of USL ALP Mode Clock Lane Management During Sensor Vblank**

1944

Table 27 USL Clock Lane Control Register

Register Name	Type	RW	Comment
TX_USL_ALP_CTRL	16-bit unsigned integer	RW	<p>Bit 0</p> <p>1: Shall trigger image sensor to pause the D-PHY clock Lane during ALP mode. The image sensor shall auto-clear the register after stopping the clock.</p> <p>Other bits</p> <p>Reserved for future use</p>

9.12.5.7 USL Hard Standby Mode (Informative)

USL hard standby entry and exit may be controlled in essentially the same manner as with legacy non-USL image sensors. During hard standby mode, all Lanes (including the clock Lane, if applicable) are typically in the forward-direction ULPS state, and the APP cannot access any SNS CCI registers.

SNS entry into hard standby is usually triggered by a hardware input signal (e.g., XSHUTDOWN) which may be asserted asynchronously with respect to image streaming. Each Lane transitions to the forward-direction ULPS state more-or-less immediately; i.e., any in-progress image streaming simply terminates and no PHY ULPS entry command is transmitted. However, the APP normally puts the SNS into soft standby mode beforehand in order to cleanly stop image streaming and put the interface into the Stop state or ULPS. Lane 1 on both the SNS and APP should also be automatically switched to the forward direction when hard standby is triggered.

Exit from hard standby is triggered by the de-assertion of the same hardware input (e.g., XSHUTDOWN) used to trigger hard standby entry, causing each Lane to transition to the Stop state in accordance with applicable PHY initialization procedures in [\[MIPI01\]](#) and [\[MIPI02\]](#).

9.12.5.8 USL Soft Standby Mode and ULPS Entry/Exit (Informative)

Similar to legacy non-USL image sensors, USL supports SNS entry into soft standby mode by the APP writing to a CCI control register causing image streaming to be halted in an orderly manner. For USL, this CCI write operation must occur while in USL_REV mode during an hblank or vblank period. The APP then uses the USL protocol to switch back to USL_FWD mode, in order to enable the SNS to output all or part of the image frame that was in progress at the moment the control register was written, ending with all data Lanes in the Stop state (and the clock Lane still running in the D-PHY ALP Mode case). The SNS then switches to USL_REV mode to enable the APP to issue further commands to the SNS.

Once streaming is halted, the APP can choose different courses of action. For example, if performing a time-critical SNS mode change, the APP can then immediately update the required SNS CCI registers, concluding the process by writing to a CCI control register that requests the restart of image streaming, and then switching back to USL_FWD mode using the USL protocol. Once returned to USL_FWD mode, and prior to actually restarting image streaming, the SNS may have to disable, reconfigure, and relock one or more internal PLLs, possibly requiring the SNS to automatically stop and restart the clock Lane, if applicable.

Another possible course of action is for the APP to put the SNS into an extended sleep state while awaiting further commands from the APP. The preferred method for accomplishing this is for the APP to first transmit a PHY ULPS entry command to the SNS on Lane 1 followed by the SNS, in turn, transmitting the PHY ULPS entry command to the APP on all forward-direction data Lanes. This method is preferred because Lane 1 remains in USL_REV mode throughout ULPS, thereby enabling the APP to subsequently transmit ULPS wakeup signaling to the SNS over Lane 1 when needed; this is also the reason why “reverse direction ULPS” support is recommended for both D-PHY and C-PHY Lane 1 in *Section 7.3*. With D-PHY ALP Mode, the SNS then stops the clock Lane and puts it into ULPS after all data Lanes have been put into ULPS. At this point, the SNS can internally power-down unnecessary circuitry while still retaining internal register states and remaining in USL_REV mode.

For D-PHY ALP mode, ULPS wakeup while in soft standby mode requires the APP to transmit a long ALP Wake pulse to the SNS as described in *[MIPI01]*. Once the SNS starts receiving the latter pulse on data Lane 1 (as signaled by the PPI, for example), it can then start transmitting a long ALP Wake pulse to the APP on each unidirectional data Lane, resulting in a total round-trip wakeup latency which is about the same as the latency in either the forward or reverse direction. The SNS also wakes-up the clock Lane to the Stop state and then restarts it in order to enable the APP to access SNS CCI registers.

For C-PHY ALP mode, ULPS wakeup while in soft standby mode requires the APP to transmit an extended ALP-Pause Wake wire state to the SNS followed by an ALP “Stop” command as described in *[MIPI02]*. At this point, the SNS can similarly signal ULPS wakeup on each unidirectional Lane. However, this results in a total round-trip wakeup latency which is about twice the latency in either the forward or reverse direction. The impact of this can be reduced or eliminated by the APP performing more transactions (e.g., CCI register accesses) using Lane 1 while wakeup is in-progress on the other Lanes, effectively hiding the additional latency. Another possible solution is put either C-PHY Lane 1 or all the other Lanes into ULPS, but not both. For example, the APP could request ULPS by writing to a CCI control register using Lane 1; the SNS would then put all unidirectional Lanes into ULPS while leaving Lane 1 in the Stop state. Conversely, the APP could request ULPS by putting only Lane 1 into ULPS, with the SNS leaving all unidirectional Lanes in the Stop state.

9.13 Data Scrambling

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

Data Scrambling shall be applied on a per-Lane basis, as illustrated in **Figure 88**. Each output of the Lane Distribution Function shall be individually scrambled by a separate scrambling function dedicated to that Lane, before the Lane data is sent to the PHY function over the Tx PPI.

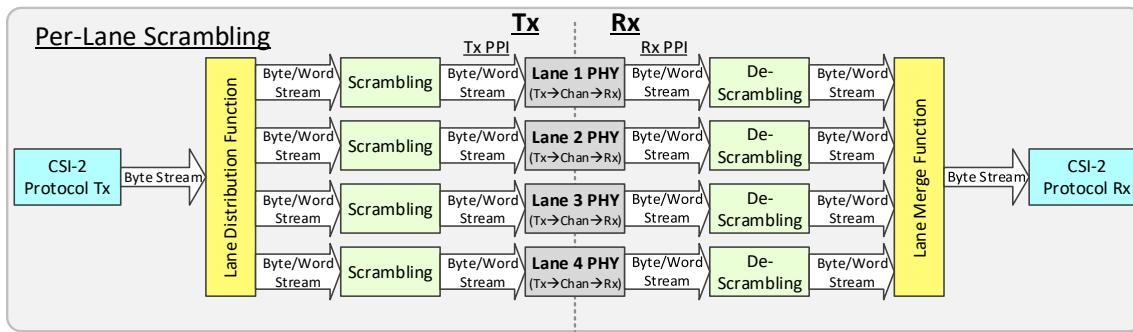


Figure 88 System Diagram Showing Per-Lane Scrambling

2007

9.13.1 CSI-2 Scrambling for D-PHY

Figure 89 shows the format of a burst transmission of two packets over two Lanes when the D-PHY physical layer is used. After the Start of Transmission, HS-ZERO and HS-SYNC are transmitted, the Packet Header and data payload are distributed across the two Lanes.

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

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

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

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

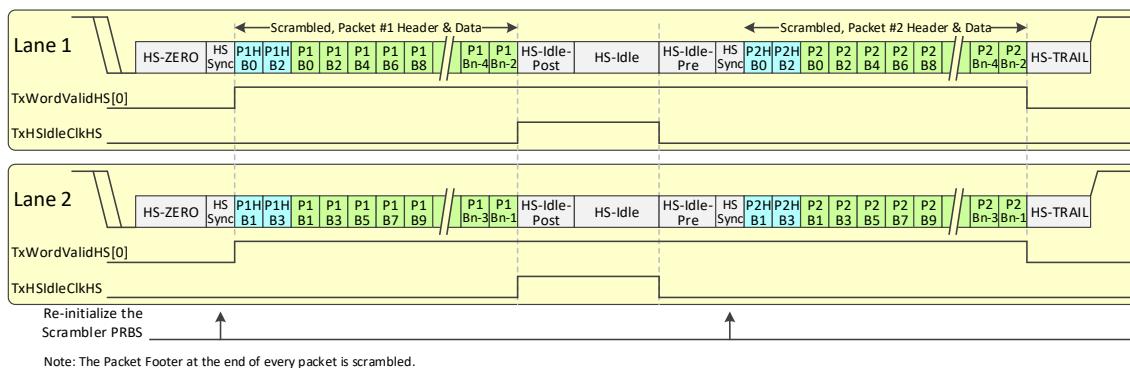


Figure 89 Example of Data Bursts in Two Lanes Using the D-PHY Physical Layer

9.13.2 CSI-2 Scrambling for C-PHY

Figure 90 shows the format of a burst transmission of two packets over two Lanes when the C-PHY physical layer is used. After the Start of Transmission, Preamble, and Sync are transmitted, the Packet Header is replicated twice on each Lane, and data payloads of each packet are distributed across the two Lanes. If the C-PHY physical layer is used, then the scrambler LFSR in each Lane shall be initialized at the beginning of every Long Packet Header or Short Packet, using one of the sixteen-bit seed values assigned to each Lane. This initialization takes place each time the Sync Word is transmitted.

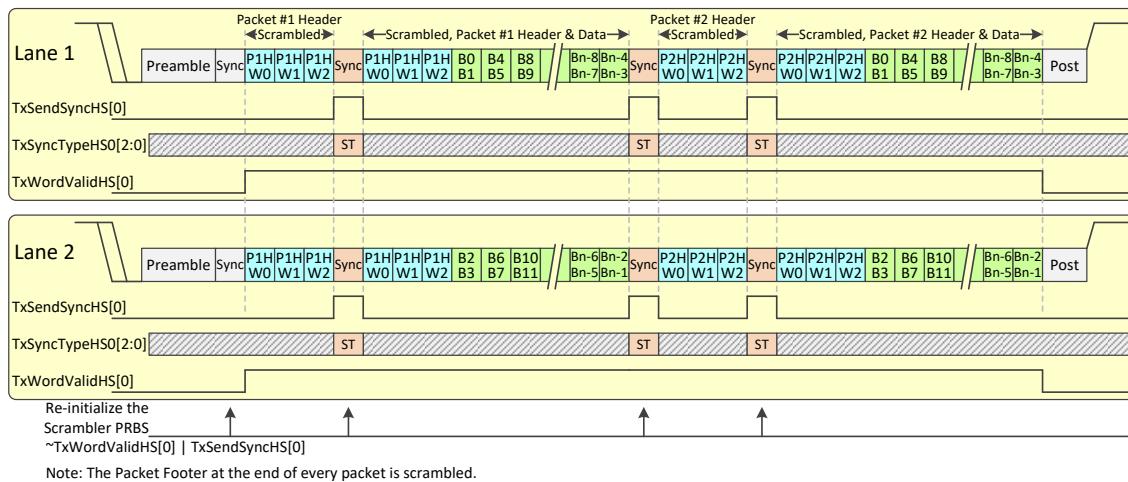


Figure 90 Example of Data Bursts in Two Lanes Using the C-PHY Physical Layer

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

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

2040
2041
2042
Table 28 shows the five possible Sync Types that the C-PHY supports. The Sync Word values are normatively specified in the C-PHY Specification and duplicated in **Table 28** for convenience. The CSI-2 protocol uses only the first four out of the five possible Sync Types, which simplifies the implementation.

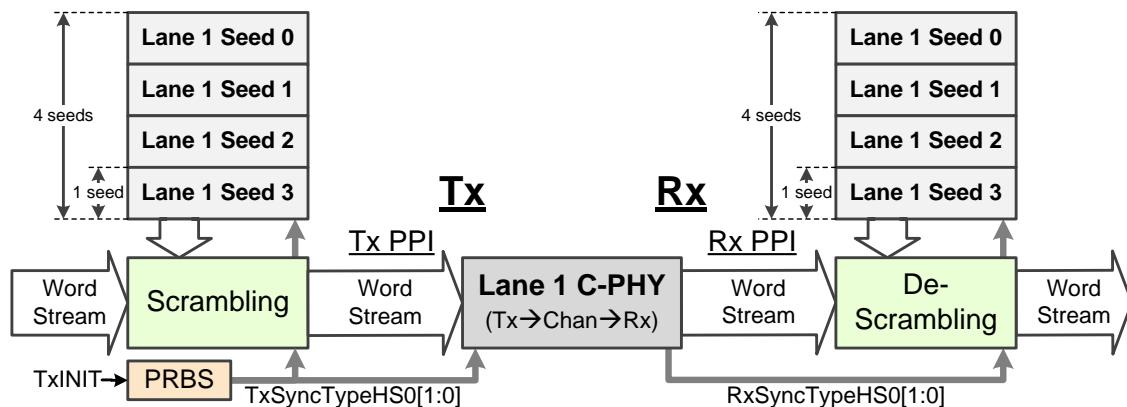
2043 **Table 28 Symbol Sequence Values Per Sync Type**

Sync Type	Sync Value	TxSyncTypeHS0[2:0], TxSyncTypeHS1[2:0]	Scrambler and Descrambler Seed Index
Type 0	3444440	0	0
Type 1	3444441	1	1
Type 2	3444442	2	2
Type 3	3444443	3	3
Type 4	3444444	4	N/A

2044 **Note:**

2045 When a single seed value is used, Sync Type 3 is the default Sync Word value.

2046 **Figure 91** shows the architecture of the scrambling in a single Lane. The pseudo-random number generated by the PRBS shall be used as the seed index to select the initial seed value from the seed list prior to sending the packet. This seed index shall also be sent to the C-PHY using the PPI signals TxSyncTypeHS0[1:0]. TxSyncTypeHS0[2] is always zero. TxSyncTypeHS1 [2:0] is used similarly for a 32-bit data path. The C-PHY ensures that the very first packet in a burst begins with a Sync Word using Sync Type 3.

2051 **Figure 91 Generating Tx Sync Type as Seed Index (Single Lane View)**

2052 The seed list may contain either one or four initial seed values. Transmitters and receivers shall have the capability to select exactly one seed value from a list of seeds. When a single seed value is used, that seed shall be identified as Seed 3 and the transmitter shall always transmit Sync Type 3. Transmitters and receivers should also have the capability to select a seed value from a list of four seed values, as shown in **Figure 91**. When a list of four seed values is used then Sync Type 0 through Sync Type 3 shall be used to convey the seed index value from the transmitter to the receiver.

2058 When the list of four seeds is used, the two-bit seed index shall be generated in the transmitter using a pseudo random generator (e.g., PRBS).

2060 Slight differences in the implementation of the PRBS generator will not affect the interoperability of the transmitter and receiver, because the receiver responds to the seed index chosen in the transmitter and 2061 conveyed to the receiver using the Sync Type.

2063 At the receiver, the C-PHY decodes the Sync Word and passes the 2-bit Sync Type value to the CSI-2 protocol logic. The CSI-2 protocol logic uses the two-bit value as a seed index to select one of four seed values to 2064

2065
2066
2067
2068 initialize the descrambler. This concept is shown in the single Lane diagram in **Figure 91**. **Figure 88** shows the use of the PPI signals to select which seed value was used to initialize the scrambler and descrambler. Since the seed selection field is transmitted via the Sync Word, no other mechanism is needed to coordinate the choice of specific descrambler initial seed values at the receiver.

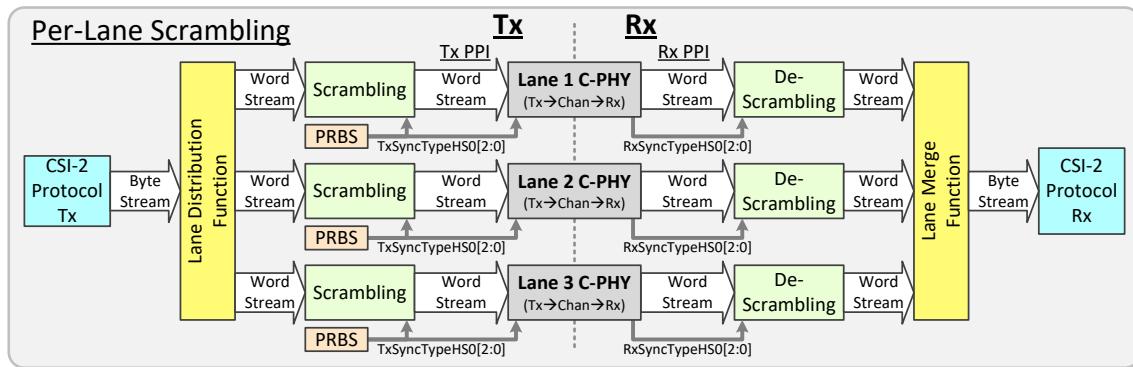


Figure 92 Generating Tx Sync Type Using the C-PHY Physical Layer

9.13.3 Scrambling Details

The Long Packet Header, Data Payload, Long Packet Footer (which may include a Filler Byte), and Short Packets shall be scrambled. Special data fields generated by the PHY that are beyond the control of the CSI-2 protocol shall not be scrambled. For clarity, *Table 29* lists all of the fields that are not scrambled.

Table 29 Fields That Are Not Scrambled

PHY	PHY-Generated	CSI-2-Protocol-Generated
D-PHY	<ul style="list-style-type: none"> • HS-Zero • Sync Word (aka Leader Sequence) • HS Trail • SoT • EoT • HS-Idle • All fields of the deskew sequence (aka deskew burst) including: <ul style="list-style-type: none"> • HS-Zero • Deskew sync pattern • '01010101' data • HS-Trail 	<ul style="list-style-type: none"> • LP Mode transactions for SoT, EoT and ULPS
C-PHY	<ul style="list-style-type: none"> • Preamble (including t₃-PREBEGIN t₃-PROGSEQ and t₃-PREEND) • Sync Word • Post • SoT • EoT 	<ul style="list-style-type: none"> • Sync Word inserted via PPI command • LP Mode transactions for SoT, EoT and ULPS

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

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

The initial D-PHY seed values in *Table 30* should be used to initialize the D-PHY scrambler LFSR in Lanes 1 through 8.

Table 30 D-PHY Scrambler PRBS Initial Seed Values for Lanes 1 Through 8

Lane	Initial Seed Value
1	0x0810
2	0x0990
3	0x0a51
4	0x0bd0
5	0x0c30
6	0x0db0
7	0x0e70
8	0x0ff0

2081 The initial C-PHY seed values in **Table 31** should be used to initialize the C-PHY scrambler LFSR in Lanes
 2082 1 through 8. The table provides initial seed values for each of the four possible Sync Type values per Lane
 2083 number. If only a single Sync Type is used, then it shall default to Sync Type 3.

2084 **Table 31 C-PHY Scrambler PRBS Initial Seed Values for Lanes 1 Through 8**

Lane	Initial Seed Value			
	Sync Type 0	Sync Type 1	Sync Type 2	Sync Type 3
1	0x0810	0x0001	0x1818	0x1008
2	0x0990	0x0180	0x1998	0x1188
3	0x0a51	0x0240	0x1a59	0x1248
4	0x0bd0	0x03c0	0x1bd8	0x13c8
5	0x0c30	0x0420	0x1c38	0x1428
6	0x0db0	0x05a0	0x1db8	0x15a8
7	0x0e70	0x0660	0x1e79	0x1668
8	0x0ff0	0x07e0	0x1ff8	0x17e8

2085 For D-PHY and C-PHY systems requiring more than eight Lanes, **Annex G** provides 24 additional seed
 2086 values for Lanes 9 through 32, as well as a mechanism for finding seed values for Lanes 33 and higher. For
 2087 each seed value, the LSB corresponds to scrambler PRBS register bit Q0 and the MSB corresponds to bit
 2088 Q15.

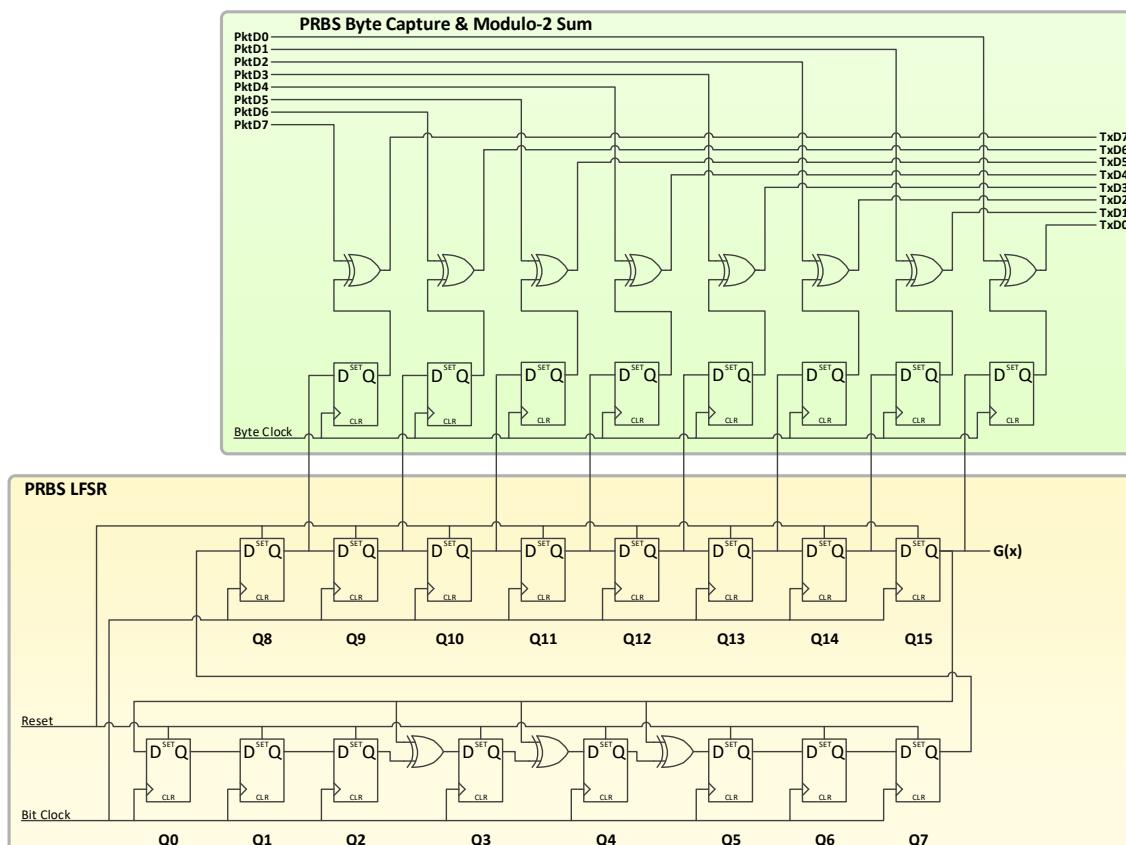
2089 The LFSR shall generate an eight-bit sequence at $G(x)$ for every byte of Payload data to be scrambled, starting
 2090 from its initial seed value. The LFSR shall generate new bit sequences of $G(x)$ by advancing eight bit cycles
 2091 for each subsequent Payload data byte.

2092 Scrambling shall be achieved by modulo-2 bit-wise addition (X-OR) of a sequence of eight bits $G(x)$ with
 2093 the CSI-2 Payload data to be scrambled.

2094
 2095 **Implementation Tip:** the 8-bit value from the PRBS is the flip of bits Q15:Q8 of the PRBS LFSR register on
 2096 every 8th bit clock. The designer might choose to implement the PRBS LFSR in parallel form to shift the
 2097 equivalent of 8 places in a single byte clock, or the PRBS LFSR might even be configured to shift a multiple
 of 8 places in a single word clock.

2098 For the example shown in **Figure 93**, Q[15:8] are captured in a temporary register, then the PRBS LFSR is
 2099 shifted eight times before Q[15:8] are captured again. The scrambling is performed as follows:

- 2100 • TxD[7] = PktD[7] \oplus Q'[8];
- 2101 • TxD[6] = PktD[6] \oplus Q'[9];
- 2102 • TxD[5] = PktD[5] \oplus Q'[10];
- 2103 • TxD[4] = PktD[4] \oplus Q'[11];
- 2104 • TxD[3] = PktD[3] \oplus Q'[12];
- 2105 • TxD[2] = PktD[2] \oplus Q'[13];
- 2106 • TxD[1] = PktD[1] \oplus Q'[14];
- 2107 • TxD[0] = PktD[0] \oplus Q'[15];



2108 **Figure 93 PRBS LFSR Serial Implementation Example**

Table 32 illustrates the sequence of the PRBS register one bit at a time, starting with the initial seed value for Lane 2. The data scrambling sequence is the output $G(x)$. The first bit output from the scrambler is the value output from $G(x)$ (also Q15 of the register in **Figure 93**) when the register contains the initial seed value.

Table 32 Example of the PRBS Bit-at-a-Time Shift Sequence

t	Q15	Q14	Q13	Q12	Q11	Q10	Q9	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0	LFSR
0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0x1188
1	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0x2310
2	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0x4620
3	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0x8C40
4	0	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	0x18B9
5	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0x3172
6	0	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0	0x62E4
7	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0	0	0xC5C8
8	1	0	0	0	1	0	1	1	1	0	1	0	1	0	0	1	0x8BA9
9	0	0	0	1	0	1	1	1	0	1	1	0	1	0	1	1	0x176B
10	0	0	1	0	1	1	1	0	1	1	0	1	0	1	1	0	0x2ED6
11	0	1	0	1	1	1	0	1	1	0	1	0	1	1	0	0	0x5DAC
12	1	0	1	1	1	0	1	1	0	1	0	1	1	0	0	0	0xBB58
13	0	1	1	1	0	1	1	0	1	0	0	0	1	0	0	1	0x7689
14	1	1	1	0	1	1	0	1	0	0	0	0	1	0	0	1	0xED12
15	1	1	0	1	1	0	1	0	0	0	0	1	1	1	0	1	0xDA1D
16	1	0	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0xB403

Table 33 shows the first ten PRBS Byte Outputs produced by the PRBS LFSR in Lane 2 when the D-PHY physical layer is being used.

Table 33 Example PRBS LFSR Byte Sequence for D-PHY Physical Layer

Scrambling Sequence	PRBS Register	PRBS Byte	Input Byte	Output Byte
Initial Seed, Byte 0	0x0990	0x90	0x2b	0xbb
Byte 1	0x91f1	0x89	0xd	0x84
Byte 2	0xee29	0x77	0x63	0x14
Byte 3	0x3dbe	0xbc	0x00	0xbc
Byte 4	0xbba5	0xdd	0x00	0xdd
Byte 5	0xbcb3	0x3d	0x00	0x3d
Byte 6	0xaalc	0x55	0x19	0x4c
Byte 7	0x061a	0x60	0x41	0x21
Byte 8	0x1a96	0x58	0x22	0x7a
Byte 9	0x942a	0x29	0x53	0x7a

2116 **Table 34** shows an example of the PRBS Word Outputs at the beginning of a packet, that are produced by the
 2117 PRBS LFSR in Lane 2 when the C-PHY physical layer is being used. In this example, the initial seed value
 2118 used for transmitting the first copy of the packet header corresponds to Sync Type 3 for Lane 2. The initial
 2119 seed value used for transmitting the second copy of the packet header corresponds to Sync Type 0 for Lane
 2120 2. As described in **Section 9.13.2**, the C-PHY ensures that the very first packet in a burst begins with a Sync
 2121 Type 3 sync word. For all subsequent sync words transmitted in a burst, such as when LRTE is enabled, the
 2122 CSI-2 protocol layer selects any of Sync Type 0 through Sync Type 3.

2123 **Table 34 Example PRBS LFSR Byte Sequence for C-PHY Physical Layer**

Scrambling Sequence Word #	PRBS Register	PRBS Word	Input Word	Output Word
Initial Seed, Header[47:32]	0x1188	0xd188	0x2b00	0xfa88
Header[31:16]	0xb403	0xd82d	0x13b0	0xcb9d
Header[15:0]	0xd613	0x406b	0x31c8	0x71a3
Sync Word (see Note 1 below)	0xc672	0x0663	0xxxxx	0xxxxx
Re-initialized Seed, Header[47:32]	0x0990	0x8990	0x2b00	0xa290
Header[31:16]	0xee29	0xbc77	0x13b0	0xafc7
Header[15:0]	0xbba5	0x3ddd	0x31c8	0x0c15
Word 0	0xaalc	0x6055	0xd000	0xb055
Word 1	0x1a96	0x2958	0x1360	0x3a38
Word 2	0x35f4	0x0fac	0x094c	0x06e0
Word 3	0x7b70	0xdede	0x100b	0xed5
Word 4	0x7873	0x1e1e	0x5fb8	0x41a6
Word 5	0x3338	0x3ccc	0xd030	0xecfc
Word 6	0xfe9c	0xd17f	0x0003	0xd17c
Word 7	0x3303	0xe0cc	0xd039	0x30f5
Word 8	0xfbaf	0x1ddf	0xa35b	0xbe84
Word 9	0xead8	0x3757	0x00ea	0x37bd

2124

Note:

1. *The Output Word is irrelevant in this word clock cycle because the CSI-2 protocol layer asserts one of the TxSendSyncHS PPI signals with a valid selection on the corresponding TxSyncTypeHS signals to transmit a sync word instead of a scrambled data word.*

9.14 Smart Region of Interest (SROI)

The Smart Region of Interest (SROI) feature supports the adaptive transfer of rectangular Regions of Interest (ROI). SROI can be used to reduce data bandwidth by selectively transmitting one or more smaller ROIs carved out from the original picture, such as a human face or a license plate. SROI has use cases in cameras for computer vision, and machine vision applications beyond mobile markets including industry, surveillance, and IoT.

In addition to reducing data bandwidth, SROI benefits include:

- **High Frame Rate / Low Latency:** Data reduction can increase the sensor frame rate and reduce processing workloads for application processors.
- **High Resolution:** A high-resolution image can be extracted by capturing the necessary information without increasing the amount of data transferred.
- **Low Power Dissipation:** Processing workload reductions can save system power.

Product platforms may facilitate SROI capability using:

1. **Integrated SROI (ISROI) System (*Figure 94*):** By integrating Vision Digital Signal Predecessor (VDSP) within a multi-stack image sensor module (SNS) to generate SROI data.
2. **External SROI (ESROI) System (*Figure 95*):** By utilizing an external SROI VDSP component or integrated VDSP within an APP to generate SROI data.

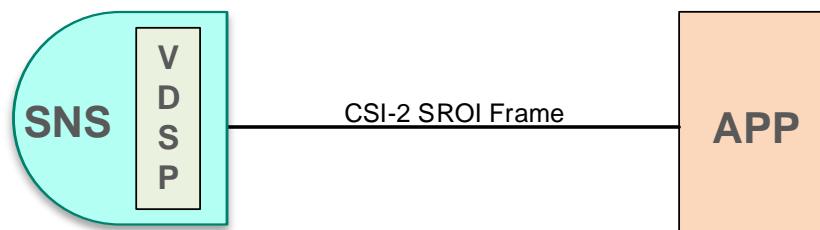


Figure 94 ISROI System Supporting Multi-Stack SNS with Integrated SROI VDSP

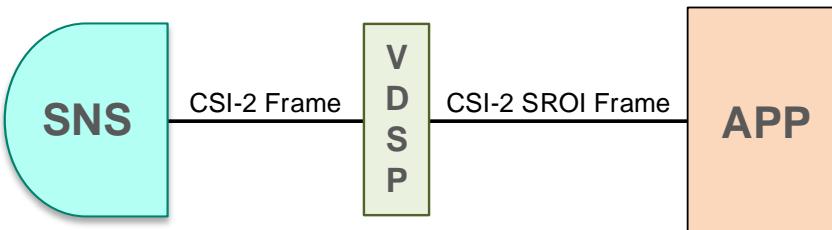


Figure 95 ESROI System Supporting Standard SNS with an External SROI VDSP

9.14.1 Overview of SROI Frame Format

As shown in **Figure 96**, each rectangular ROI is defined by position (X, Y coordinates of ROI rectangle upper left-hand corner pixel), size (Height and Width of ROI rectangle, in pixels), and other metadata (e.g., ADC bit depth; see **Table 37**).

Each ROI has corresponding a ROI Information field with values for position, size, and other items (see **Section 9.14.5**). ROI Information values either are determined by a detection function integrated into an APP or image sensor, or for some use cases such as factory automation are set in advance. The method of detecting an ROI is out of scope for this specification.

The maximum number of ROI is also out of scope for this specification, because it is implementation-specific. However, prior to the SROI transfer the application processor and the image sensor shall agree on both the maximum number of ROI and the ROI ID format in **Table 37**. ROI ID (1 byte) and ROI ID (2 byte) cannot both coexist.

SROI Frames use three data packet types:

- **SROI Short Packet** is used for transmitting a Frame Start (FS) Packet and a Frame End (FE) Packet in an image frame with the SROI Long Packet.

If Line synchronization packets are needed, then Line Start (LS) Packets and Line End (LE) Packets are also permitted as SROI Short Packets.

- **SROI Embedded Data Packet** is used for transmitting ROI Information, as detailed in **Section 9.14.5**.

If the SROI Embedded Data Packet is present in an image frame, it shall be located at the beginning of the image frame.

- **SROI Long Packet** (excludes the SROI Embedded Data Packet) is used for transmitting ROI image data for an image frame.

The SROI Long Packet should use the same image Data Type used to transmit the image frame's normal, non-SROI image lines (e.g., RAW10), however a User-Defined Data Type is also permitted. For SROI transfers, it is not required for all SROI Long Packets within a given image frame to have equal length.

When there are multiple ROIs within one line of original image data, all ROI data shall be merged together into a single SROI Long Packet, with no blanking between the regions. See example of A(0) and D(0) in **Figure 96**.

Any image region overlapped by multiple ROIs, for example A(n-2) and B(0) in **Figure 96**, shall be transmitted only once (i.e., shall not be transmitted multiple times, one per ROI). As a result, each overlapped image region shall be counted only once when calculating the value of the SROI Long Packet Word Count field (i.e., shall not be counted multiple times, one per ROI).

The term **SROI Packet** means any SROI Short Packet, SROI Embedded Data Packet, or SROI Long Packet.

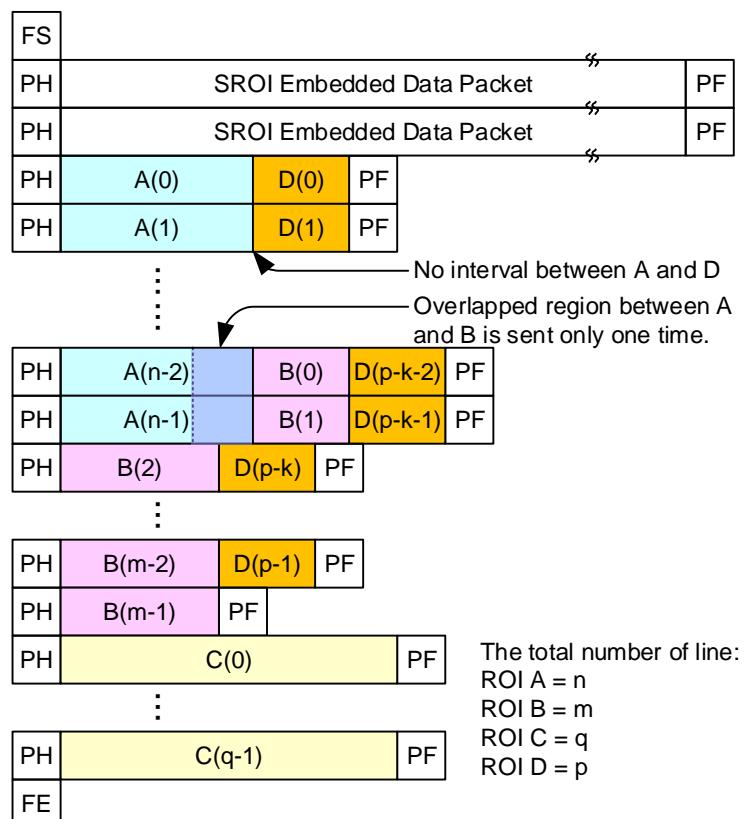
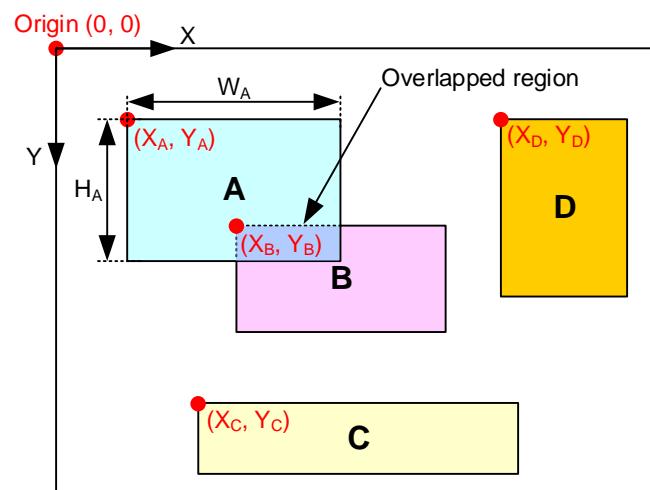


Figure 96 SROI Frame Format Example

9.14.2 Transmission of SROI Embedded Data Packet

This Section specifies when the image sensor is required to send the SROI Embedded Data Packet (SEDP) in the image frame.

This depends upon whether the application processor (APP) knows, vs. does not know, all required ROI Information (see *Table 35*). When the image sensor does send the SEDP, the AP's method of detecting SROI Packets depends upon the SROI Packet Option (Option 1 vs. Option 2, see *Section 9.14.3*) that the AP and image sensor have agreed to use.

- If the application processor is aware of all ROI information, then the VDSP may be configured by the APP to optionally transmit the SEDP.

If SEDP is sent and SROI Packet Option 1 is used, then the APP can detect SROI Packets in the image frame by inspecting the Virtual Channel value (see *Section 9.14.3.1*).

- If the application processor is unaware of ROI information, then the VDSP shall generate the ROI data.

The VDSP supporting SROI shall support a 16-bit control register, **REG_SROI_CONTROL[15:0]**. The SROI features and capabilities are detailed in the sections below. The unused bits are reserved for future developments.

The following four bits are used to configure the ROI data generation by the VDSP:

- **REG_SROI_CONTROL[0]** (field **SROI_EN**) shall be used to enable SROI capability:

- **1'b0**: Disable SROI
- **1'b1**: Enable SROI

- **REG_SROI_CONTROL[1]** (field **ROI_AWARE**) shall be used to select APP or VDSP as the ROI aware generator:

- **1'b0**: APP is not aware of the ROI. Enable VDSP to internally generate and transmit the SROI.
- **1'b1**: APP is aware of the ROI. Enable VDSP to transmit SROI predetermined by APP.

- **REG_SROI_CONTROL[4:3]** (field **ROI_GEN**) shall be used to configure the ROI data generation:

- **2'b00**: VDSP generates ROI Meta Data (SEDP.)
- **2'b01**: VDSP generates ROI Meta Data (SEDP) with the captured CSI-2 frame
- **2'b10**: VDSP generates ROI Meta Data (SEDP) and ROI Pixel Data
- **2'b11**: VDSP generates ROI Meta Data (SEDP) and ROI Pixel Data with the captured CSI-2 frame

The method the APP uses to detect SROI Packets in an image frame depends upon which SROI Packet Option is in use: by Virtual Channel for Option 1 (see *Section 9.14.3.1*), or by Data Type for Option 2 (see *Section 9.14.3.2*). The VDSP shall transport ROI data using a predefined Data Type or a VC.

- **REG_SROI_CONTROL[2]** (field **DT_VC_SEL**) shall be used to transmit ROI data using DT or VC:

- **1'b0**: VDSP transports ROI Meta Data and Pixel Data using Data Type.
- **1'b1**: VDSP transports ROI Meta Data and Pixel Data using a unique VC.

- **REG_SROI_CONTROL[15:8]** (field **UNIQUE_SROI_ID**) shall be used to configure DT or VC for SROI generation by VDSP.

- **REG_SROI_CONTROL[7:5]** bits are reserved for future use.

Example use cases are shown in *Section 9.14.4*.

2221

Table 35 Transmission of SROI Embedded Data Packet

		APP Knowledge of All Required ROI Information	
		APP Knows (See <i>Section 9.14.4.1</i>)	APP Does Not Know (See <i>Section 9.14.4.2</i>)
SEDP Required		No	Yes
SROI Packet Option	Option 1 (See <i>Section 9.14.3.1</i>)	APP either knows SROI Packet, or can detect SROI Packet by Virtual Channel	APP can detect SROI Packet by Virtual Channel
	Option 2 (See <i>Section 9.14.3.2</i>)	APP knows SROI Packet	APP can detect SROI Packet by Data Type

9.14.3 SROI Packet Detection Options

The following sub-sections detail the two options for distinguishing the SROI Packets in an image frame from the non-SROI Packets.

Prior to the SROI transfer, the APP and the image sensor shall agree on which of the two options (i.e., Option 1 vs. Option 2) will be used.

9.14.3.1 SROI Packet Option 1

Option 1 distinguishes SROI Packets from non-SROI packets by using a different Virtual Channel value.

For Option 1, all SROI Packets for a given image frame shall use the same Virtual Channel, and this SROI Virtual Channel shall be different from the Virtual Channel used in the image frame's non-SROI Packets (see **Figure 97**). Virtual Channel Interleaving may be used.

An image frame that includes SROI Packets may use Data Type Interleaving, but only if the Data Type used in the SROI Long Packet is different from the Data Type used in the non-SROI Long Packet (e.g., PDAF).

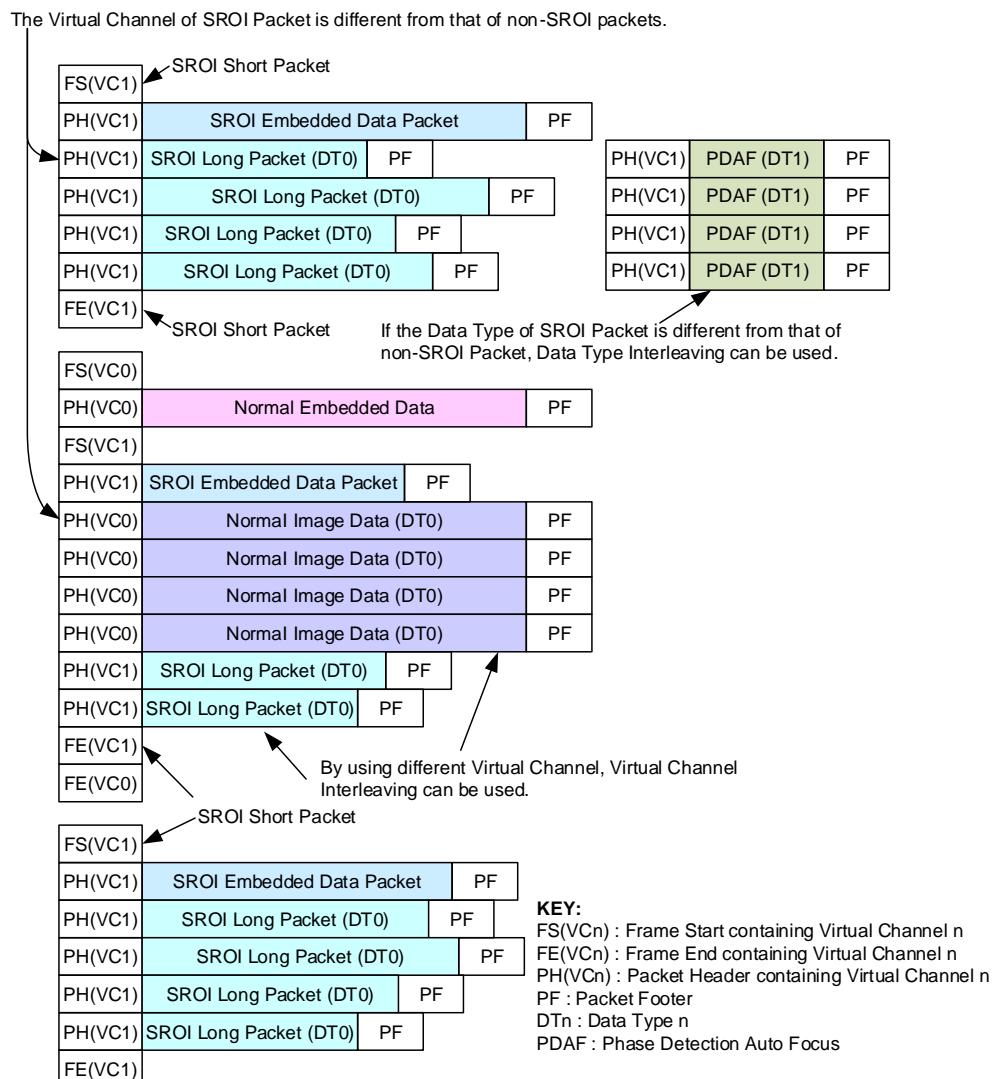


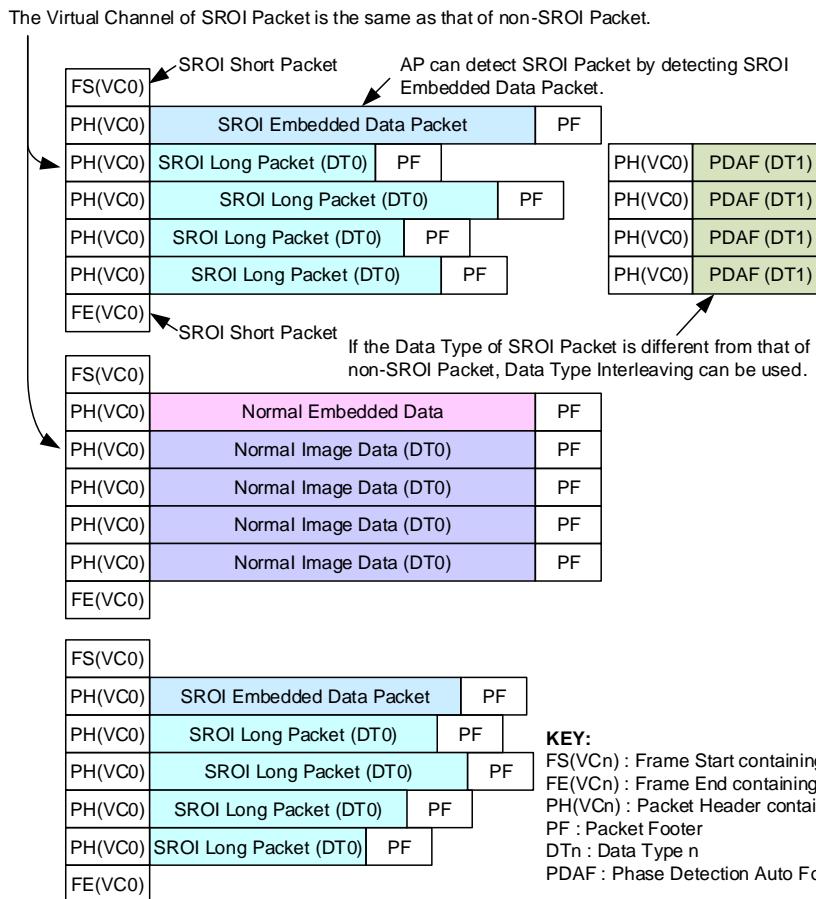
Figure 97 SROI Packet Option 1

9.14.3.2 SROI Packet Option 2

2233 Option 2 distinguishes SROI Packets from non-SROI packets by Data Type; in particular, by detecting the
 2234 presence of the SROI Embedded Data Packet in the image frame.

2235 For Option 2, all SROI Packets for a given image frame shall use the same Virtual Channel that the non-
 2236 SROI Packets use (see **Figure 98**).

2237 An image frame that includes SROI Packets may use Data Type Interleaving, but only if the Data Type used
 2238 in the SROI Long Packet is different from the Data Type used in the non-SROI Long Packet (e.g., PDAF).



2239 **Figure 98 SROI Packet Option 2**

9.14.4 SROI Use Cases (Informative)

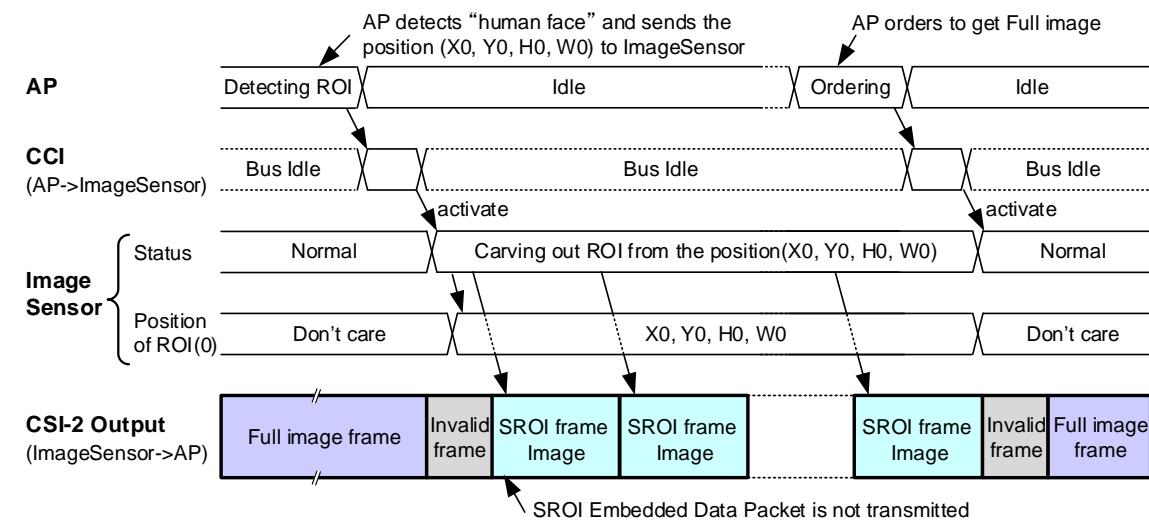
2240 This Section presents informative use cases illustrating ways in which the SROI feature can be used.

9.14.4.1 Use Case 1: ROI Detection by the Application Processor

2241 If the APP is responsible for detecting an ROI, or has ROI information that is set in advance, then the image
2242 sensor is not required to transmit the SROI Embedded Data Packet. An example is shown in **Figure 99**.

2243 In this use case, the APP detects a ROI (e.g., a human face), and then sends the ROI's position and size to an
2244 image sensor through the CCI (see **Section 6**). The image sensor then just carves out the ROI, based on the
2245 ordered position and size, and then keeps sending SROI Long Packets for the same image region.

2246 In this case the SROI Embedded Data Packet is not transmitted, because it would be unnecessary: the APP
2247 already knows all ROI information (e.g., the ROI position and size) required to receive the SROI Packets.



2248 **Figure 99 Use Case 1: SROI Embedded Data Packet Not Transmitted**

9.14.4.2 Use Case 2: ROI Detection by the Image Sensor

If the image sensor is responsible for detecting an ROI, then the image sensor is required to transmit the SROI Embedded Data Packet as illustrated in *Figure 100*.

The APP orders the image sensor to get an ROI (e.g., a human face). The image sensor detects the ROI and carves it out, and then keeps tracking the ROI.

In this case the SROI Embedded Data Packet is (and must be) transmitted, because the APP doesn't know all of the ROI information (e.g., position and size) required to receive the SROI Packets.

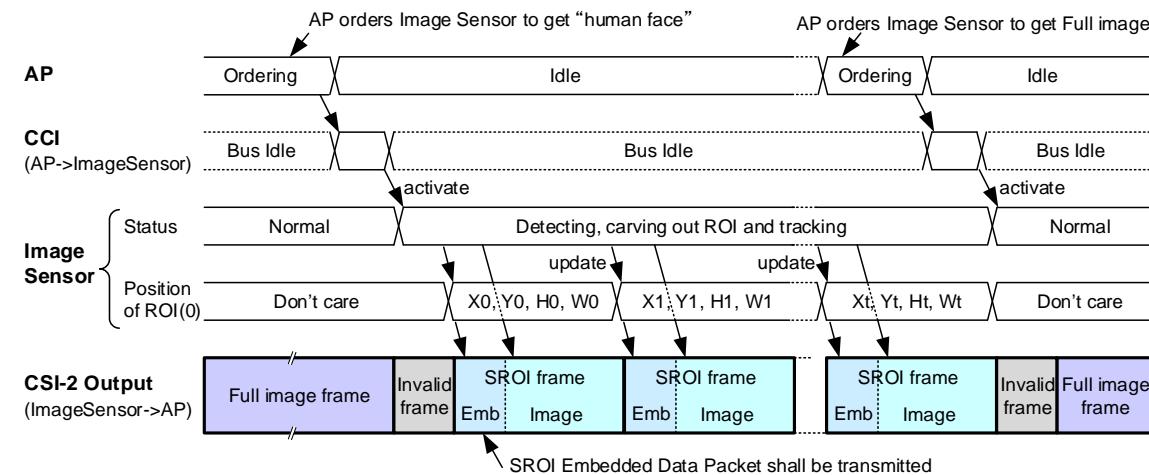


Figure 100 Use Case 2: SROI Embedded Data Packet Is Transmitted

9.14.5 Format of SROI Embedded Data Packet (SEDP)

For SROI transfers, the Embedded Data Format is based on the definition in the MIPI CCS specification [[MIPI04](#)]. The Embedded Data Format Code for SROI frames is 0x0D (1 byte).

The SROI Embedded Data Packet (see [Figure 101](#)) is composed of multiple ROI Information fields, each containing ROI Information about one extracted image region. Every ROI Information field shall start with the ROI ID, followed by the ROI Element Information field defined by [Table 36](#). [Table 37](#) defines the allocation and assignment of values for the Type ID sub-field used in the ROI Element Information field.

At the end of the embedded data line, the End of Line shall be inserted after the end of the last ROI Information field.

The length of the SROI Embedded Data Packet line shall not exceed the length of the full-resolution image data line. After the End of Line, the remainder of the line should be padded with 0x07 characters if the SROI Embedded Data Packet line is shorter than the length of the full-resolution image data line.

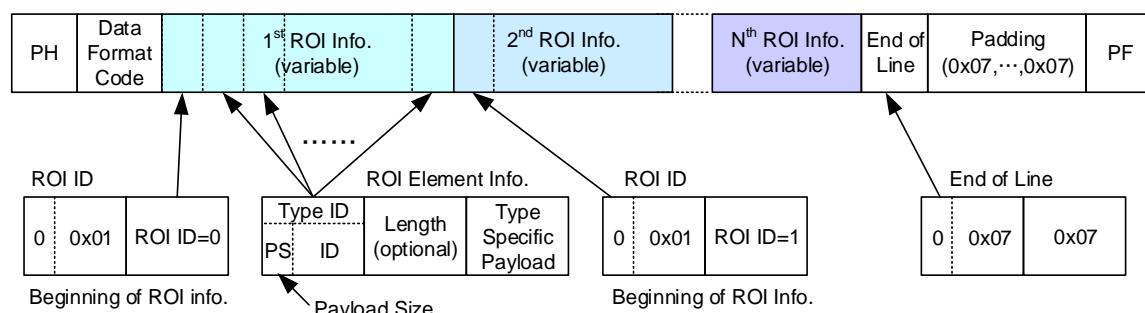


Figure 101 SROI Embedded Data Packet Format

Table 36 ROI Element Information Field Format

Field Name		Size (bits)	Description
Type ID	Payload Size	2	Size of the Type Specific Payload field: 0: 1 byte 1: 2 bytes 2: 4 bytes 3: Size is given by the Length field (below)
	ID	6	Type Identifier (bottom bits of first byte): 0x00 – 0x1F MIPI Defined ID 0x20 – 0x3E User Defined ID 0x3F MIPI Defined ID
Length (optional)		8	If Payload Size is 3 (2'b11): Number of 8-bit data bytes in the Type Specific Payload field <i>Example:</i> A value of 0x20 in the Length field indicates that the Type Specific Payload contains 32 bytes If Payload size is 0, 1, or 2: Length field is not included in the ROI Element Information field
Type Specific Payload	Variable		The payload data is byte-based, i.e., the data size shall be divisible by 8 bits.

2269

Table 37 ROI Element Type ID Definitions

Type ID	Name	Description
MIPI Defined IDs		
Payload Size = 1 Byte		
8'b00_0_00000	Reserved	Reserved for future use
8'b00_0_00001	ROI ID	Identification number (1 Byte) of each ROI. (Mandatory)
8'b00_0_00010	ADC Bit Depth	Bit depth of Analog Digital Converter of each ROI
8'b00_0_00011 through 8'b00_0_00110	Reserved	Reserved for future use
8'b00_0_00111	End of Line	The end of SROI Embedded Data line. The value of Type Specific Payload is 0x07.
8'b00_0_01000 through 8'b00_0_11111	Reserved	Reserved for future use
Payload Size = 2 Bytes		
8'b01_0_00000	Reserved	Reserved for future use
8'b01_0_00001	ROI ID (2 Bytes)	Identification number (2 Bytes) of each ROI. (Optional)
8'b01_0_00010 through 8'b01_0_00111	Reserved	Reserved for future use
8'b01_0_01000	X	X coordinate of upper left of region [pixel]
8'b01_0_01001	Y	Y coordinate of upper left of region [pixel]
8'b01_0_01010	Height	Height of region [pixels]
8'b01_0_01011	Width	Width of region [pixels]
8'b01_0_01100 through 8'b01_0_11111	Reserved	Reserved for future use
Payload Size = 4 Bytes		
8'b10_0_00000 through 8'b10_0_11111	Reserved	Reserved for future use
Payload Size = Length		
8'b11_0_00000 through 8'b11_0_11111	Reserved	Reserved for future use

Type ID	Name	Description
User Defined IDs		
Payload Size = 1 Byte		
8'b00_1_00000 through 8'b00_1_11111	User Defined ID	Reserved for user definition
Payload Size = 2 Bytes		
8'b01_1_00000 through 8'b01_1_11111	User Defined ID	Reserved for user definition
Payload Size = 4 Bytes		
8'b10_1_00000 through 8'b10_1_11111	User Defined ID	Reserved for user definition
Payload Size = Length		
8'b11_1_00000 through 8'b11_1_11110	User Defined ID	Reserved for user definition
MIPI Defined ID		
8'b11_1_11111	Reserved	Reserved for future use

9.15 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, the USL Data Type (code 0x38), and Data Types for SROI Long Packet, long packets can have arbitrary length. The spacing between packets can also vary.

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

9.16 Frame Format Examples

2283

This is an informative section.

2284

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

2285

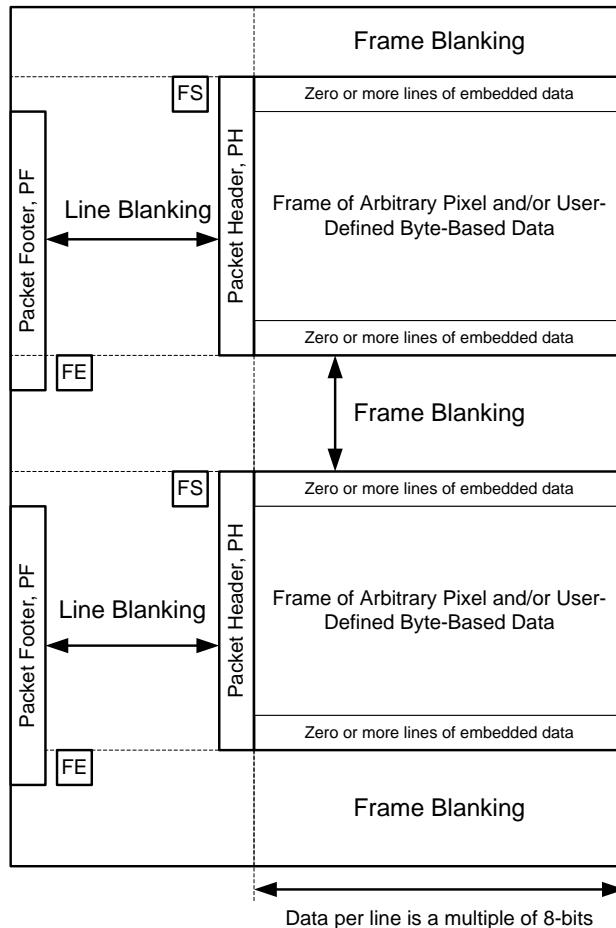
- General Frame Format Example, **Figure 102**

2286

- Digital Interlaced Video Example, **Figure 103**

2287

- Digital Interlaced Video with accurate synchronization timing information, **Figure 104**



KEY:

PH – Packet Header

PF – Packet Footer + Filler (if applicable)

FS – Frame Start

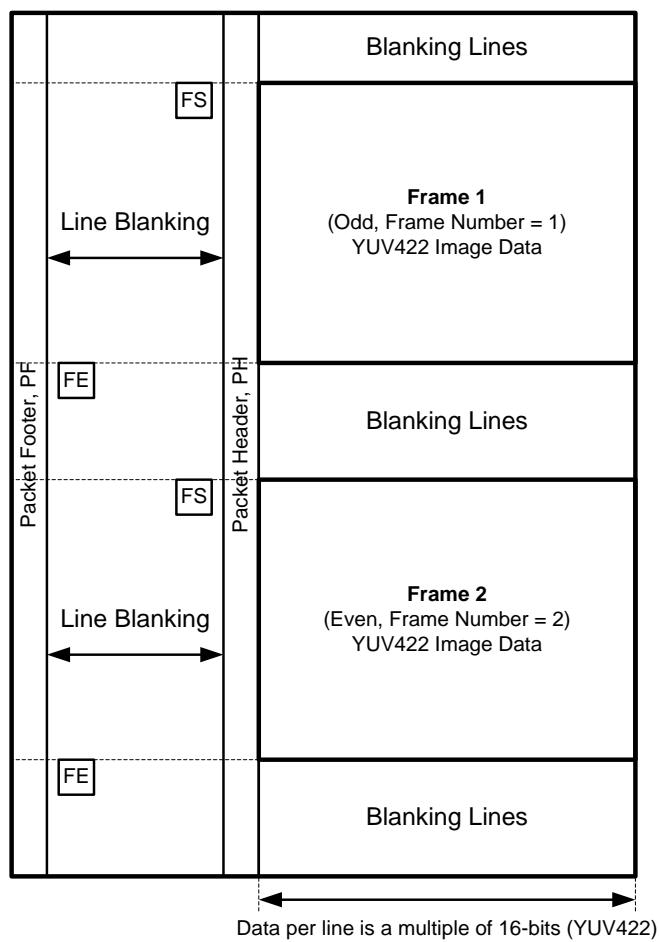
FE – Frame End

LS – Line Start

LE – Line End

2288

Figure 102 General Frame Format Example

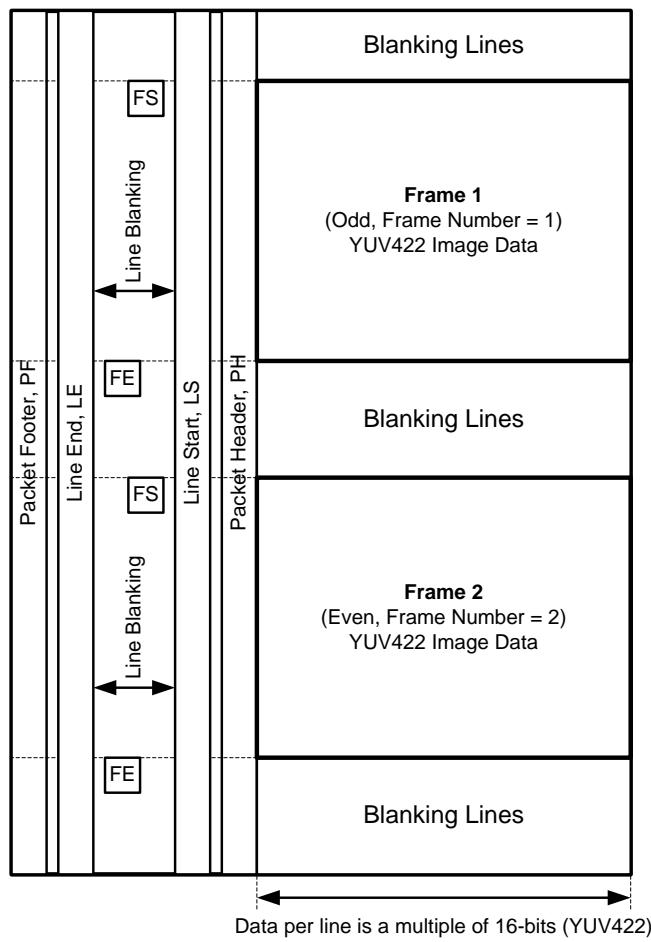
**KEY:**

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

PF – Packet Footer + Filler (if applicable)
 FE – Frame End
 LE – Line End

2289

Figure 103 Digital Interlaced Video Example

**KEY:**

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

PF – Packet Footer + Filler (if applicable)
 FE – Frame End
 LE – Line End

2290

Figure 104 Digital Interlaced Video with Accurate Synchronization Timing Information

9.17 Data Interleaving

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

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

- Data Type
- Virtual Channel Identifier

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

9.17.1 Data Type Interleaving

The Data Type value uniquely defines the data format for that packet of data. The receiver uses the Data Type value in the packet header to de-multiplex data packets containing different data formats as illustrated in **Figure 105**. Note, in the figure the Virtual Channel Identifier is the same in each Packet Header.

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

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

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

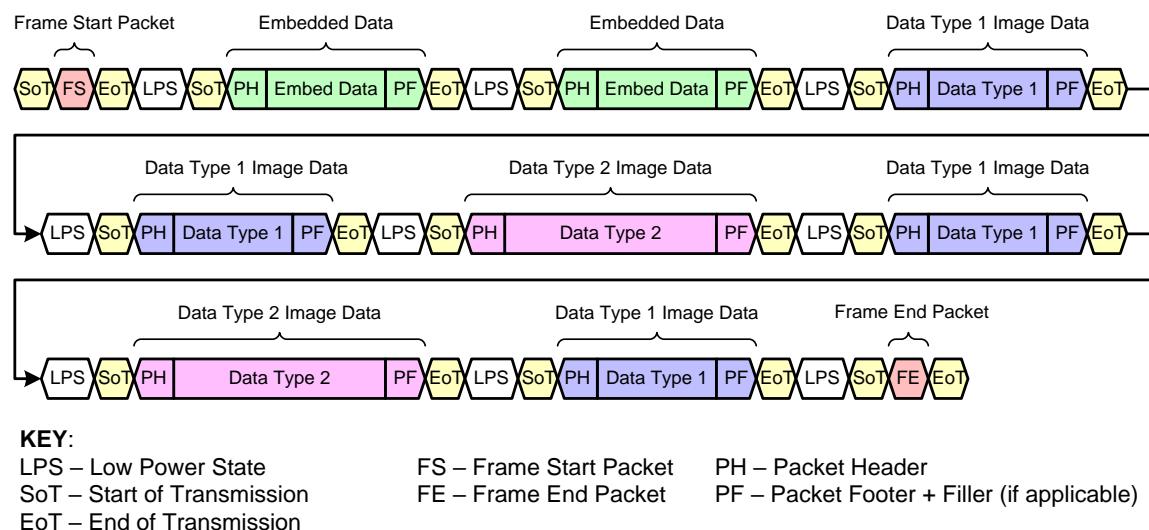
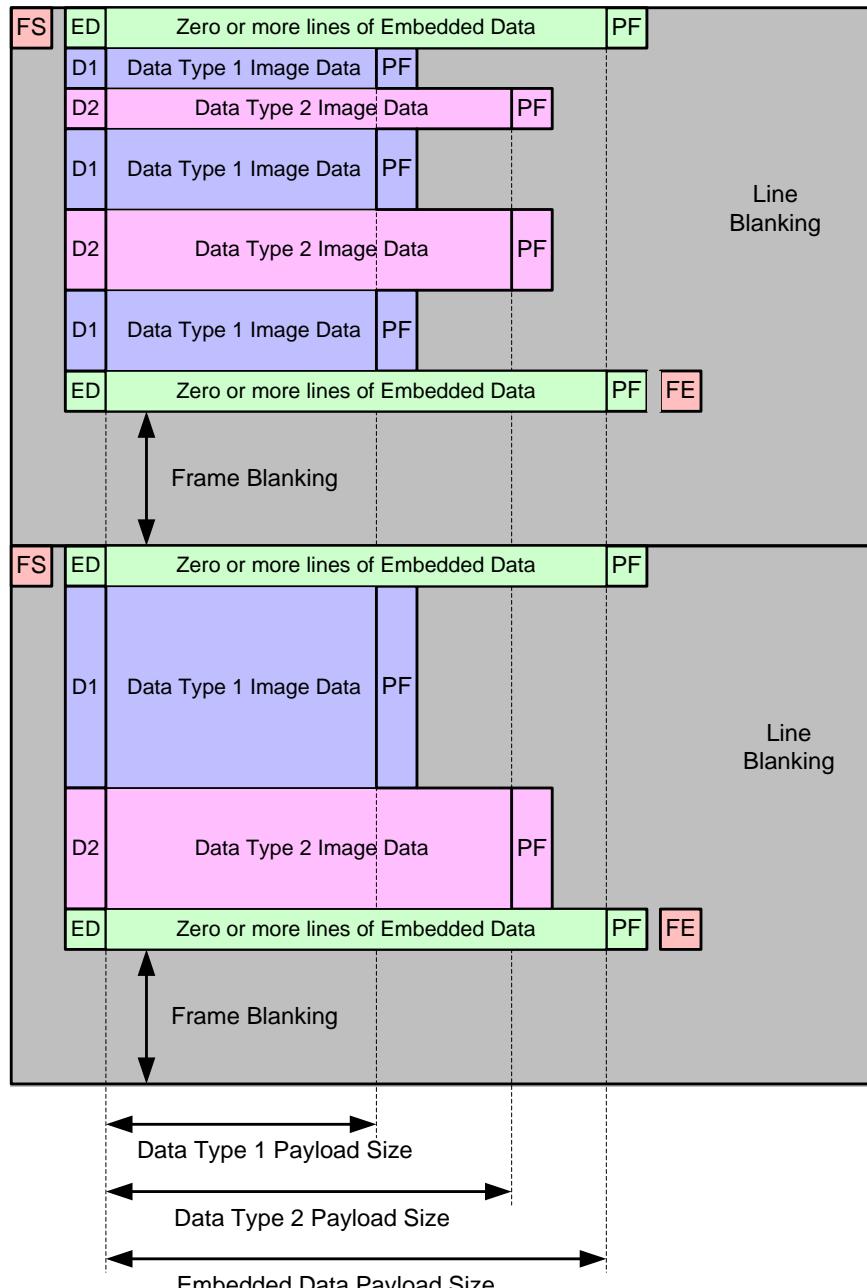


Figure 105 Interleaved Data Transmission using Data Type Value

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

2317 type, between a Frame Start and a Frame End packet within the same virtual channel belong to the same
 2318 frame.

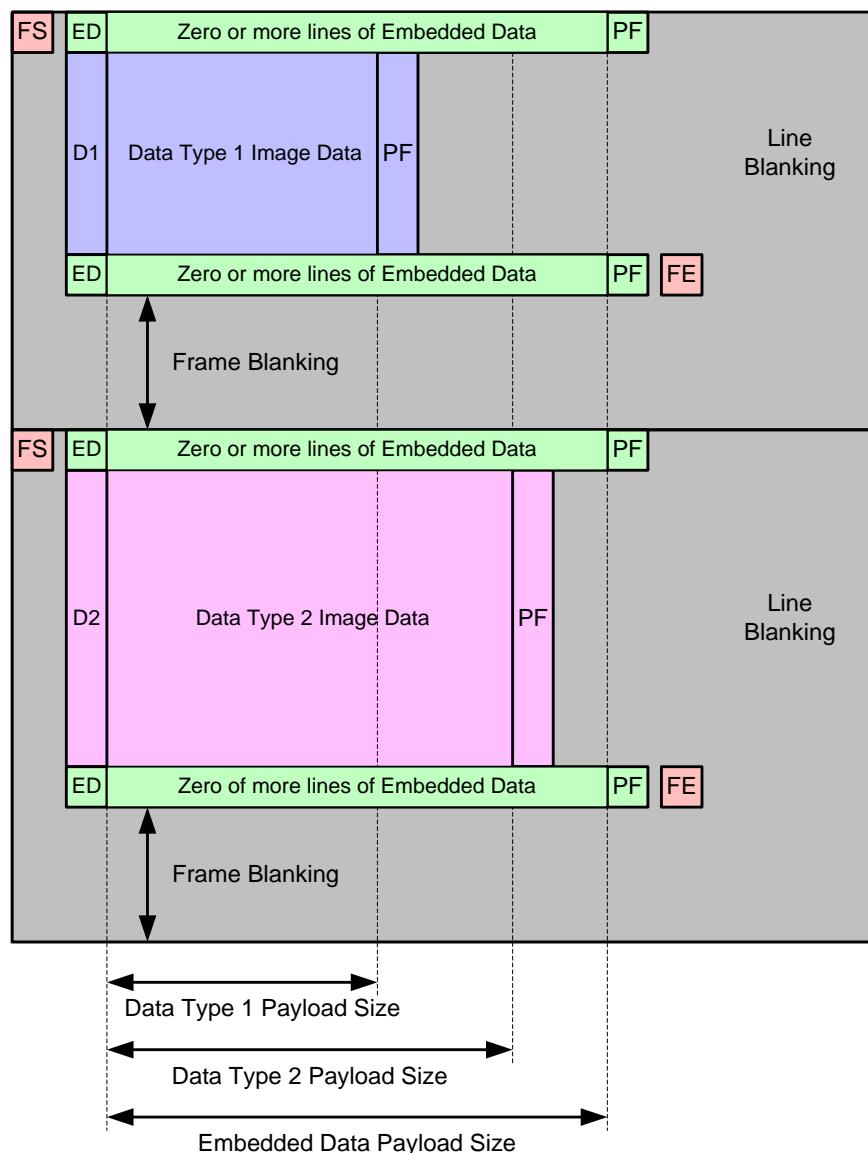
2319 Packets of different data types may be interleaved at either the packet level as illustrated in *Figure 106* or
 2320 the frame level as illustrated in *Figure 107*. Data formats are defined in *Section 11*.

**KEY:**

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

2321

Figure 106 Packet Level Interleaved Data Transmission



2322

Figure 107 Frame Level Interleaved Data Transmission

9.17.2 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 108** illustrates data interleaving using the Virtual Channel Identifier.

Each virtual channel has its own Frame Start and Frame End packet (except for virtual channels used exclusively with USL Packets; see **Section 9.12**). Therefore, it is possible for different virtual channels to have different frame rates, though the data rate for both channels would remain the same.

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

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

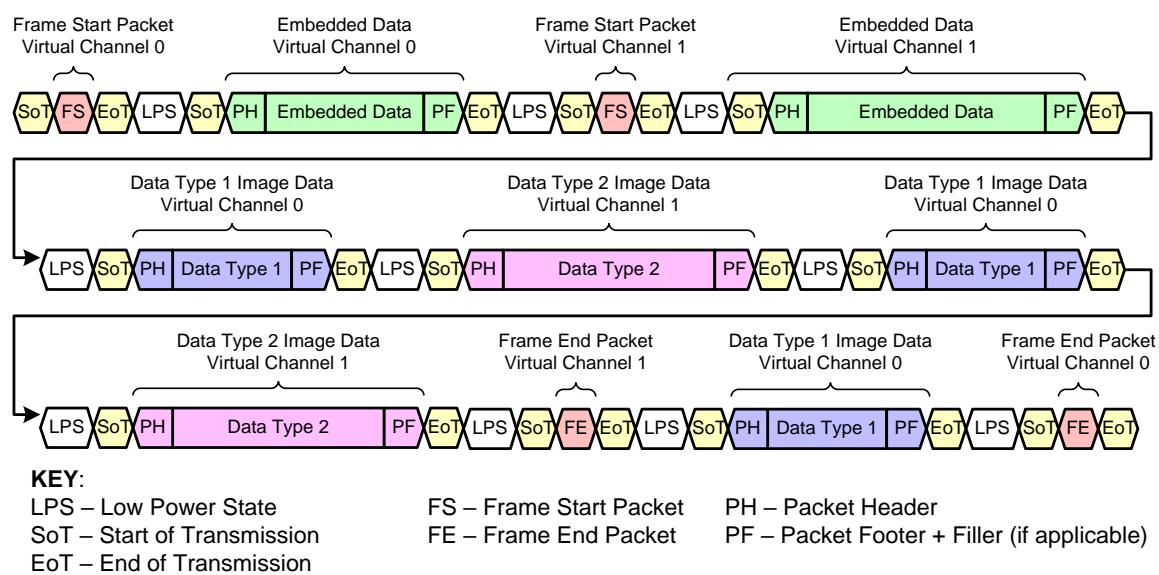


Figure 108 Interleaved Data Transmission using Virtual Channels

This page intentionally left blank.

10 Color Spaces

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

10.1 RGB Color Space Definition

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

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

10.2 YUV Color Space Definition

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

This page intentionally left blank.

11 Data Formats

The intent of this section is to provide a definitive reference for data formats typically used in CSI-2 applications. **Table 38** summarizes the formats, followed by individual definitions for each format. Generic data types not shown in the table are described in **Section 11.1**. For simplicity, all examples are single Lane configurations.

The formats most widely used in CSI-2 applications are distinguished by a “primary” designation in **Table 38**. 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 38 Primary and Secondary Data Formats Definitions

Data Format	Primary	Secondary
YUV420 8-bit (legacy)	–	S
YUV420 8-bit	–	S
YUV420 10-bit	–	S
YUV420 8-bit (CSPS)	–	S
YUV420 10-bit (CSPS)	–	S
YUV422 8-bit	P	–
YUV422 10-bit	–	S
RGB888	P	–
RGB666	–	S
RGB565	P	–
RGB555	–	S
RGB444	–	S
RAW6	–	S
RAW7	–	S
RAW8	P	–
RAW10	P	–
RAW12	–	S
RAW14	–	S
RAW16	–	S
RAW20	–	S
RAW24	–	S
RAW28	–	S
Generic 8-bit Long Packet Data Types	P	–
User Defined Byte-based Data (Note 1)	P	–
USL Packet Data (See Section 9.12)	–	S

Note:

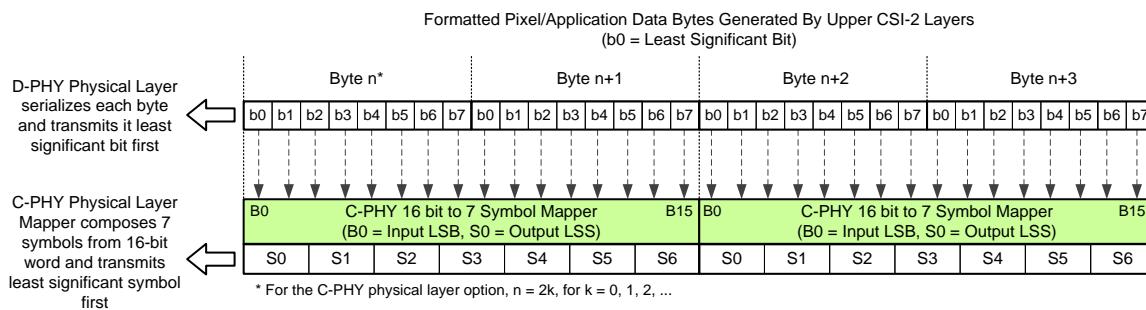
1. Compressed image data should use the user defined, byte-based data type codes

2355 For clarity the Start of Transmission and End of Transmission sequences in the figures in this section have
 2356 been omitted.

2357 The balance of this section details how sequences of pixels and other application data conforming to each of
 2358 the data types listed in **Table 38** are converted into equivalent byte sequences by the CSI-2 Pixel to Byte
 2359 Packing Formats layer shown in **Figure 3**.

2360 Various figures in this section depict these byte sequences as shown at the top of **Figure 109**, where Byte n
 2361 always precedes Byte m for $n < m$. Also note that even though each byte is shown in LSB-first order, this is
 2362 not meant to imply that the bytes themselves are bit-reversed by the Pixel to Byte Packing Formats layer
 2363 prior to output.

2364 For the D-PHY physical layer option, each byte in the sequence is serially transmitted LSB-first, whereas for
 2365 the C-PHY physical layer option, successive byte pairs in the sequence are encoded and then serially
 2366 transmitted LSS-first. **Figure 109** illustrates these options for a single-Lane system.



2367 **Figure 109 Byte Packing Pixel Data to C-PHY Symbol Illustration**

11.1 Generic 8-bit Long Packet Data Types

2368 *Table 39* defines the generic 8-bit Long packet data types.

2369 **Table 39 Generic 8-bit Long Packet Data Types**

Data Type	Description	See Section
0x10	Null	11.1.1
0x11	Blanking Data	
0x12	Embedded 8-bit non Image Data	11.1.2
0x13	Generic long packet data type 1	11.1.3
0x14	Generic long packet data type 2	
0x15	Generic long packet data type 3	
0x16	Generic long packet data type 4	
0x17	Reserved	—

11.1.1 Null and Blanking Data

2370 For both the null and blanking data types the receiver must ignore the content of the packet payload data.

2371 A blanking packet differs from a null packet in terms of its significance within a video data stream. A null
2372 packet has no meaning whereas the blanking packet may be used, for example, as the blanking lines between
2373 frames in an ITU-R BT.656 style video stream.

11.1.2 Embedded Information

It is possible to embed extra lines containing additional information to the beginning and to the end of each picture frame as presented in the **Figure 110**. If embedded information exists, then the lines containing the embedded data must use the embedded data code in the data identifier.

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

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

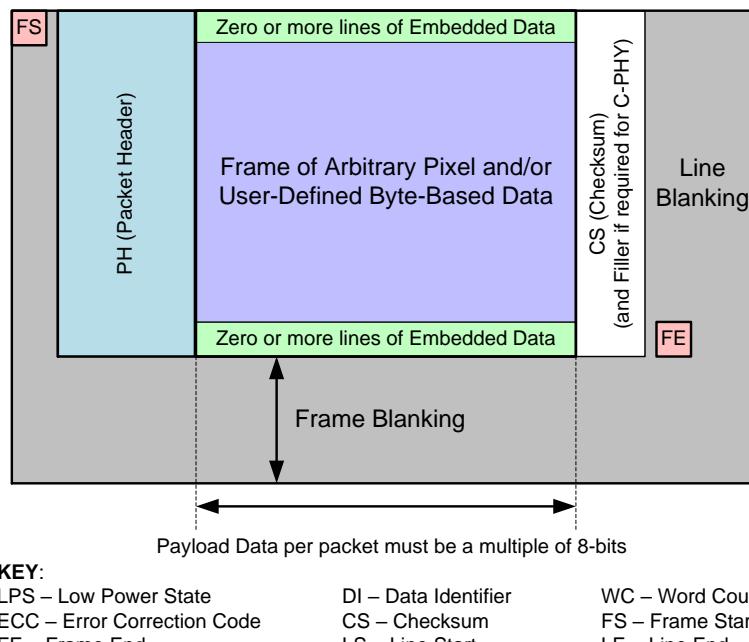


Figure 110 Frame Structure with Embedded Data at the Beginning and End of the Frame

11.1.3 Generic Long Packet Data Types 1 Through 4

These codes have no specific definitions and may be used, for example, to identify various types of vendor-specific metadata packets transmitted within an image frame.

11.2 YUV Image Data

Table 40 defines the data type codes for YUV data formats described in this section. The number of lines transmitted for the YUV420 data type shall be even.

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

Table 40 YUV Image Data Types

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

11.2.1 Legacy YUV420 8-bit

Legacy YUV420 8-bit data transmission is performed by transmitting UYY... / VYY... sequences in odd / even lines. U component is transferred in odd lines (1, 3, 5 ...) and V component is transferred in even lines (2, 4, 6 ...). This sequence is illustrated in *Figure 111*.

Table 41 specifies the packet size constraints for YUV420 8-bit packets. Each packet must be a multiple of the values in the table.

Table 41 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 112*.

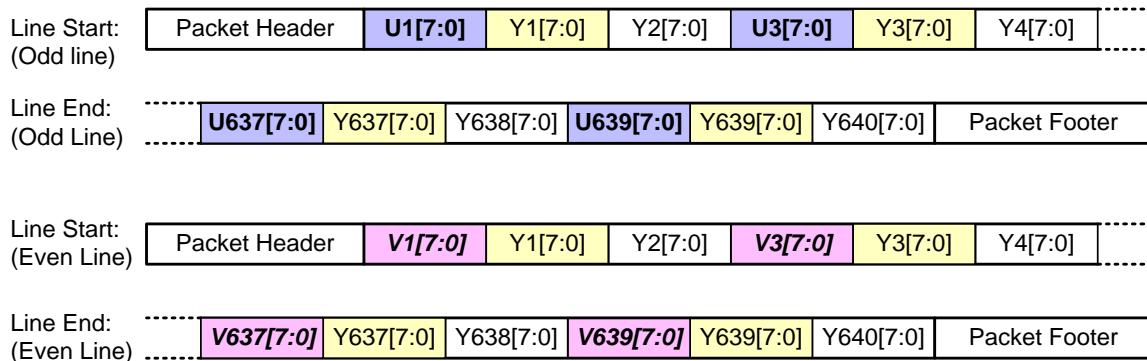


Figure 111 Legacy YUV420 8-bit Transmission

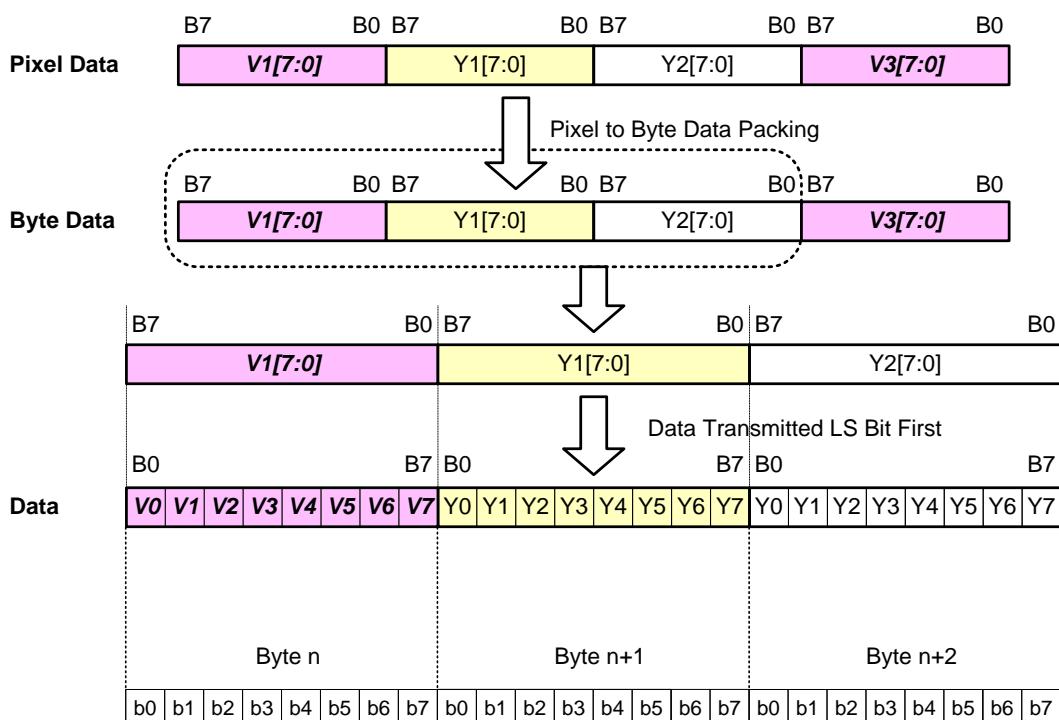
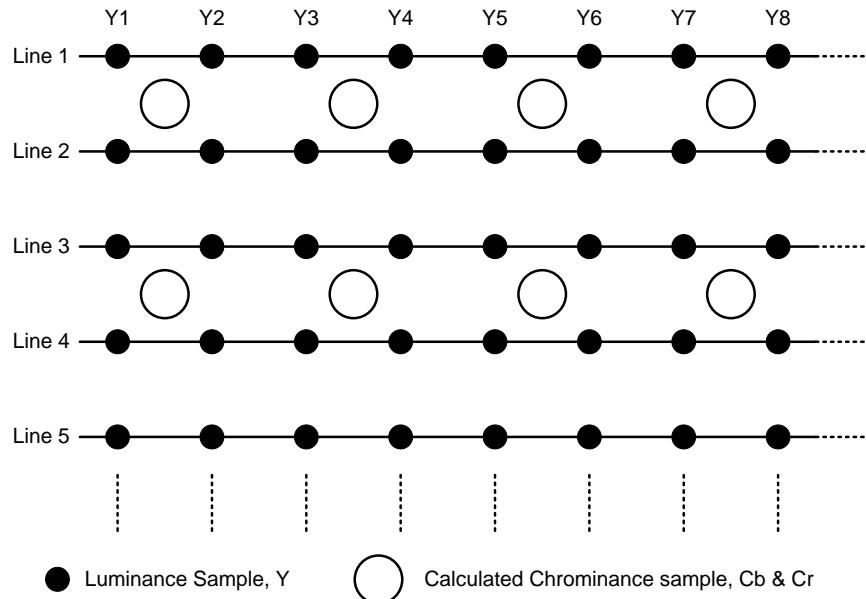


Figure 112 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

2400 There is one spatial sampling option

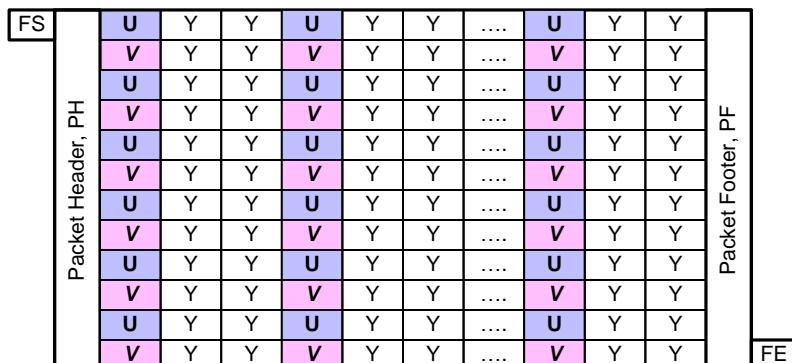
- 2401 • H.261, H.263 and MPEG1 Spatial Sampling (*Figure 113*).



2402

Figure 113 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1

2403

**Figure 114 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 115*.

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

Table 42 specifies the packet size constraints for YUV420 8-bit packets. Each packet must be a multiple of the values in the table.

Table 42 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 116*.

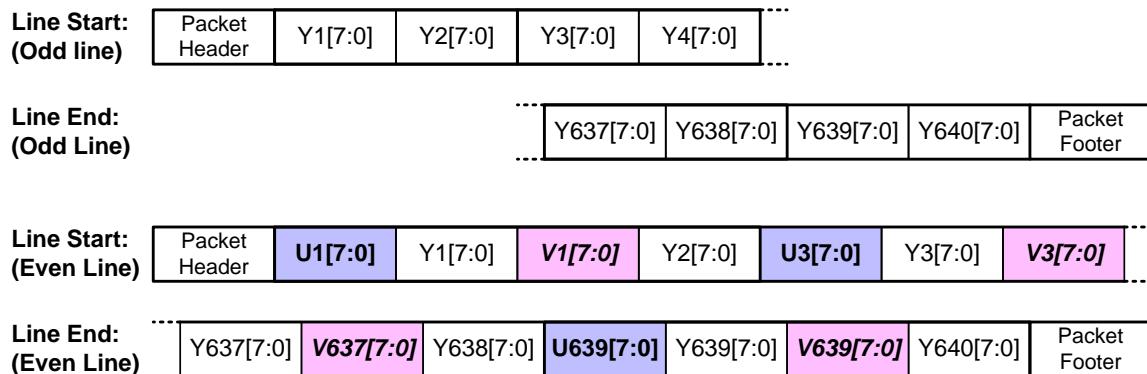
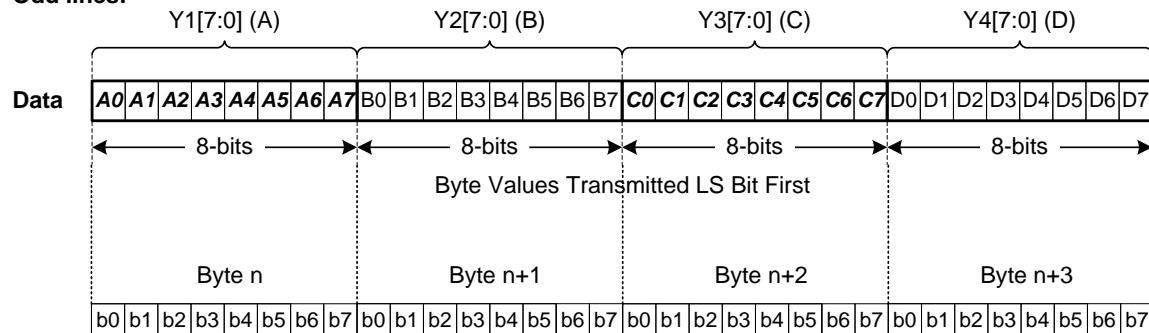
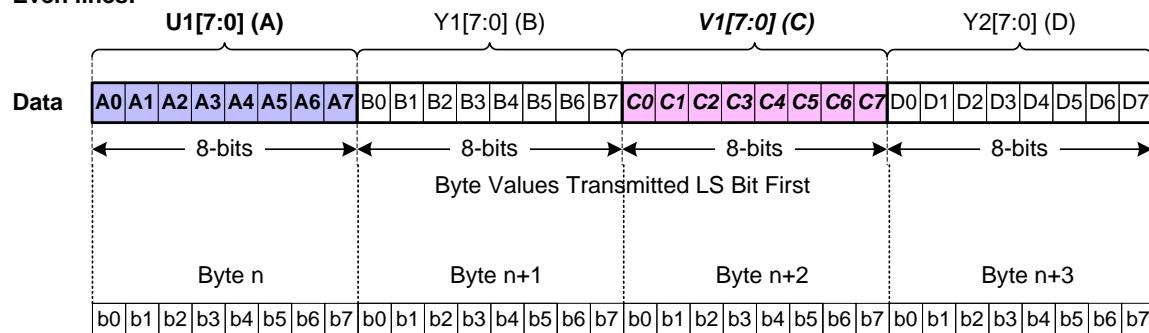


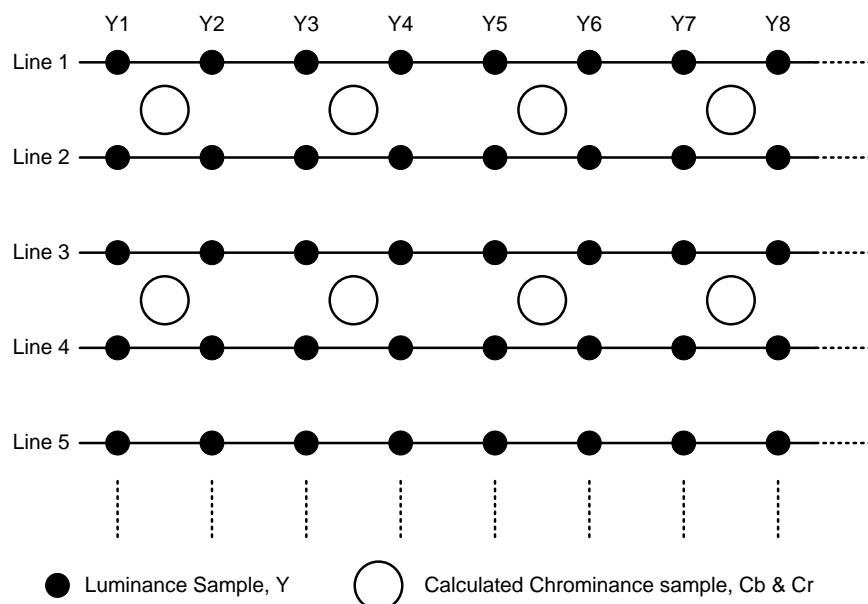
Figure 115 YUV420 8-bit Data Transmission Sequence

Odd lines:**Even lines:****Figure 116 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration**

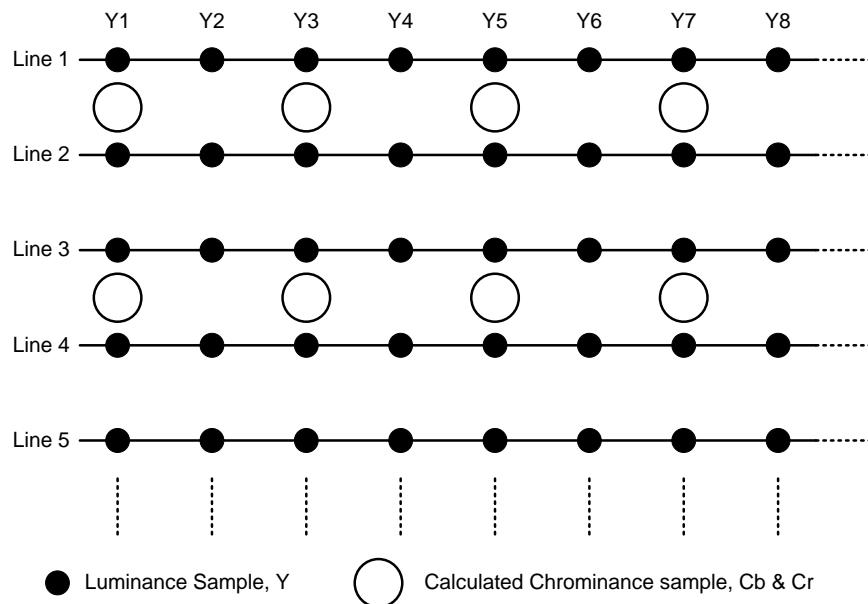
2418 There are two spatial sampling options

- 2419 • H.261, H.263 and MPEG1 Spatial Sampling (**Figure 117**).
- 2420 • Chroma Shifted Pixel Sampling (CSPS) for MPEG2, MPEG4 (**Figure 118**).

2421 **Figure 119** shows the YUV420 frame format.



2422

Figure 117 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1

2423

Figure 118 YUV420 Spatial Sampling for MPEG 2 and MPEG 4

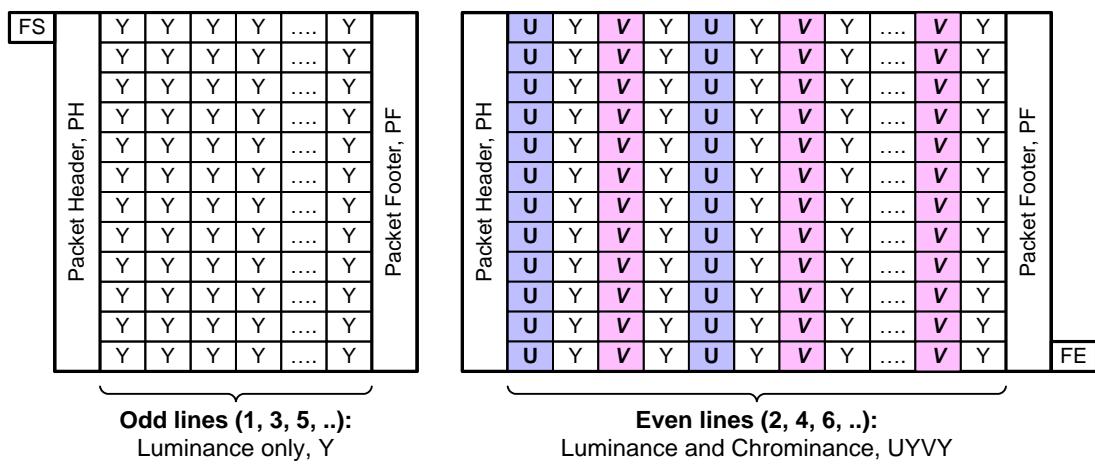


Figure 119 YUV420 8-bit Frame Format

2424

11.2.3 YUV420 10-bit

YUV420 10-bit data transmission is performed by transmitting YYYY... / UYVVUYVY... sequences in odd / even lines. Only the luminance component (Y) is transferred in odd lines (1, 3, 5...) and both luminance (Y) and chrominance (U and V) components transferred in even lines (2, 4, 6...). The format for the even lines (UYVY) is identical to the YUV422 –10-bit data format. The sequence is illustrated in *Figure 120*.

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

Table 43 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 43 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 121*.

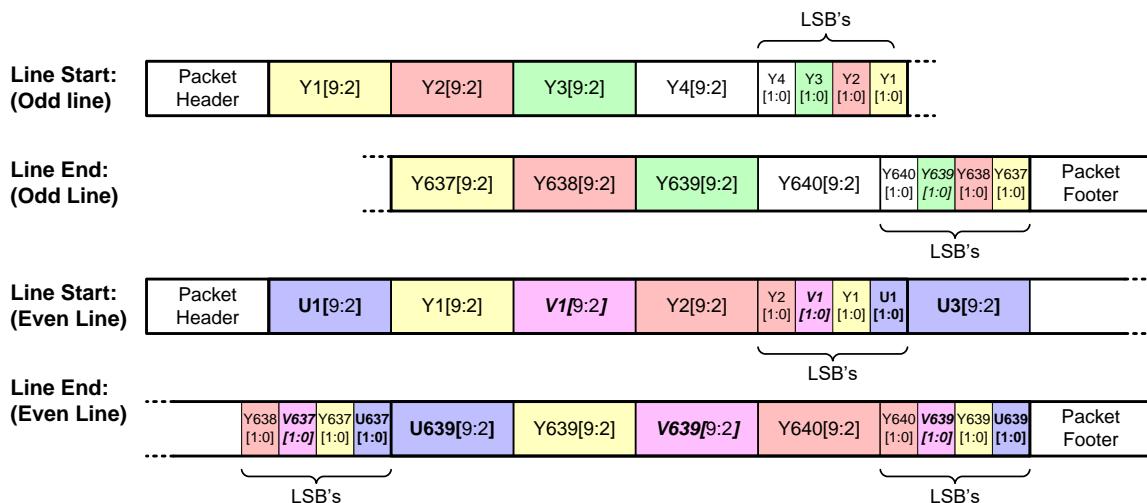
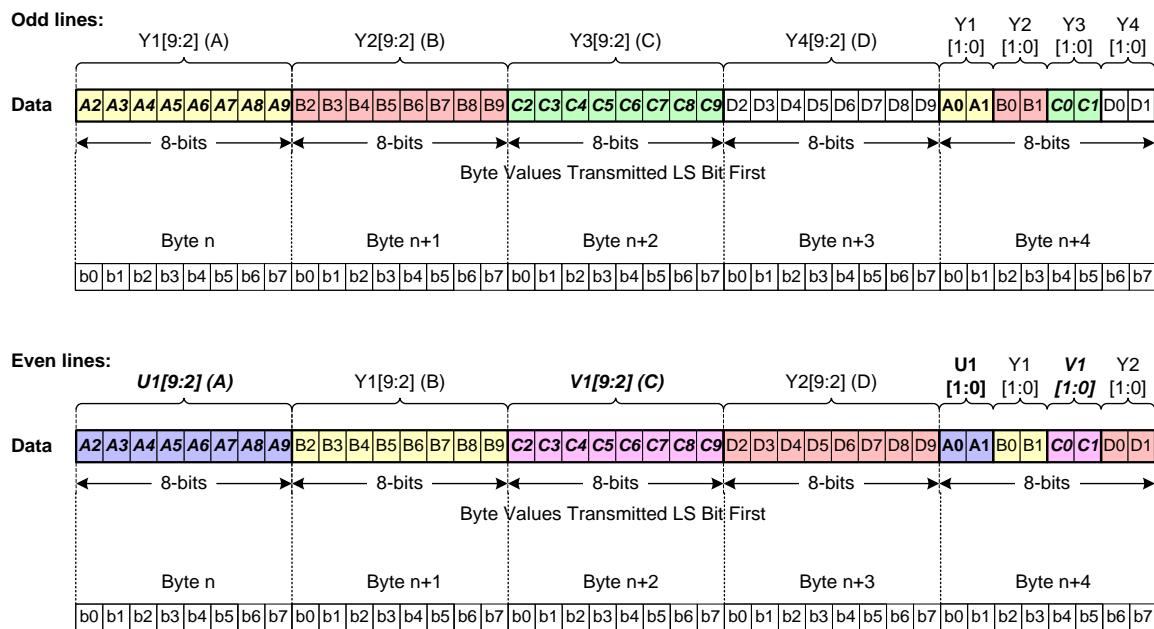


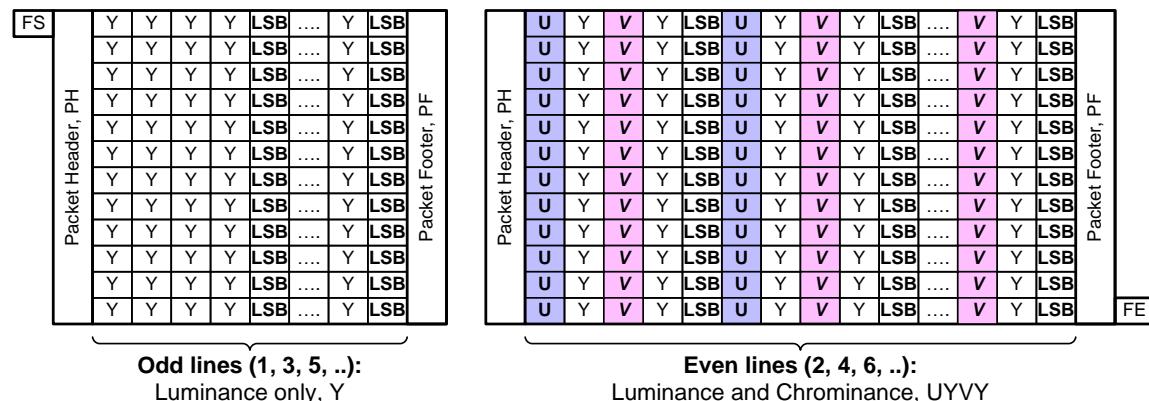
Figure 120 YUV420 10-bit Transmission

**Figure 121 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration**

2437

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

2439

**Figure 122 YUV420 10-bit Frame Format**

11.2.4 YUV422 8-bit

YUV422 8-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in *Figure 123*.

Table 44 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 44 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 124*.

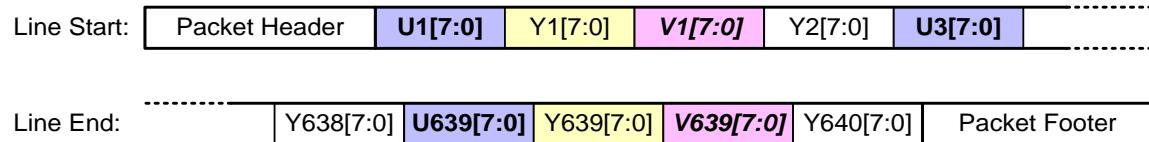


Figure 123 YUV422 8-bit Transmission

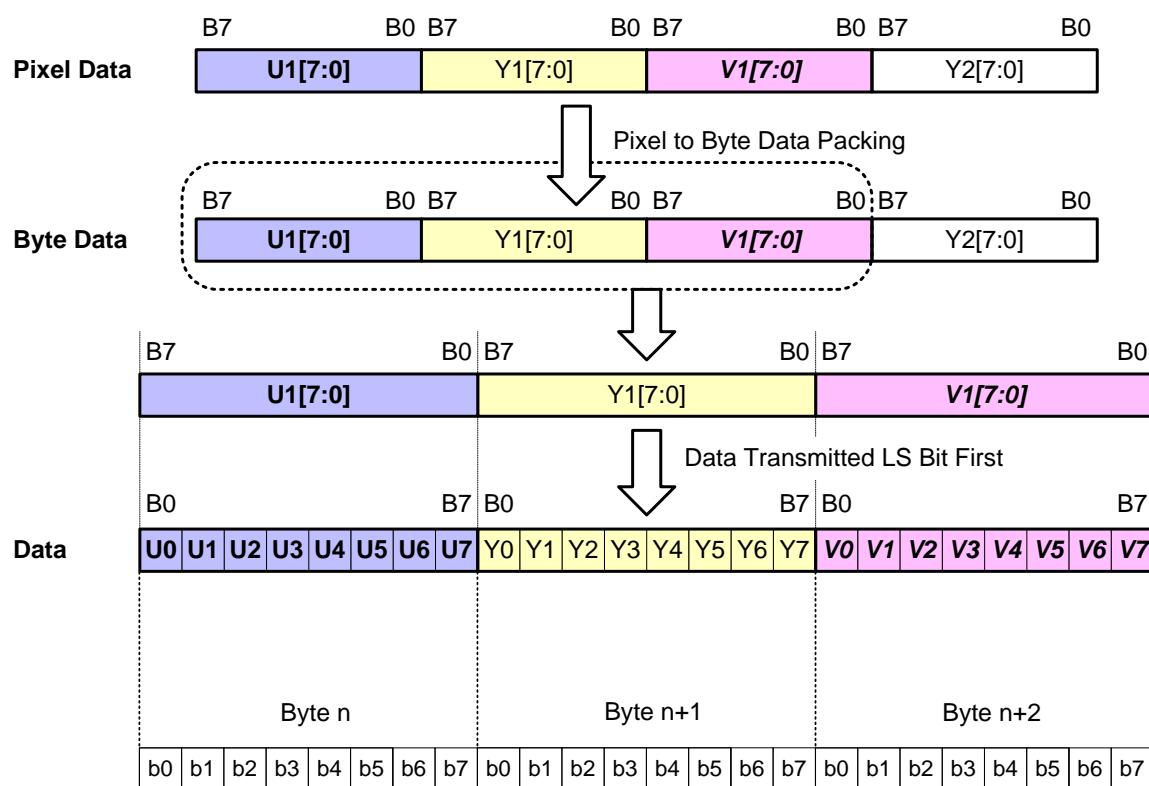
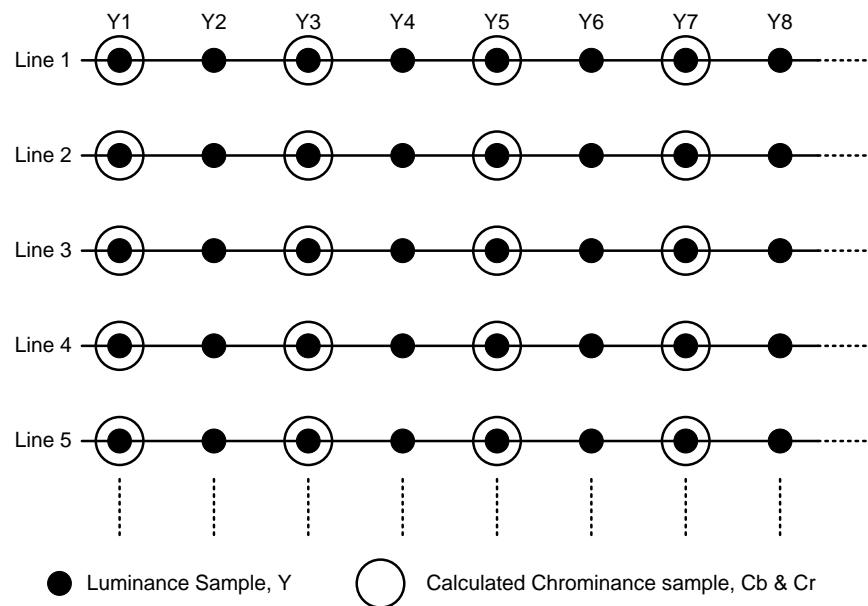


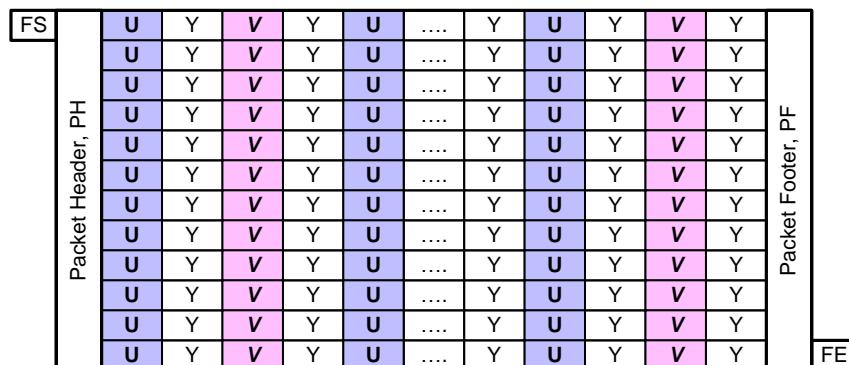
Figure 124 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration



2449

Figure 125 YUV422 Co-sited Spatial Sampling2450
2451

The pixel spatial alignment is the same as in CCIR-656 standard. The frame format for YUV422 is presented in **Figure 126**.



2452

Figure 126 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 127.

Table 45 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 45 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 128*.

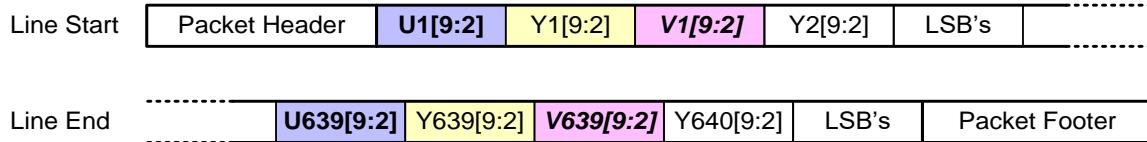
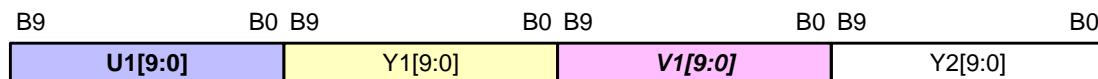


Figure 127 YUV422 10-bit Transmitted Bytes

Pixel Data:



Byte Data:

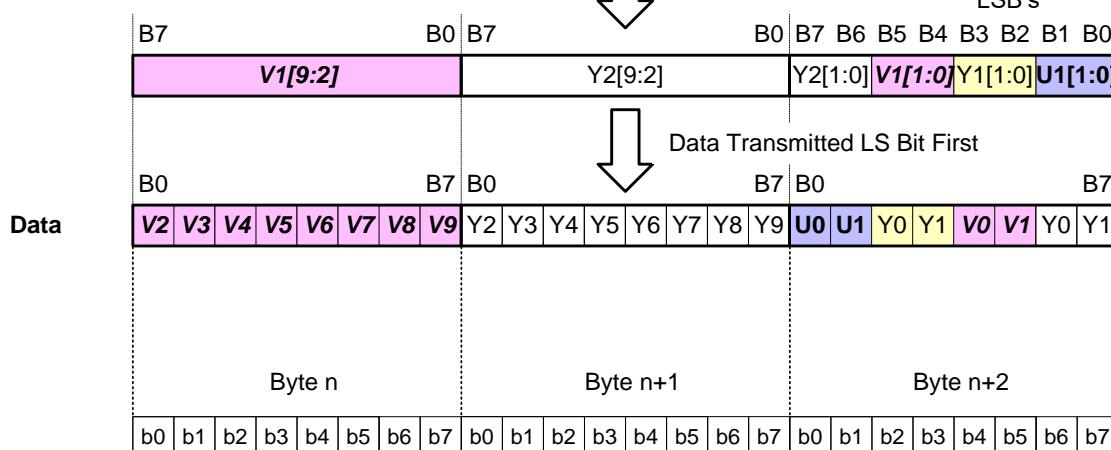
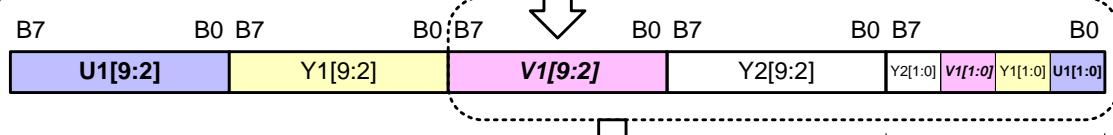


Figure 128 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration

The pixel spatial alignment is the same as in the YUV422 8-bit data case. The frame format for YUV422 is presented in *Figure 129*.

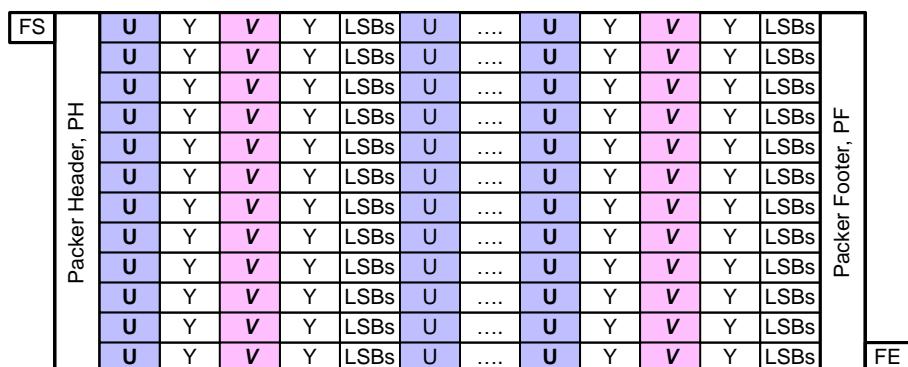


Figure 129 YUV422 10-bit Frame Format

2464

11.3 RGB Image Data

2465

Table 46 defines the data type codes for RGB data formats described in this section.

2466

Table 46 RGB Image Data Types

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

11.3.1 RGB888

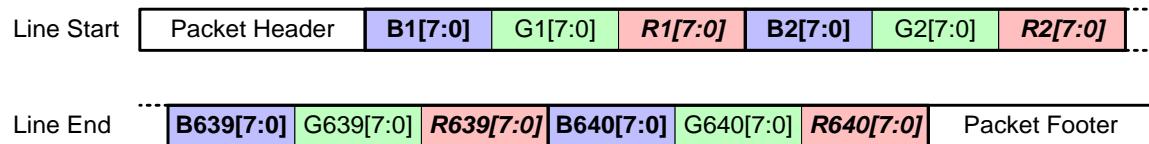
2467
2468
RGB888 data transmission is performed by transmitting a BGR byte sequence. This sequence is illustrated
in *Figure 130*. The RGB888 frame format is illustrated in *Figure 132*.

2469
2470
Table 47 specifies the packet size constraints for RGB888 packets. The length of each packet must be a
multiple of the values in the table.

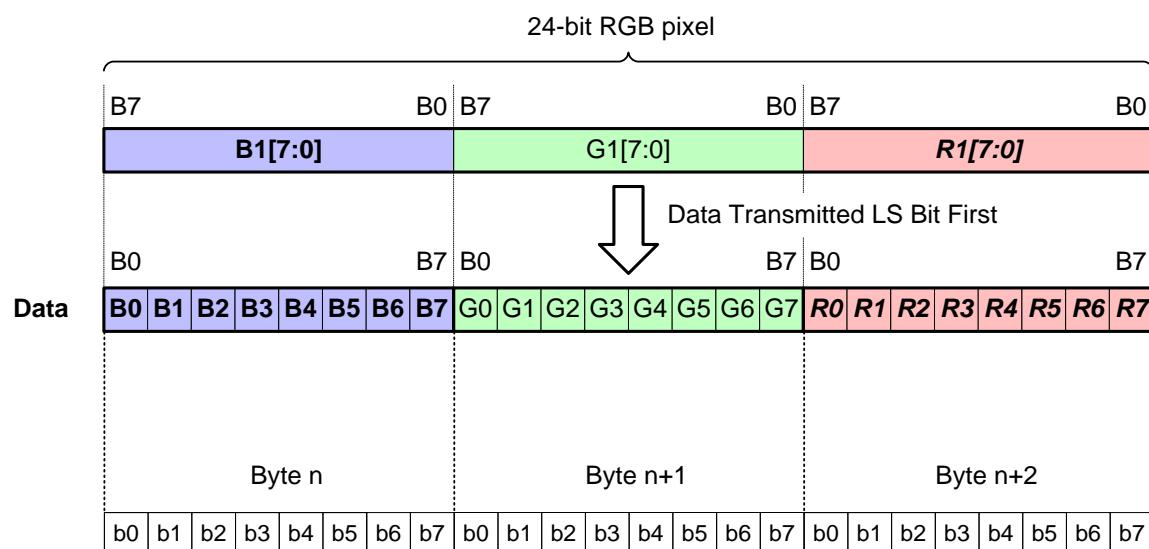
2471 **Table 47 RGB888 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	3	24

2472 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
2473 in *Figure 131*.



2474 **Figure 130 RGB888 Transmission**



2475 **Figure 131 RGB888 Transmission in CSI-2 Bus Bitwise Illustration**

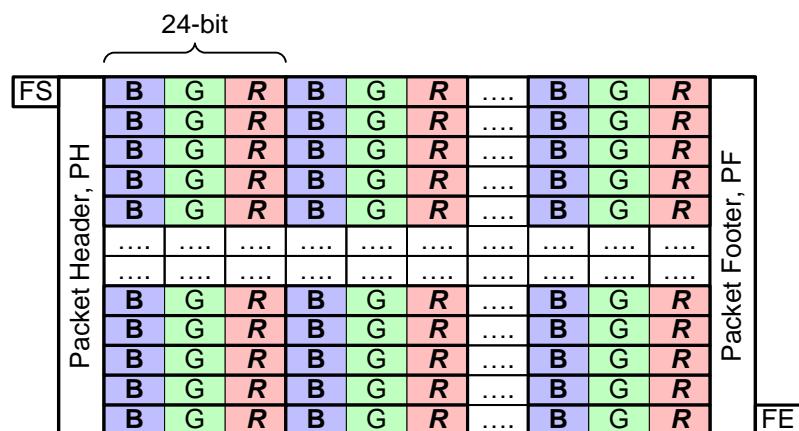


Figure 132 RGB888 Frame Format

2476

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 133*. The frame format for RGB666 is presented in the *Figure 135*.

Table 48 specifies the packet size constraints for RGB666 packets. The length of each packet must be a multiple of the values in the table.

Table 48 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 134*.

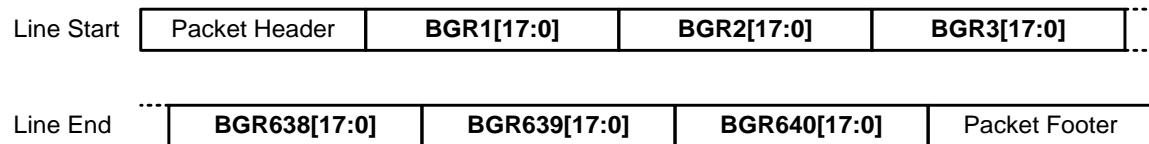


Figure 133 RGB666 Transmission with 18-bit BGR Words

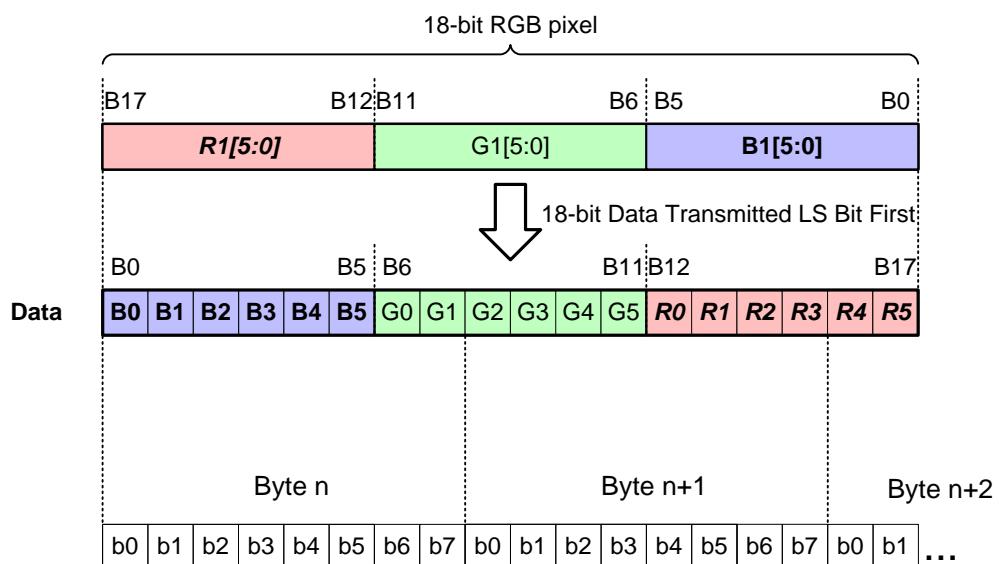
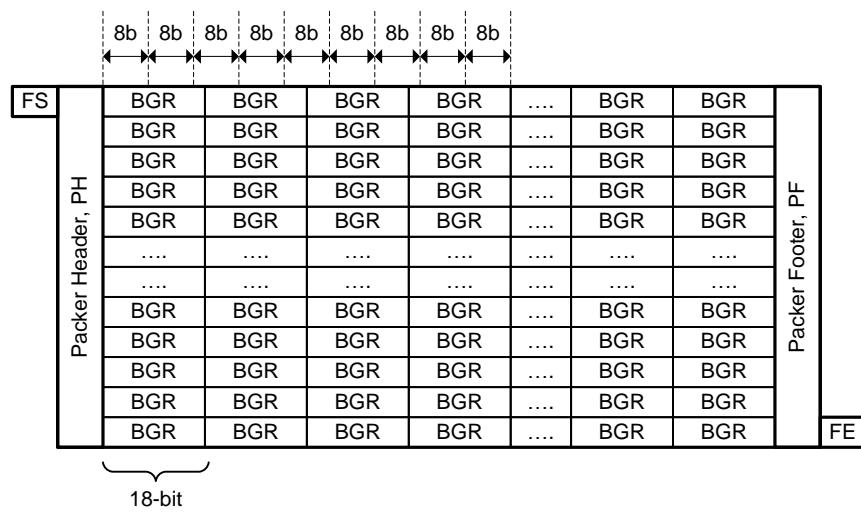


Figure 134 RGB666 Transmission on CSI-2 Bus Bitwise Illustration



2487

Figure 135 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 136*. The frame format for RGB565 is presented in the *Figure 138*.

Table 49 specifies the packet size constraints for RGB565 packets. The length of each packet must be a multiple of the values in the table.

Table 49 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 137*.

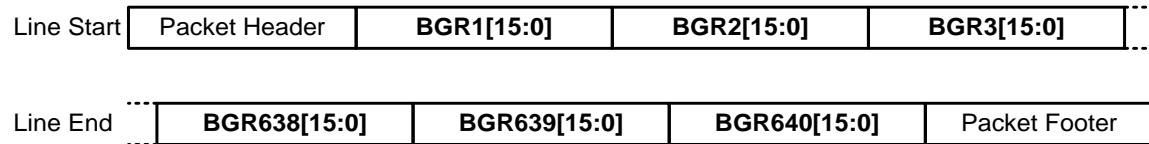


Figure 136 RGB565 Transmission with 16-bit BGR Words

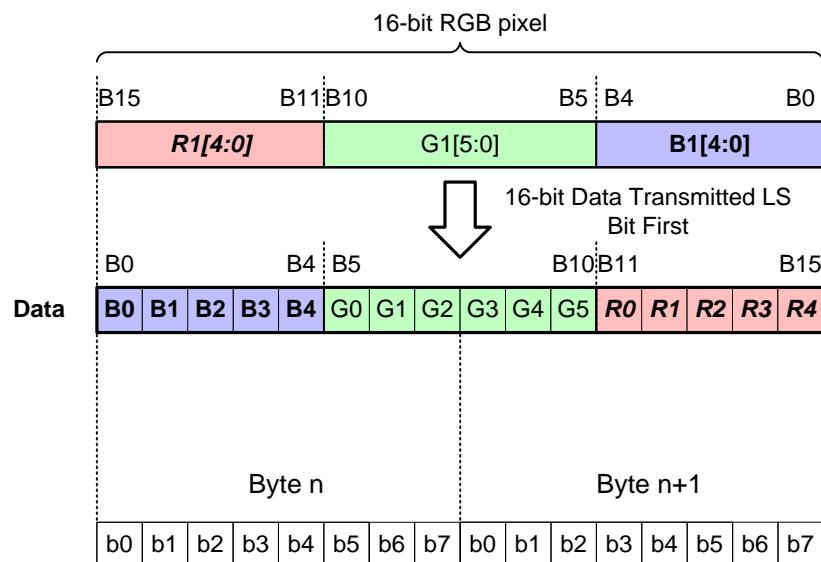


Figure 137 RGB565 Transmission on CSI-2 Bus Bitwise Illustration

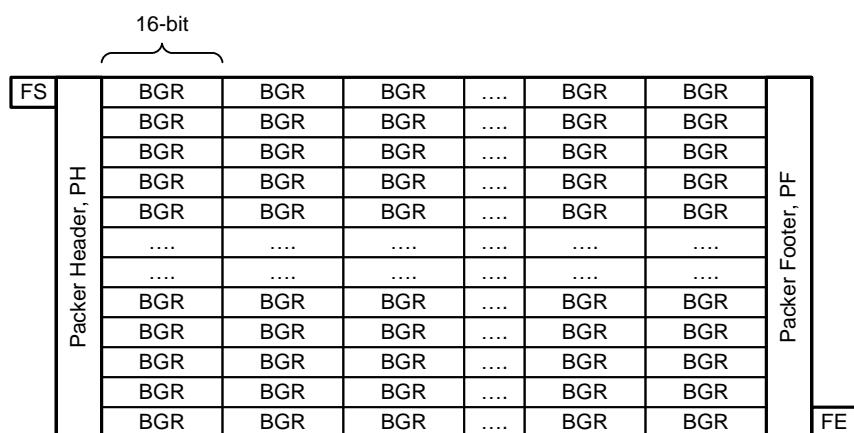


Figure 138 RGB565 Frame Format

2499

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 139**.

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 139**.

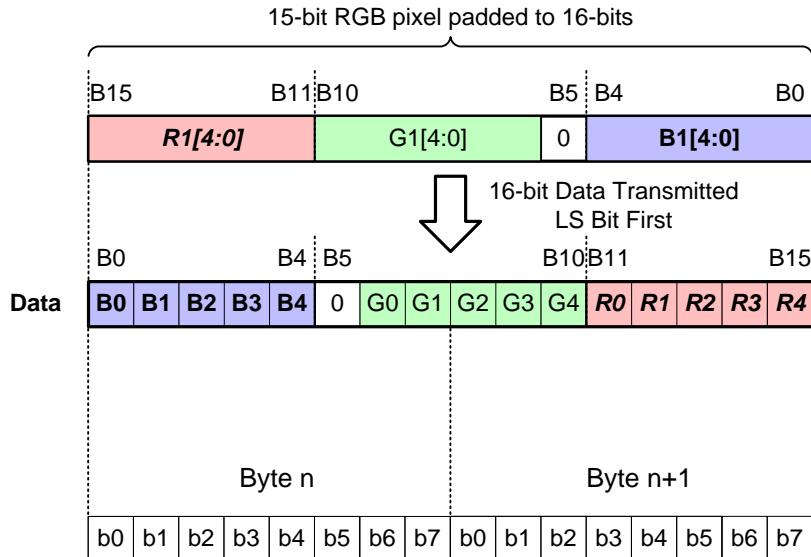


Figure 139 RGB555 Transmission on CSI-2 Bus Bitwise Illustration

11.3.5 RGB444

RGB444 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB444 data should be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs of each color component as illustrated in **Figure 140**.

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

Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB444 case the length of one data word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words; i.e. instead of flipping each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in **Figure 140**.

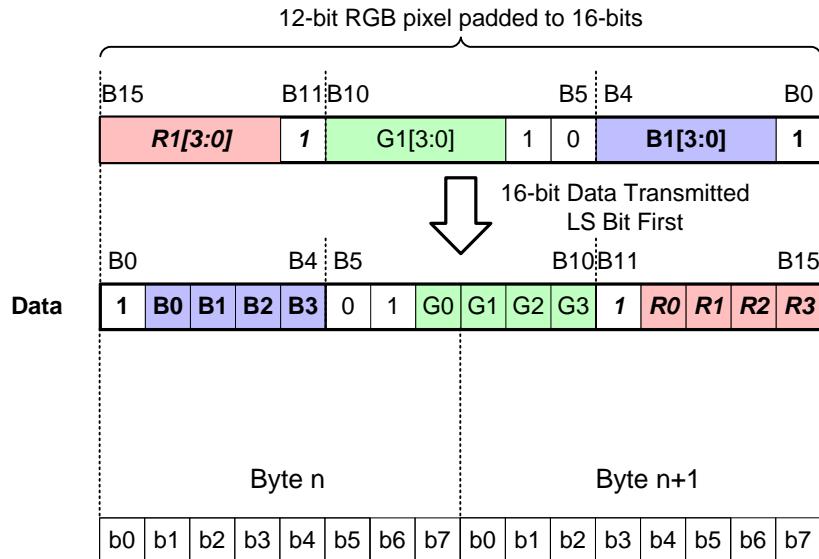


Figure 140 RGB444 Transmission on CSI-2 Bus Bitwise Illustration

11.4 RAW Image Data

The RAW 6/7/8/10/12/14/16/20/24/28 modes are used for transmitting Raw image data from the image sensor.

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

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

Table 50 defines the data type codes for RAW data formats described in this section.

Table 50 RAW Image Data Types

Data Type	Description
0x26	RAW28
0x27	RAW24
0x28	RAW6
0x29	RAW7
0x2A	RAW8
0x2B	RAW10
0x2C	RAW12
0x2D	RAW14
0x2E	RAW16
0x2F	RAW20

11.4.1 RAW6

The 6-bit Raw data transmission is done by transmitting the pixel data over CSI-2 bus. This sequence is illustrated in *Figure 141* (VGA case). *Table 51* specifies the packet size constraints for RAW6 packets. The length of each packet must be a multiple of the values in the table.

Table 51 RAW6 Packet Data Size Constraints

Pixels	Bytes	Bits
4	3	24

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

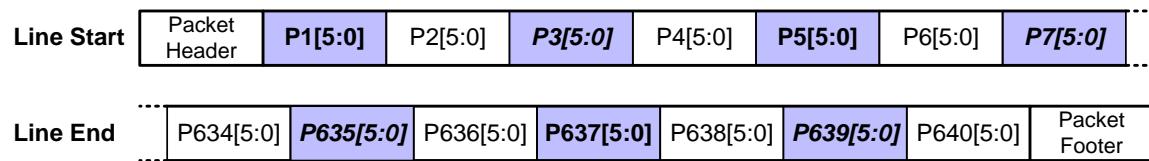


Figure 141 RAW6 Transmission

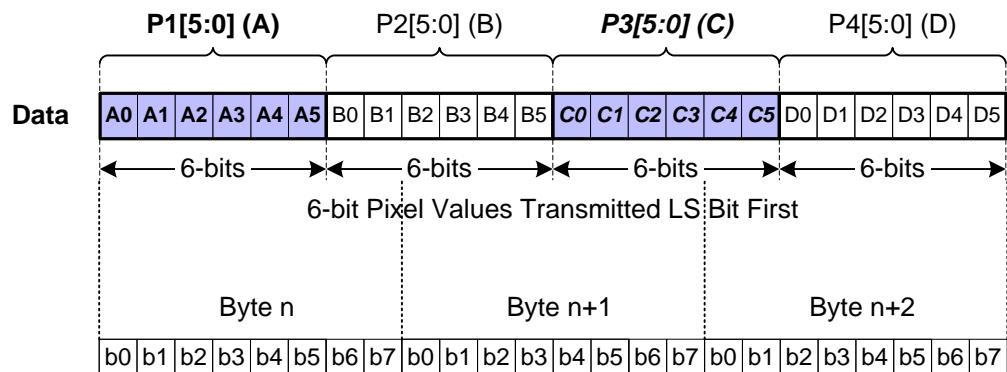


Figure 142 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration

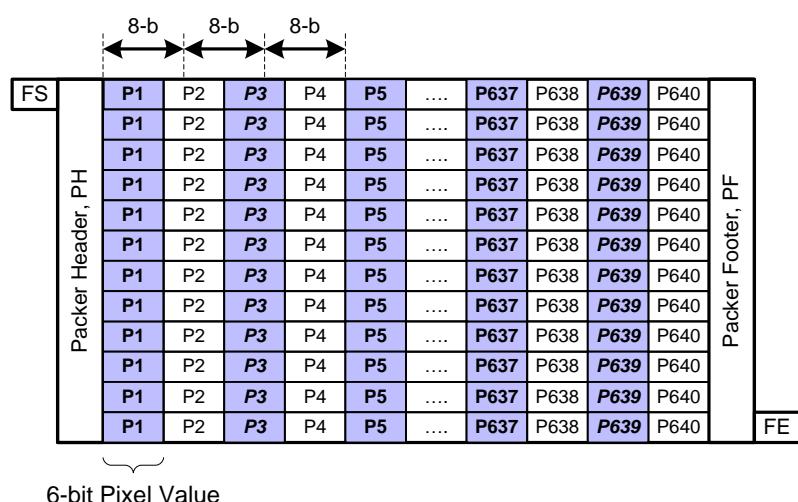


Figure 143 RAW6 Frame Format

11.4.2 RAW7

The 7-bit Raw data transmission is done by transmitting the pixel data over CSI-2 bus. This sequence is illustrated in *Figure 144* (VGA case). *Table 52* specifies the packet size constraints for RAW7 packets. The length of each packet must be a multiple of the values in the table.

Table 52 RAW7 Packet Data Size Constraints

Pixels	Bytes	Bits
8	7	56

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

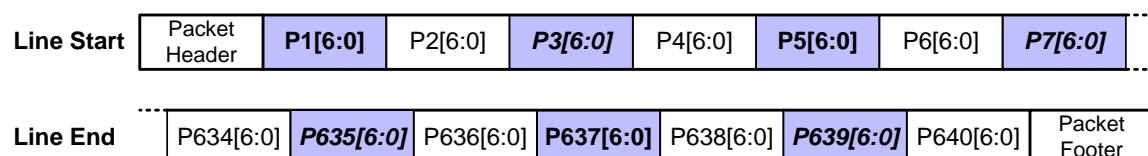


Figure 144 RAW7 Transmission

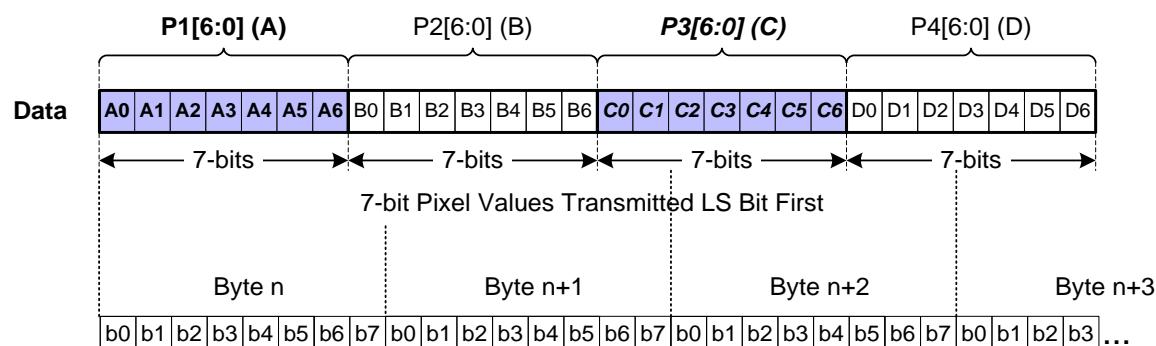


Figure 145 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration

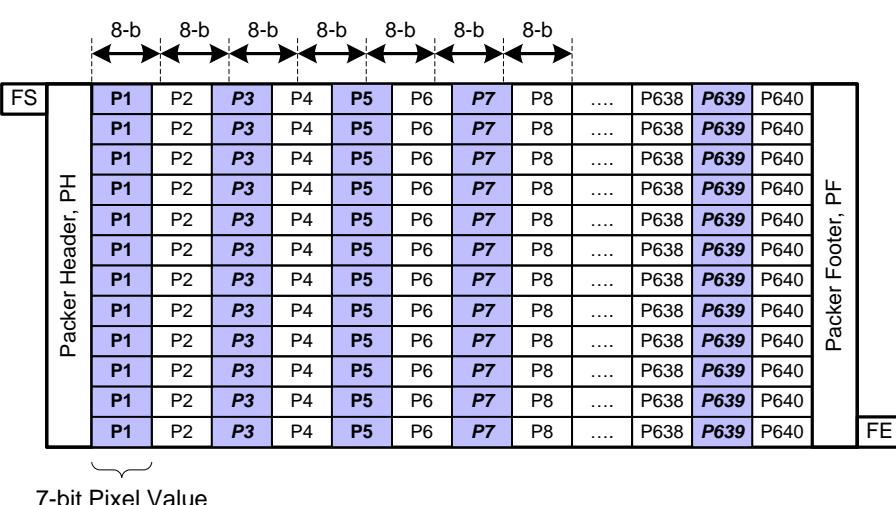


Figure 146 RAW7 Frame Format

11.4.3 RAW8

The 8-bit Raw data transmission is done by transmitting the pixel data over a CSI-2 bus. **Table 53** specifies the packet size constraints for RAW8 packets. The length of each packet must be a multiple of the values in the table.

Table 53 RAW8 Packet Data Size Constraints

Pixels	Bytes	Bits
1	1	8

This sequence is illustrated in **Figure 147** (VGA case).

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

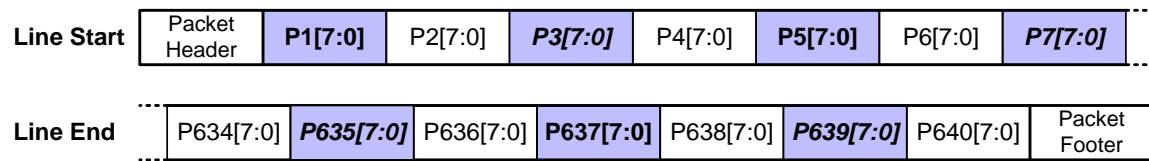


Figure 147 RAW8 Transmission

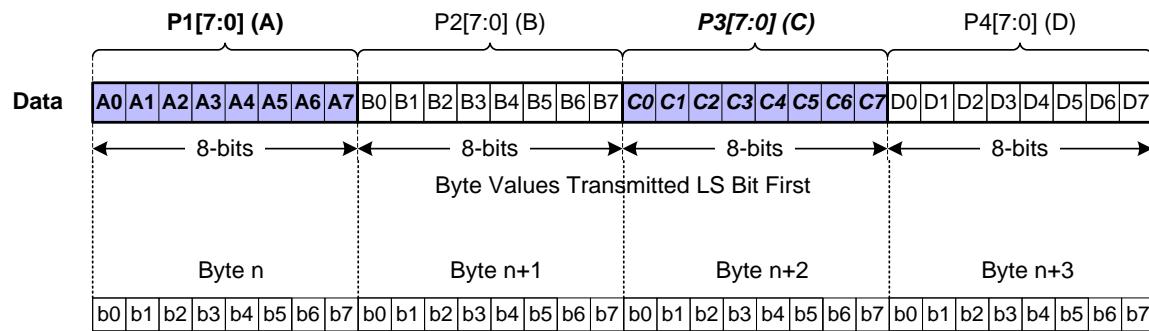


Figure 148 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration

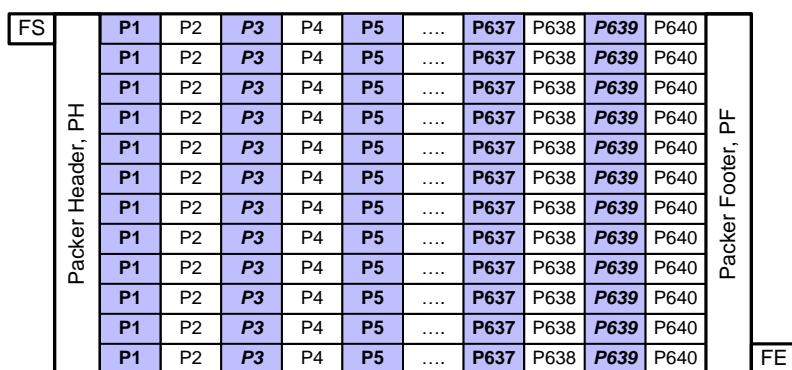


Figure 149 RAW8 Frame Format

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

Table 54 RAW10 Packet Data Size Constraints

Pixels	Bytes	Bits
4	5	40

This sequence is illustrated in **Figure 150** (VGA case).

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

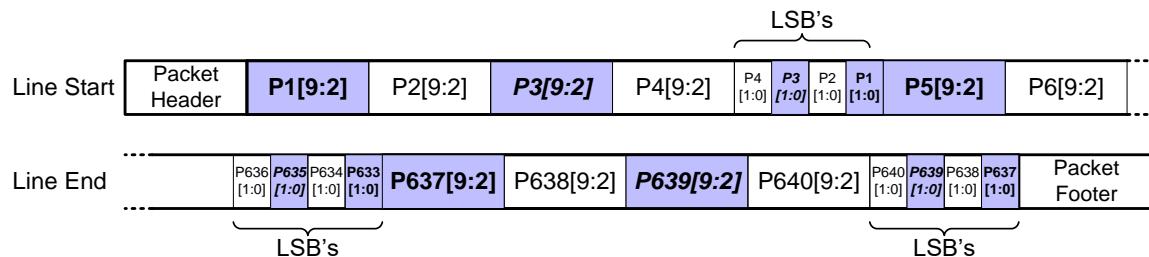


Figure 150 RAW10 Transmission

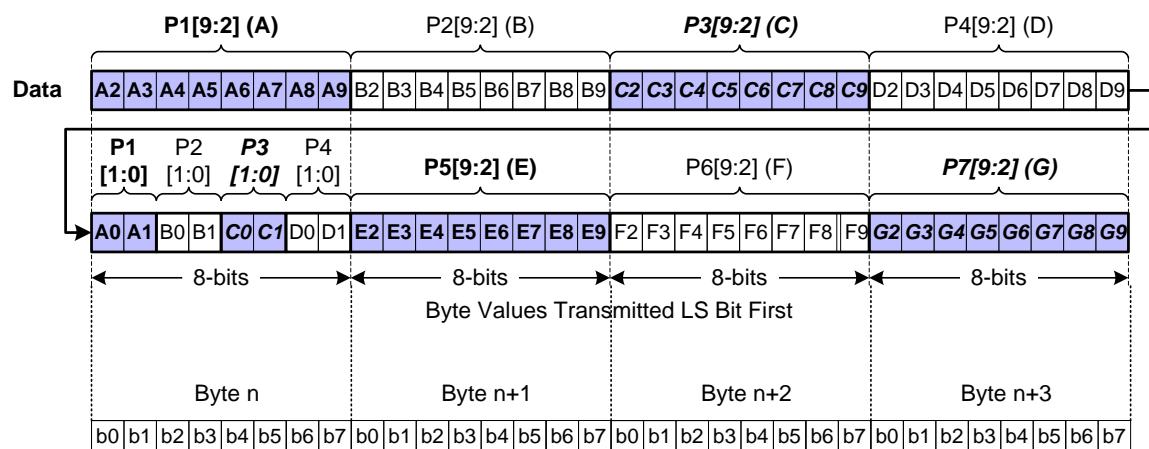
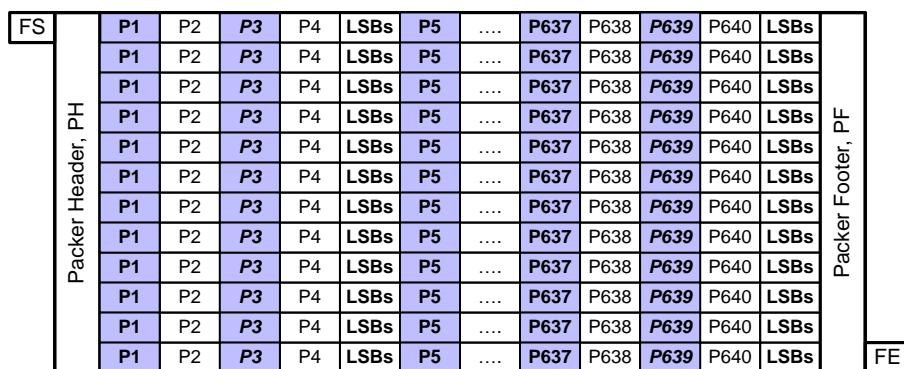


Figure 151 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration

2558

**Figure 152 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 55** specifies the packet size constraints for RAW12 packets. The length of each packet must be a multiple of the values in the table.

Table 55 RAW12 Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

This sequence is illustrated in **Figure 153** (VGA case).

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

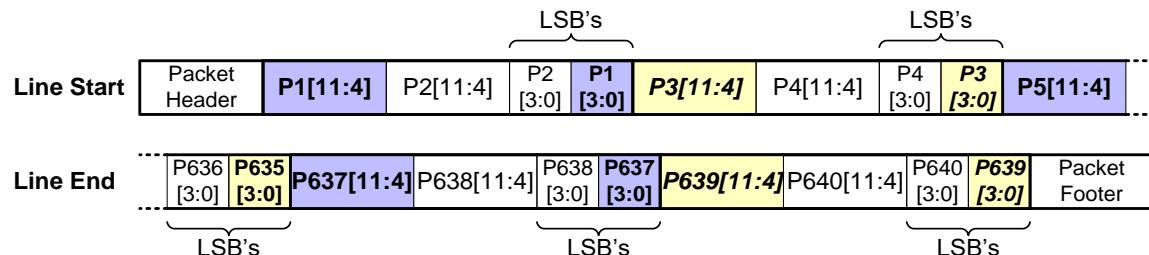


Figure 153 RAW12 Transmission

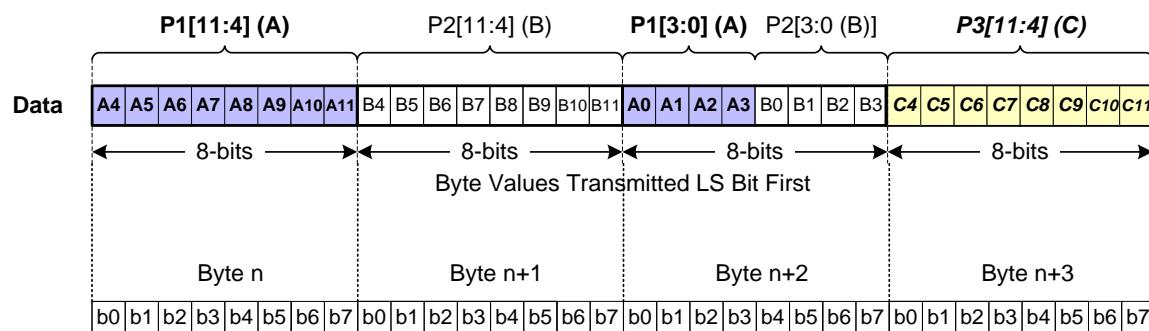


Figure 154 RAW12 Transmission on CSI-2 Bus Bitwise Illustration

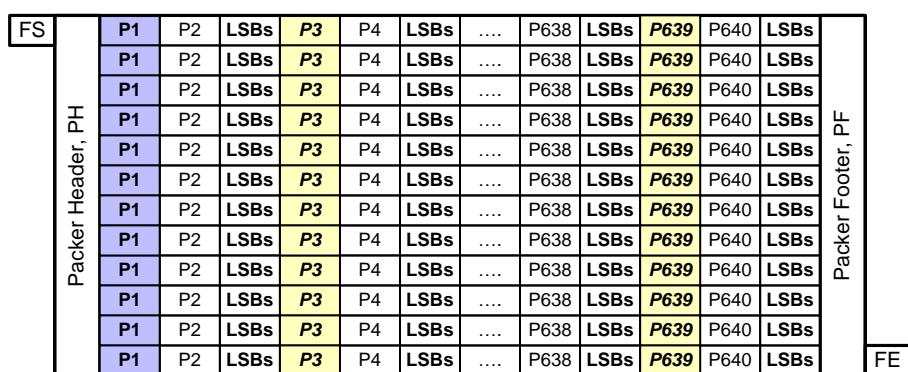


Figure 155 RAW12 Frame Format

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

Table 56 RAW14 Packet Data Size Constraints

Pixels	Bytes	Bits
4	7	56

The sequence is illustrated in **Figure 156** (VGA case).

The LS bits for P1, P2, P3, and P4 are distributed in three bytes as shown in **Figure 156** and **Figure 157**. The same is true for the LS bits for P637, P638, P639, and P640. The bit order during byte transmission follows the general CSI-2 rule, i.e. LSB first.

Note:

Figure 156 has been modified relative to the figures shown in the CSI-2 Specification version 2.0 and earlier, in order to more clearly correspond with **Figure 157**. The RAW14 byte packing and transmission formats themselves have not changed relative to earlier CSI-2 Specification versions.

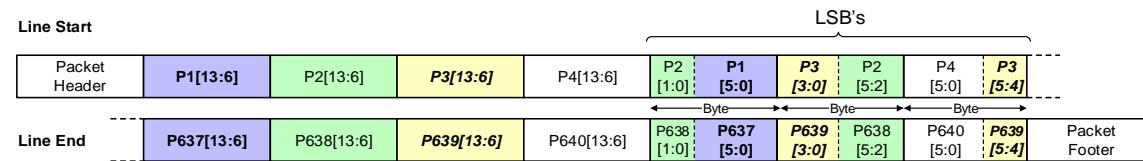


Figure 156 RAW14 Transmission

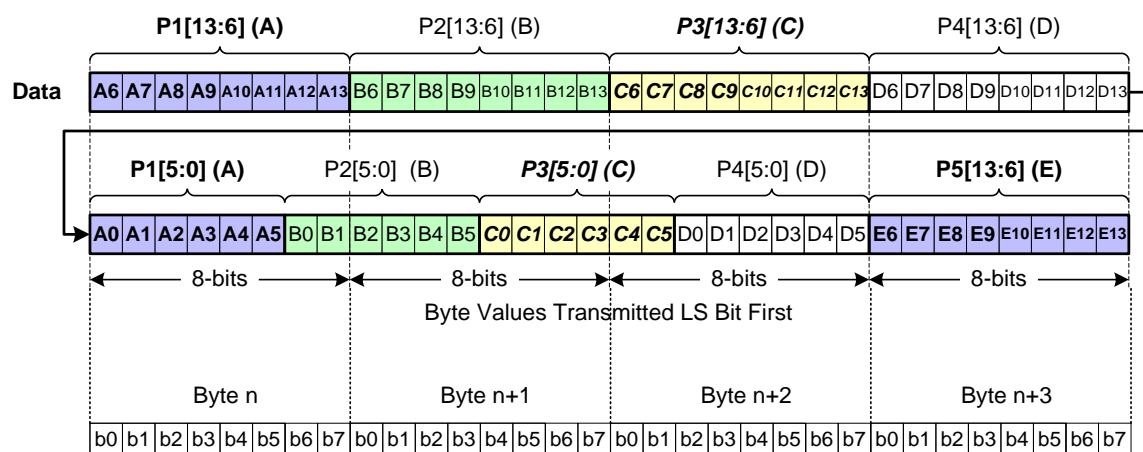


Figure 157 RAW14 Transmission on CSI-2 Bus Bitwise Illustration

2582

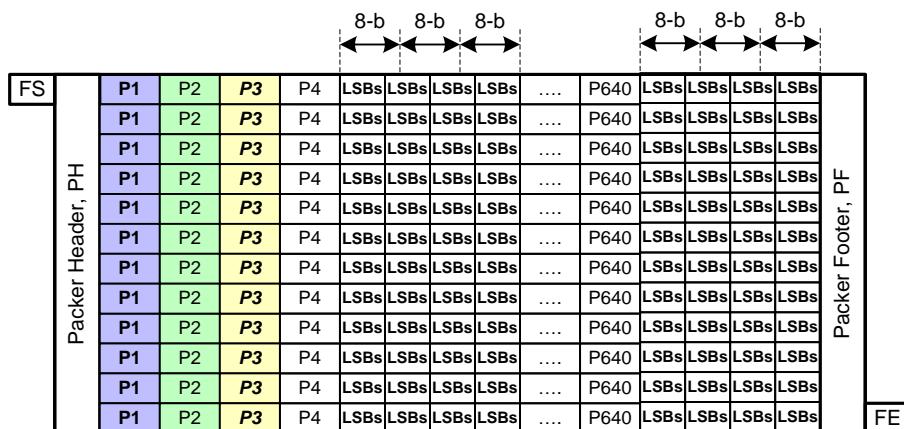


Figure 158 RAW14 Frame Format

11.4.7 RAW16

The transmission of 16-bit Raw data is done by packing the 16-bit pixel data to look like the 8-bit data format. **Table 57** specifies the packet size constraints for RAW16 packets. The length of each packet must be a multiple of the values in the table.

Table 57 RAW16 Packet Data Size Constraints

Pixels	Bytes	Bits
1	2	16

This sequence is illustrated in **Figure 159** (VGA case).

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

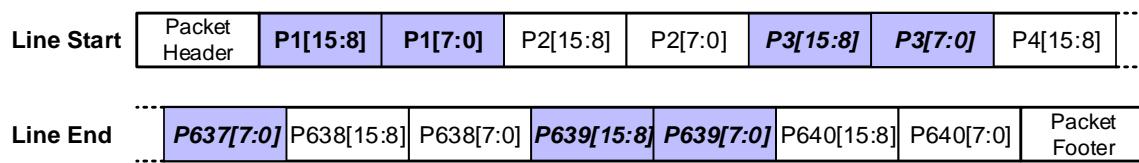


Figure 159 RAW16 Transmission

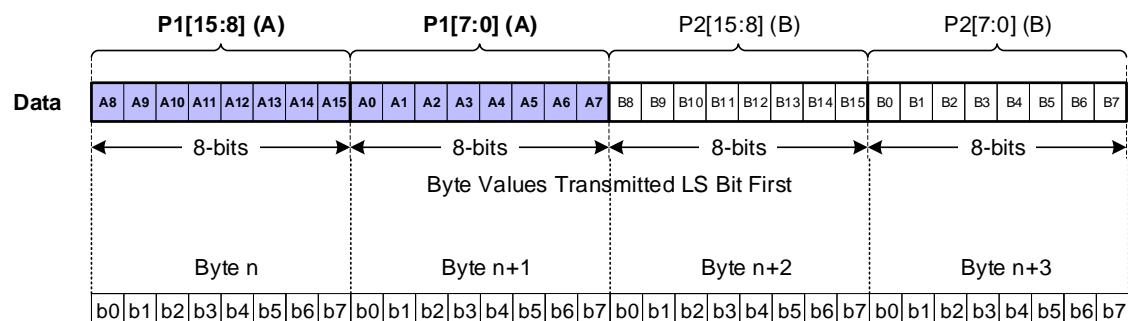


Figure 160 RAW16 Transmission on CSI-2 Bus Bitwise Illustration

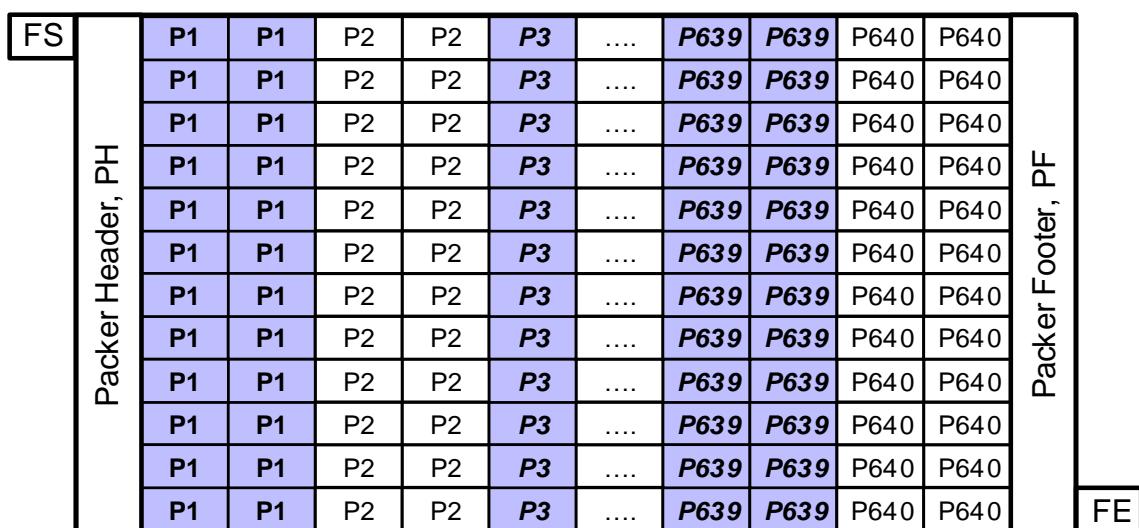


Figure 161 RAW16 Frame Format

11.4.8 RAW20

The transmission of 20-bit Raw data is done by packing the 20-bit pixel data to look like the 10-bit data format. **Table 58** specifies the packet size constraints for RAW20 packets. The length of each packet must be a multiple of the values in the table.

Table 58 RAW20 Packet Data Size Constraints

Pixels	Bytes	Bits
2	5	40

This sequence is illustrated in **Figure 162** (VGA case).

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

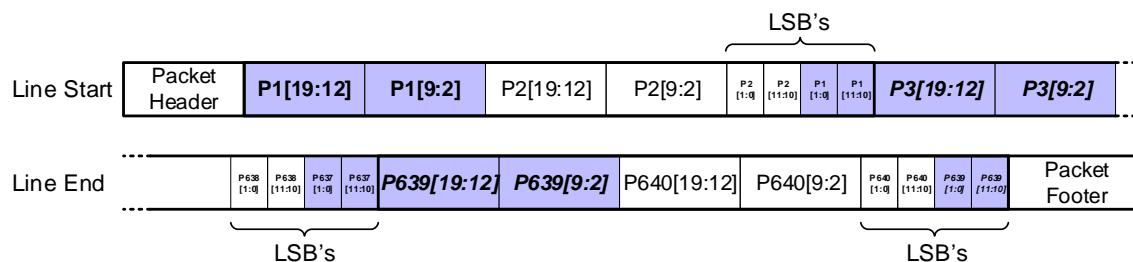


Figure 162 RAW20 Transmission

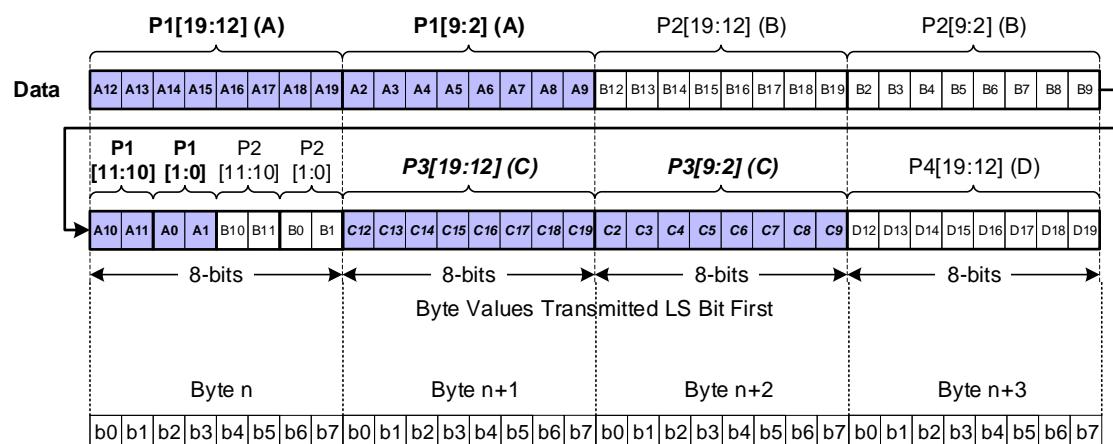


Figure 163 RAW20 Transmission on CSI-2 Bus Bitwise Illustration

2600

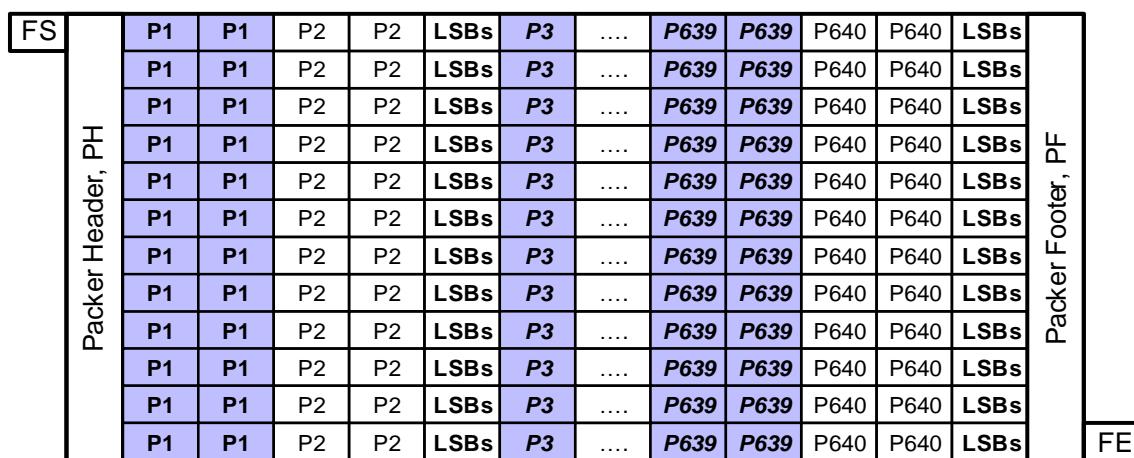


Figure 164 RAW20 Frame Format

11.4.9 RAW24

The transmission of 24-bit Raw data is done by packing the 24-bit pixel data to look like the 12-bit data format. **Table 59** specifies the packet size constraints for RAW24 packets. The length of each packet must be a multiple of the values in the table.

Table 59 RAW24 Packet Data Size Constraints

Pixels	Bytes	Bits
1	3	24

This sequence is illustrated in **Figure 165** (VGA case).

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

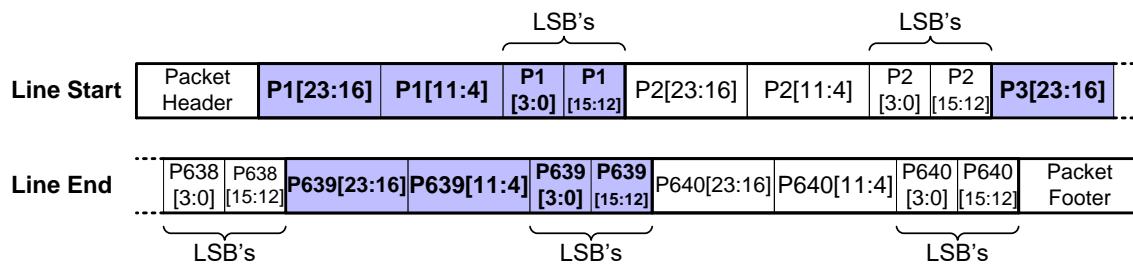


Figure 165 RAW24 Transmission

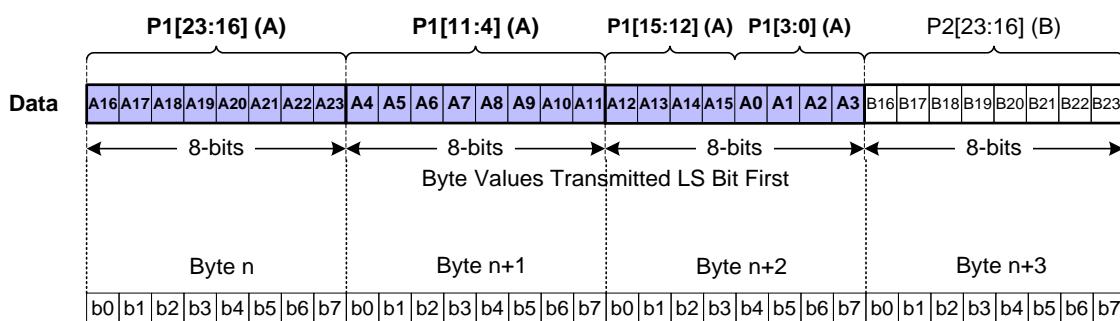


Figure 166 RAW24 Transmission on CSI-2 Bus Bitwise Illustration

2609

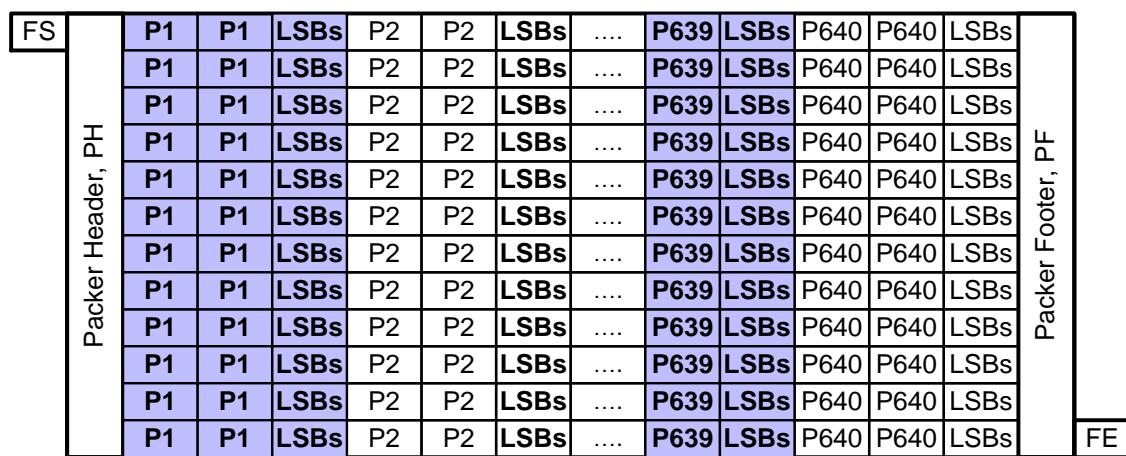


Figure 167 RAW24 Frame Format

11.4.10 RAW28

The transmission of 28-bit Raw data is done by packing the 28-bit pixel data to look like the 14-bit data format. **Table 60** specifies the packet size constraints for RAW28 packets. The length of each packet must be a multiple of the values in the table.

Table 60 RAW28 Packet Data Size Constraints

Pixels	Bytes	Bits
2	7	56

This sequence is illustrated in **Figure 168** (VGA case).

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

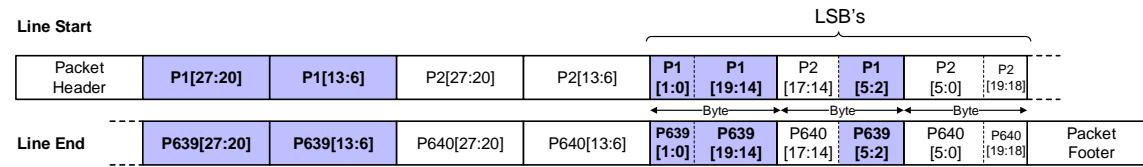


Figure 168 RAW28 Transmission

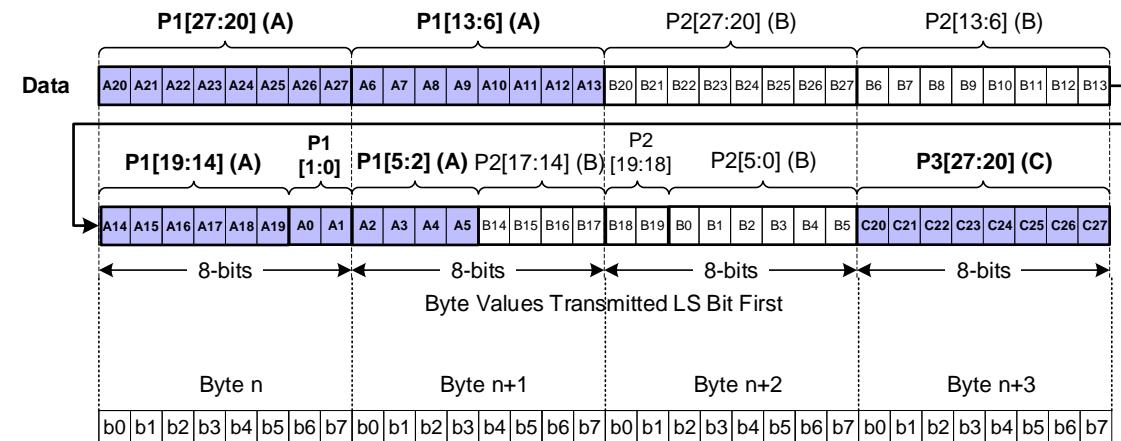


Figure 169 RAW28 Transmission on CSI-2 Bus Bitwise Illustration

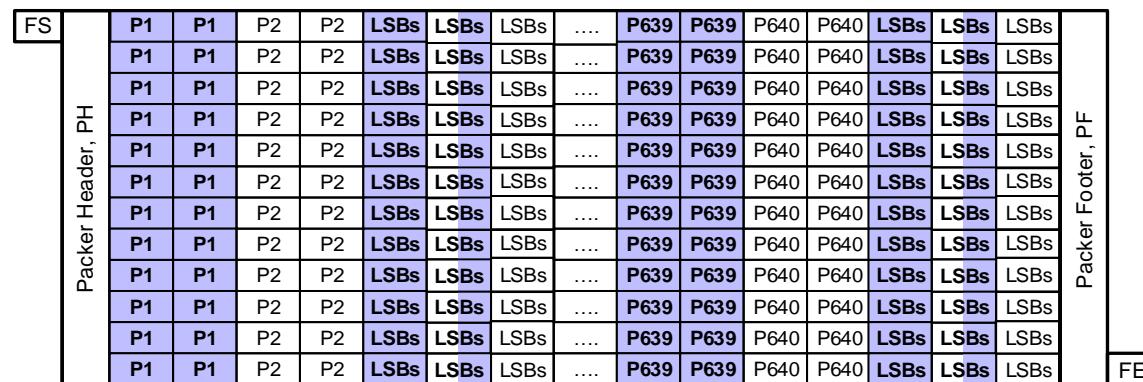


Figure 170 RAW28 Frame Format

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.

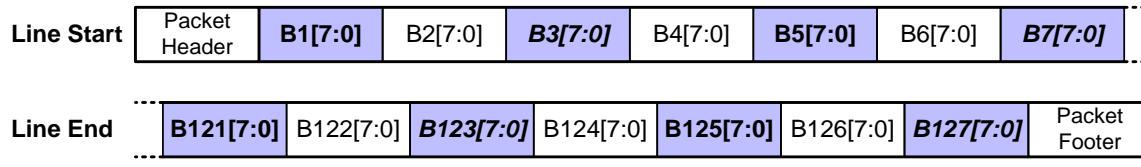


Figure 171 User Defined 8-bit Data (128 Byte Packet)

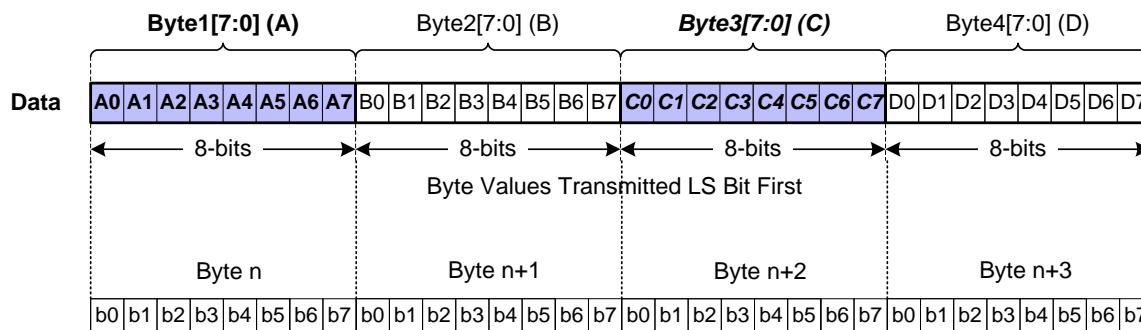


Figure 172 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration

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

For User Defined data:

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

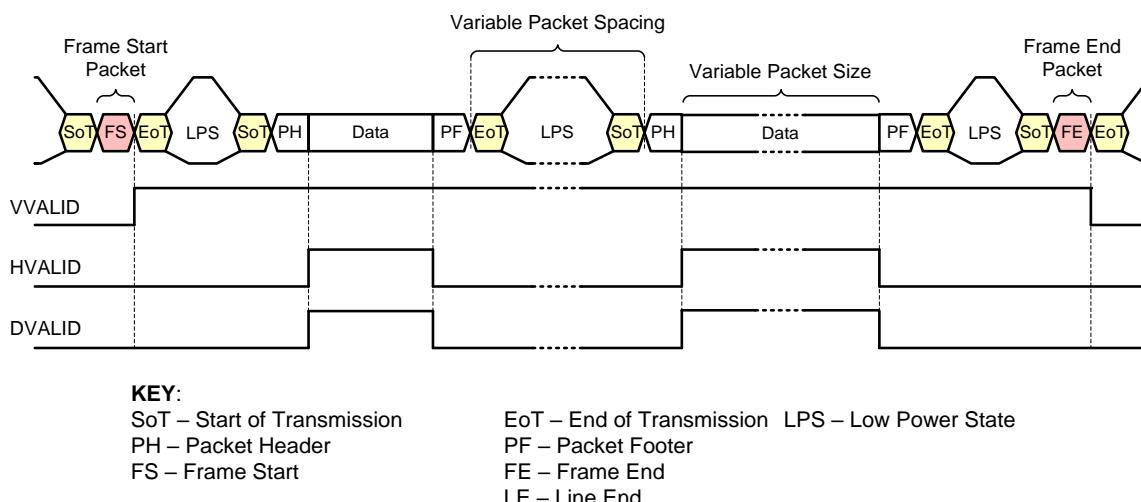


Figure 173 Transmission of User Defined 8-bit Data

2631 Eight different User Defined data type codes are available as shown in **Table 61**.

2632

Table 61 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

This page intentionally left blank.

12 Recommended Memory Storage

2633

This section is informative.

2634

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

2635

2636

2637

12.1 General/Arbitrary Data Reception

2638

2639

2640

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.

2641

Figure 174 shows the generic CSI-2 byte to 32-bit memory word mapping rule.

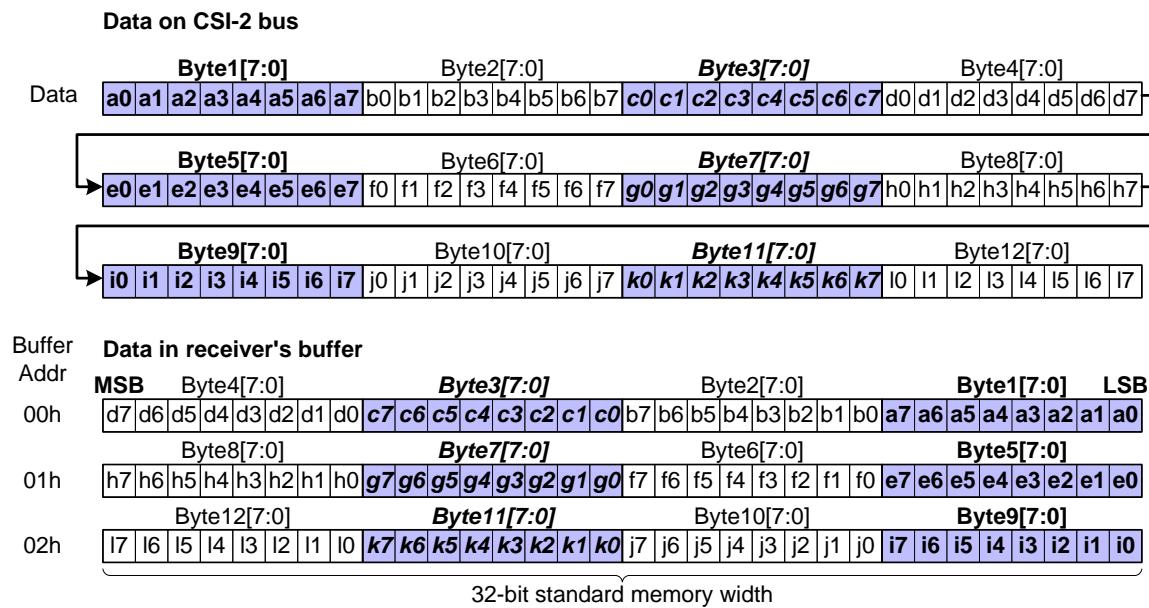


Figure 174 General/Arbitrary Data Reception

2642

12.2 RGB888 Data Reception

2643

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

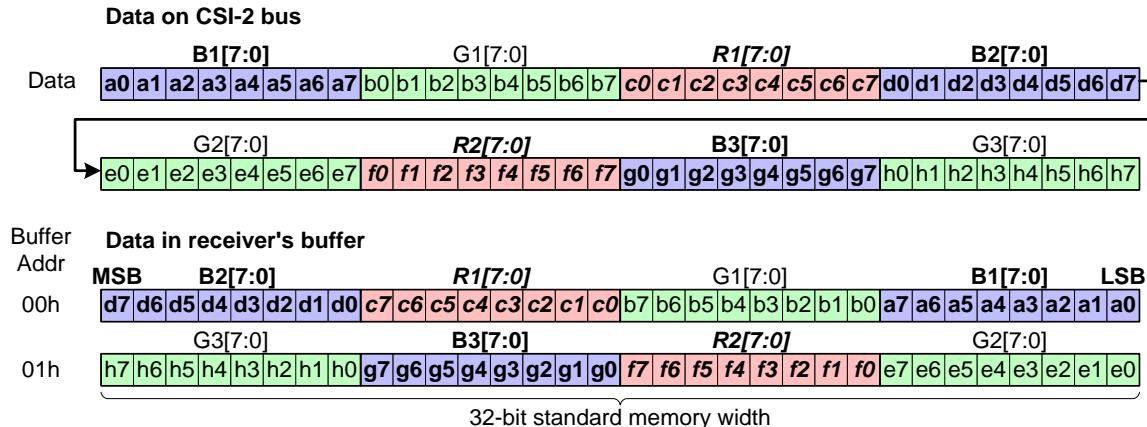


Figure 175 RGB888 Data Format Reception

12.3 RGB666 Data Reception

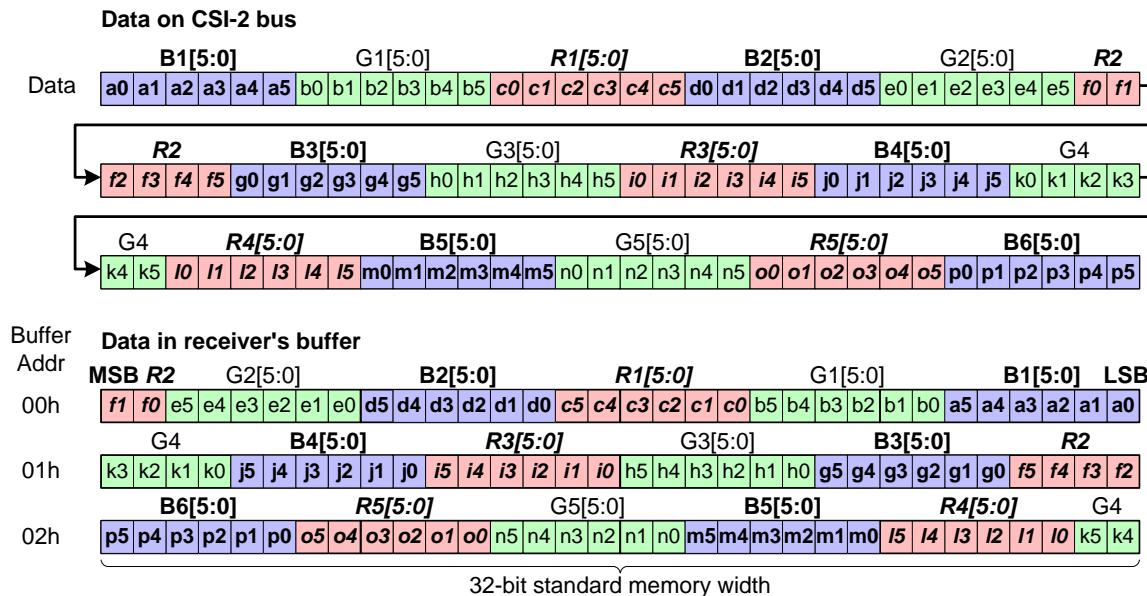


Figure 176 RGB666 Data Format Reception

12.4 RGB565 Data Reception

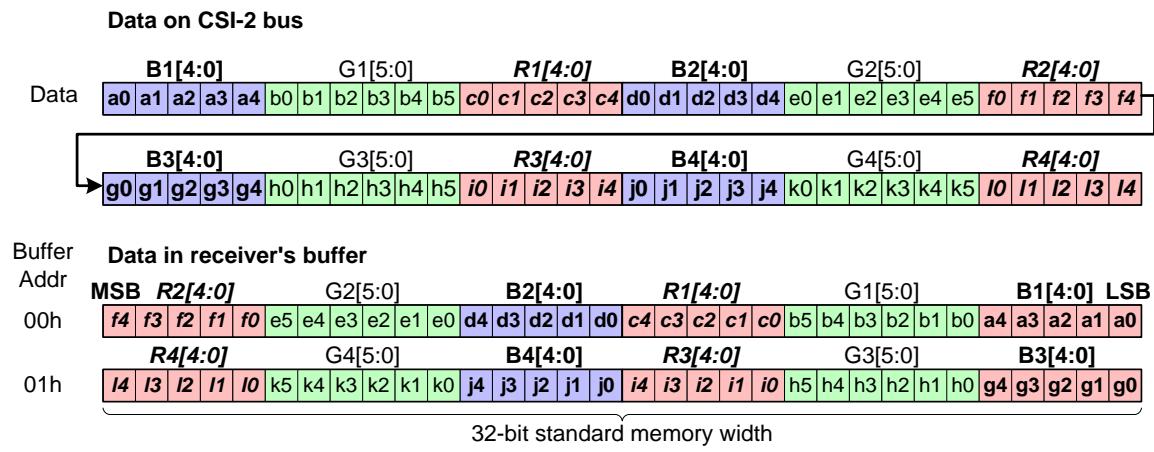


Figure 177 RGB565 Data Format Reception

12.5 RGB555 Data Reception

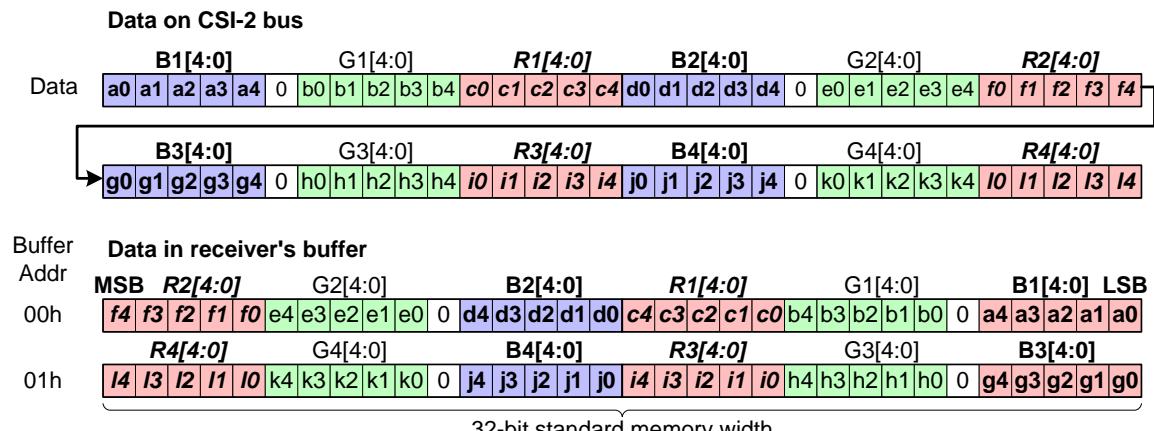


Figure 178 RGB555 Data Format Reception

12.6 RGB444 Data Reception

The RGB444 data format byte to 32-bit memory word mapping has a special transform as shown in *Figure 179*.

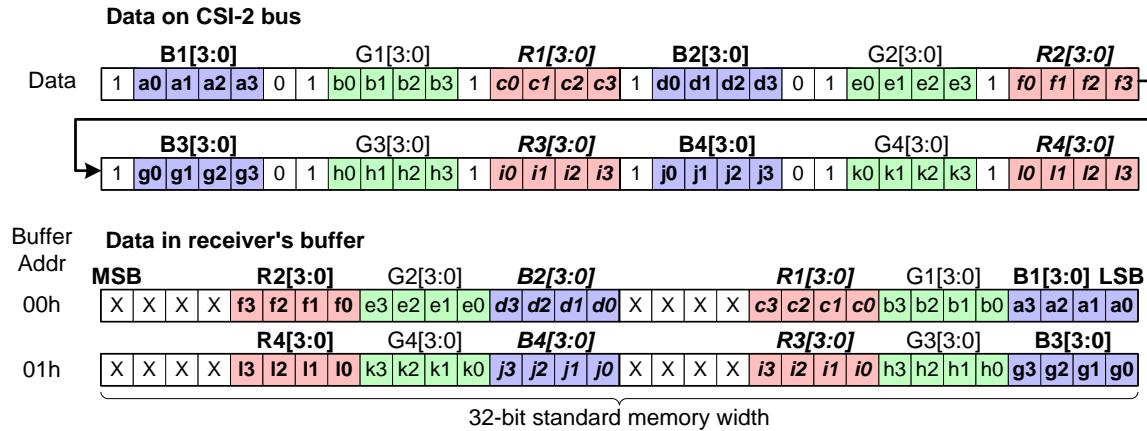


Figure 179 RGB444 Data Format Reception

12.7 YUV422 8-bit Data Reception

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

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

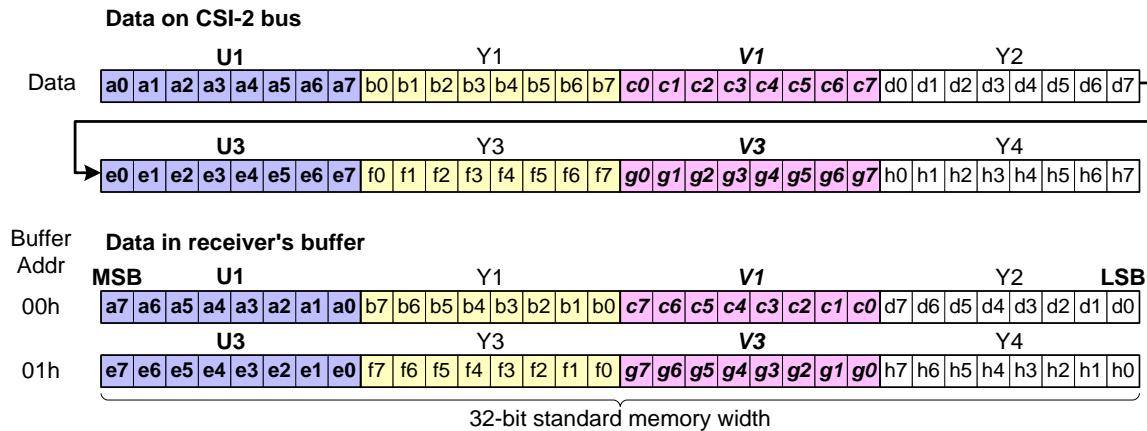
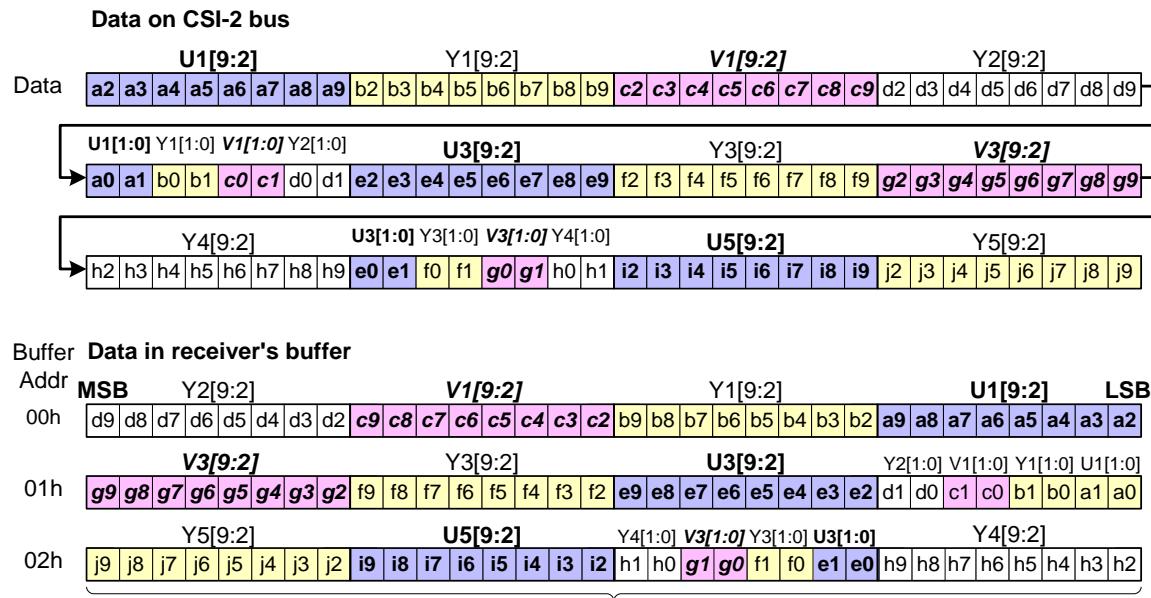


Figure 180 YUV422 8-bit Data Format Reception

12.8 YUV422 10-bit Data Reception

2657

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



2658

Figure 181 YUV422 10-bit Data Format Reception

12.9 YUV420 8-bit (Legacy) Data Reception

2659
2660 The YUV420 8-bit (legacy) data format the byte to 32-bit memory word mapping does not follow the generic
CSI-2 rule.

2661 For YUV422 8-bit (legacy) data format the first byte of payload data transmitted maps the MS byte of the
2662 32-bit memory word and the fourth byte of payload data transmitted maps to the LS byte of the 32-bit memory
2663 word.

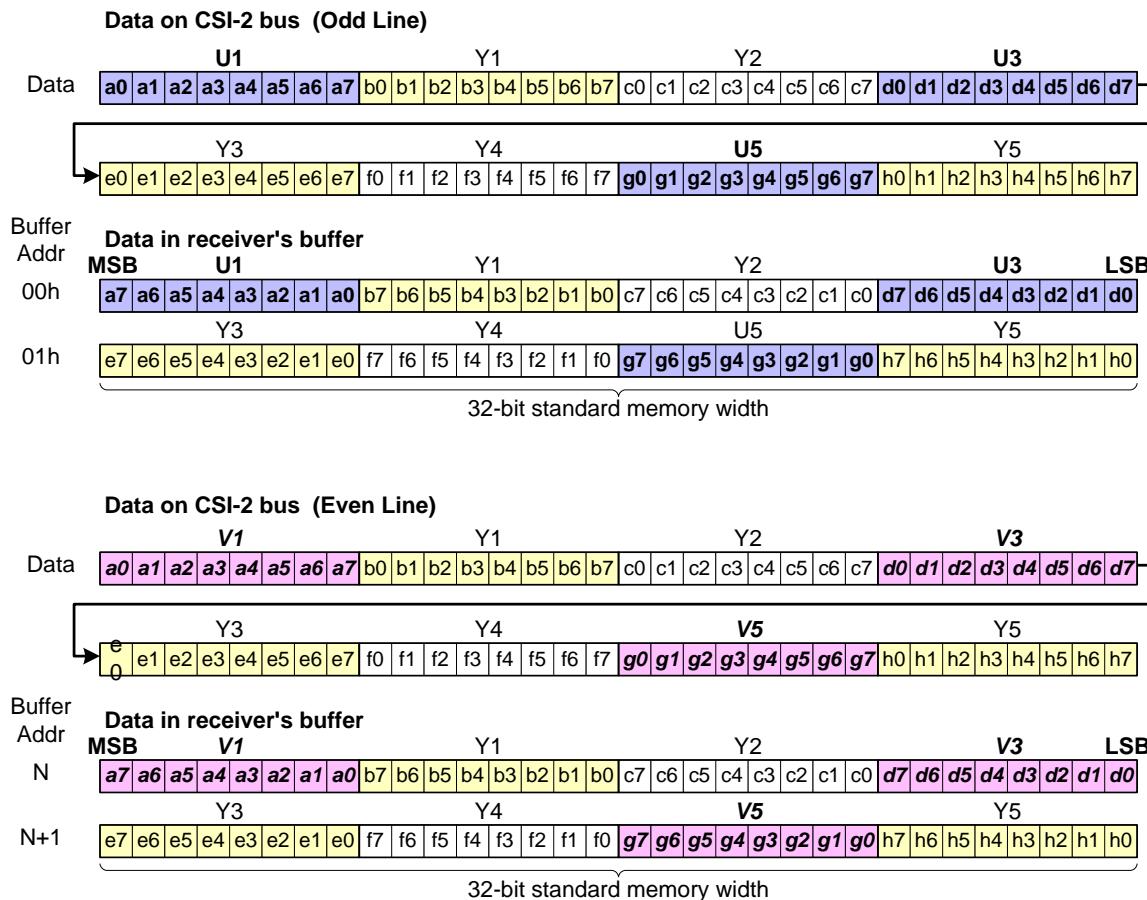


Figure 182 YUV420 8-bit Legacy Data Format Reception

12.10 YUV420 8-bit Data Reception

2665

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

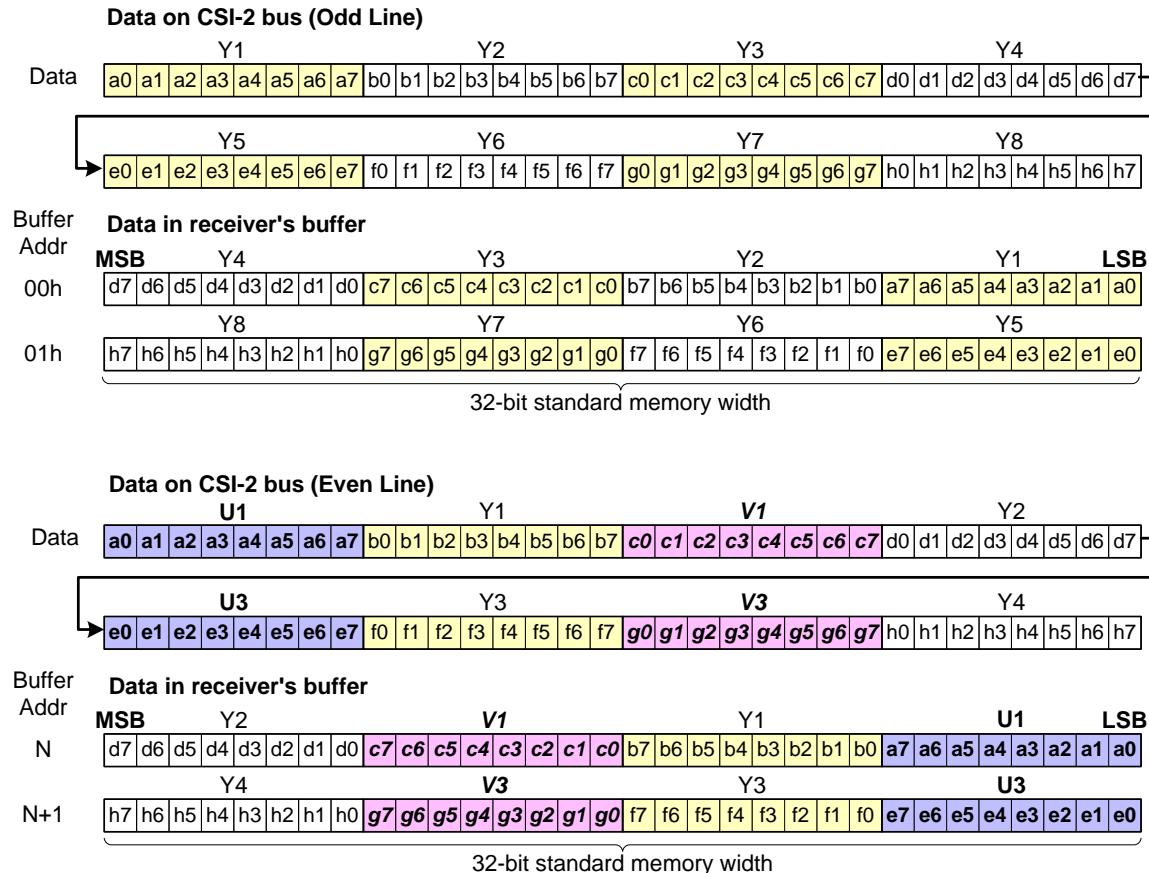


Figure 183 YUV420 8-bit Data Format Reception

2666

12.11 YUV420 10-bit Data Reception

2667

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

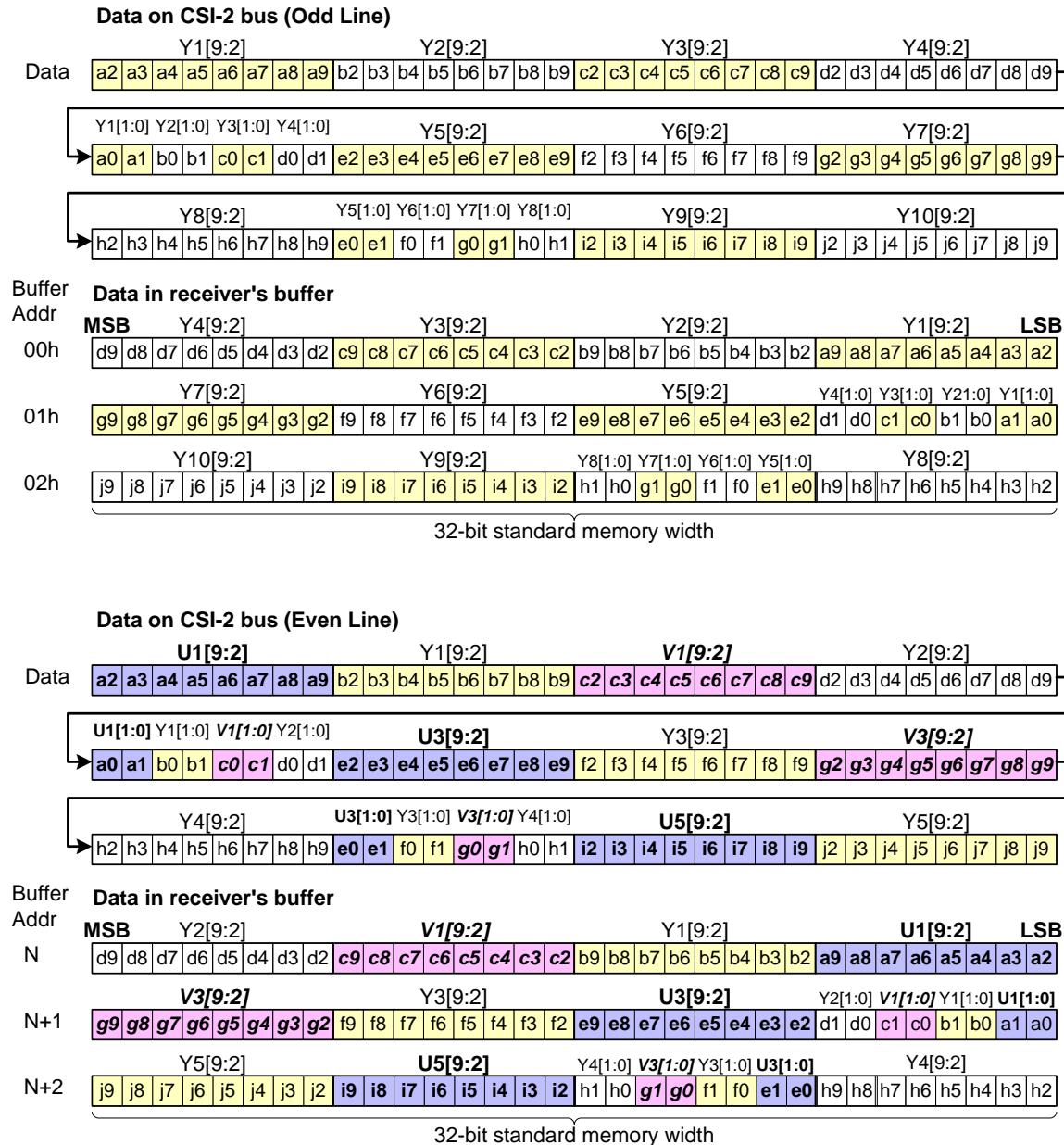


Figure 184 YUV420 10-bit Data Format Reception

2668

12.12 RAW6 Data Reception

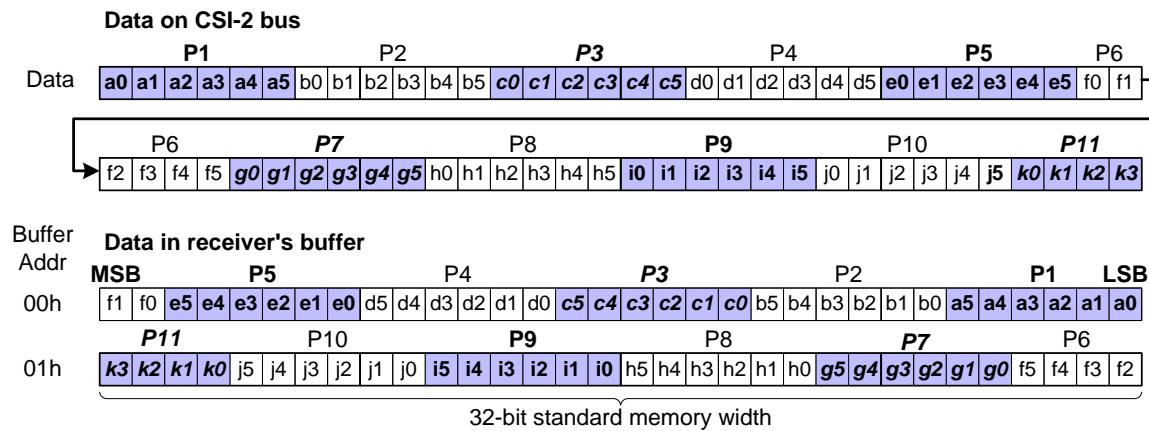


Figure 185 RAW6 Data Format Reception

12.13 RAW7 Data Reception

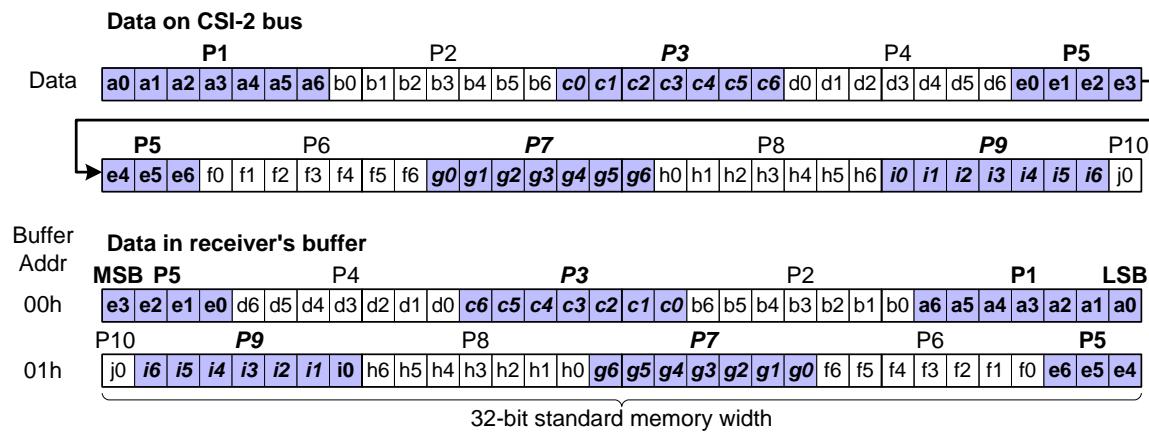
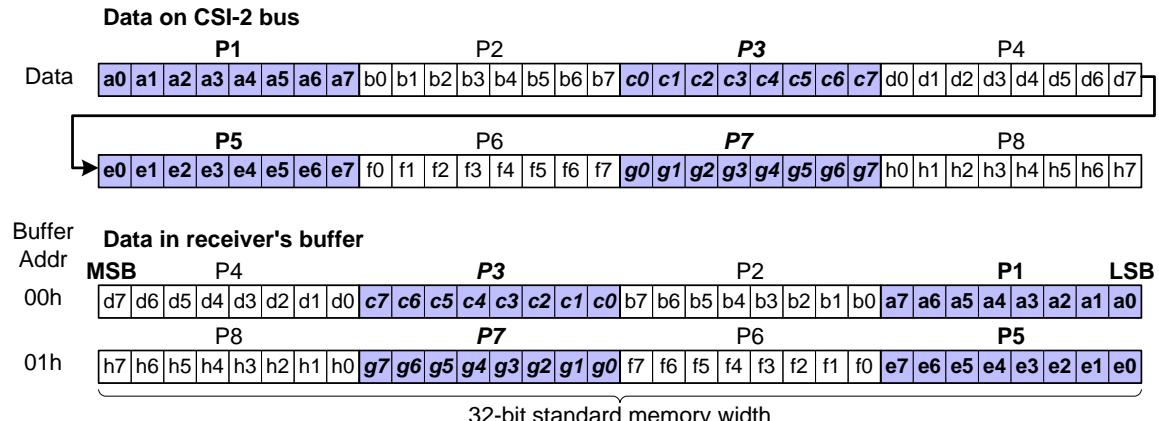


Figure 186 RAW7 Data Format Reception

12.14 RAW8 Data Reception

2671

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



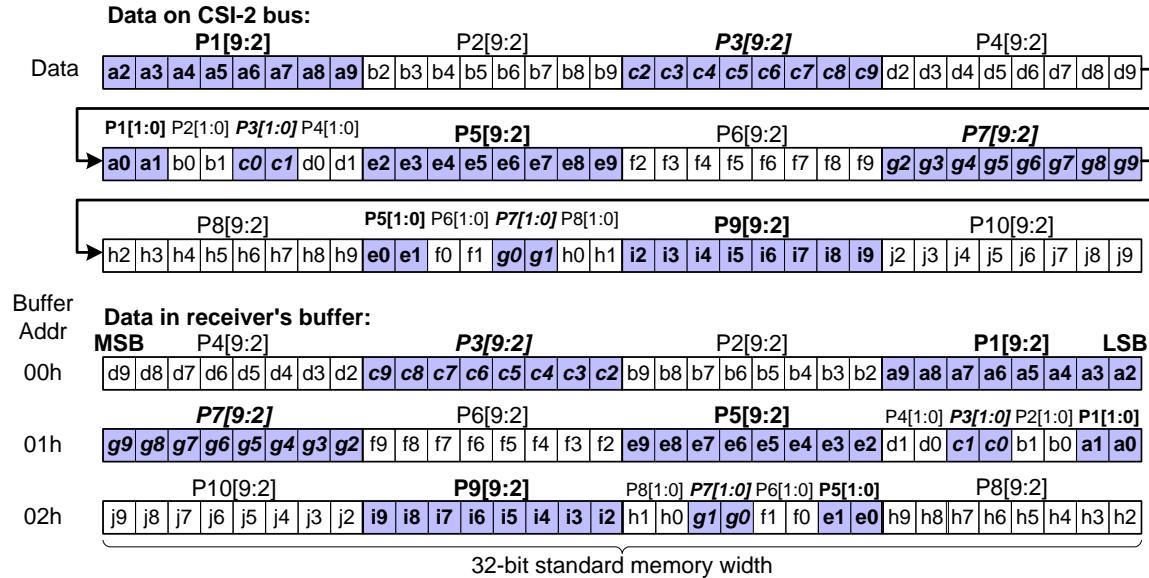
2672

Figure 187 RAW8 Data Format Reception

12.15 RAW10 Data Reception

2673

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



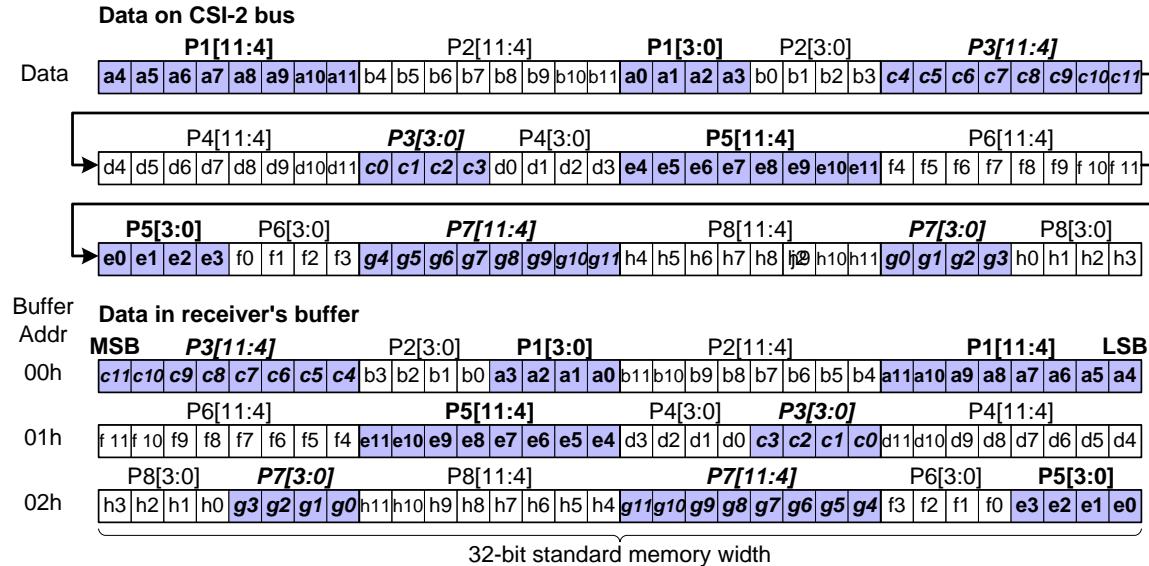
2674

Figure 188 RAW10 Data Format Reception

12.16 RAW12 Data Reception

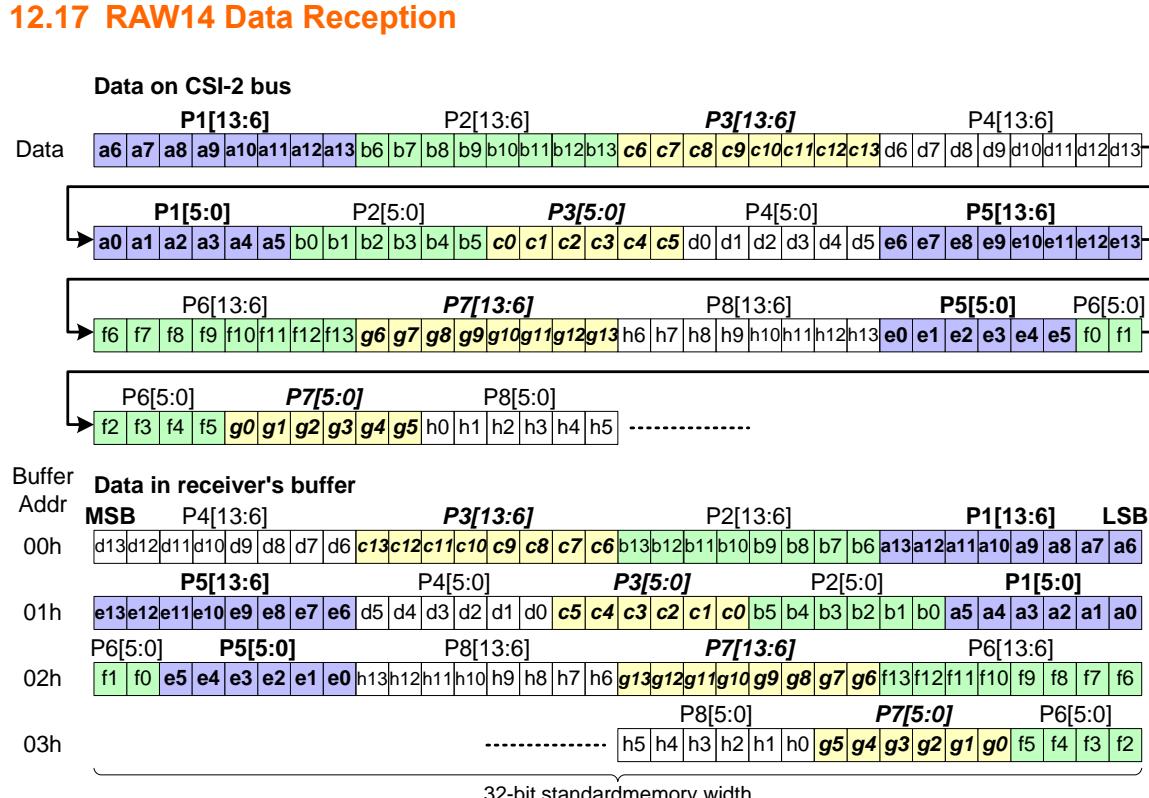
2675

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



2676

Figure 189 RAW12 Data Format Reception



2677

Figure 190 RAW 14 Data Format Reception

12.18 RAW16 Data Reception

2678

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

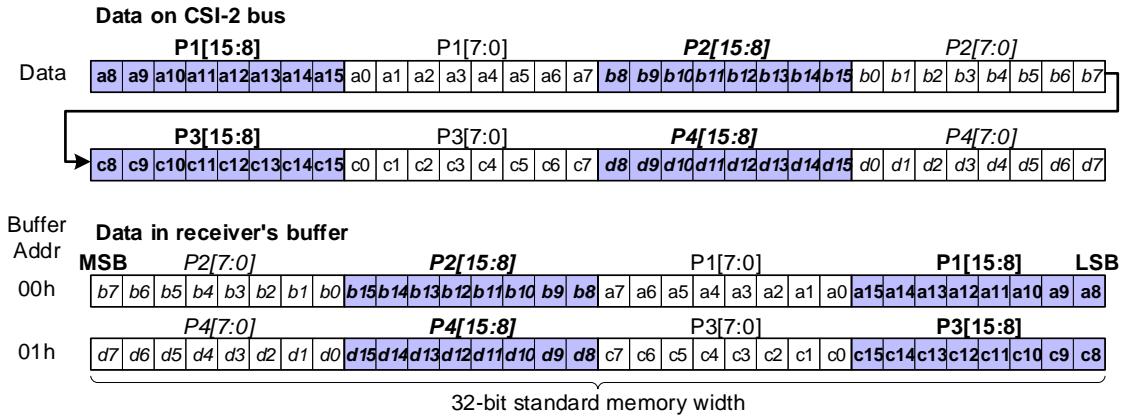


Figure 191 RAW16 Data Format Reception

12.19 RAW20 Data Reception

2680

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

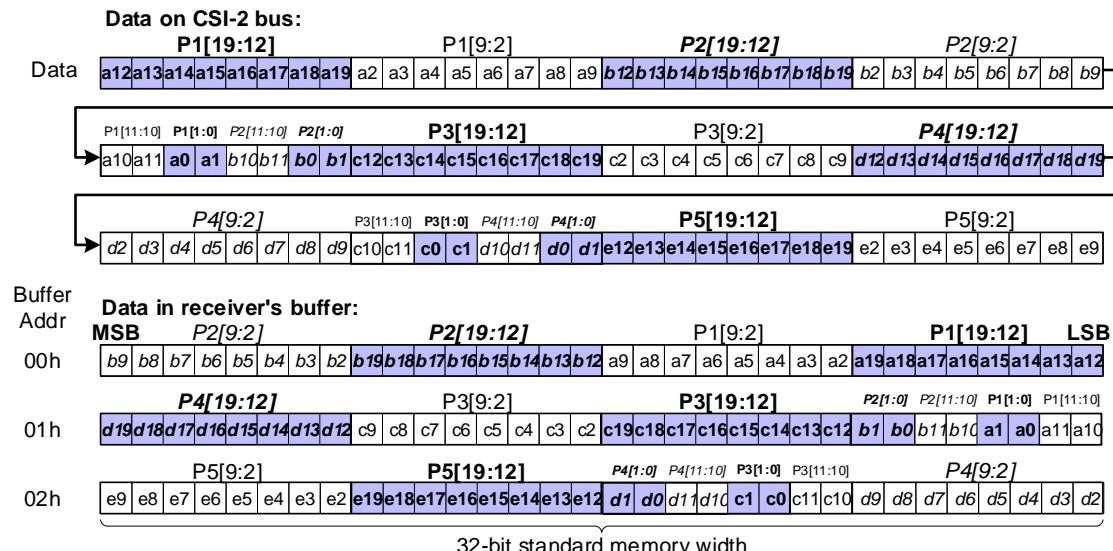
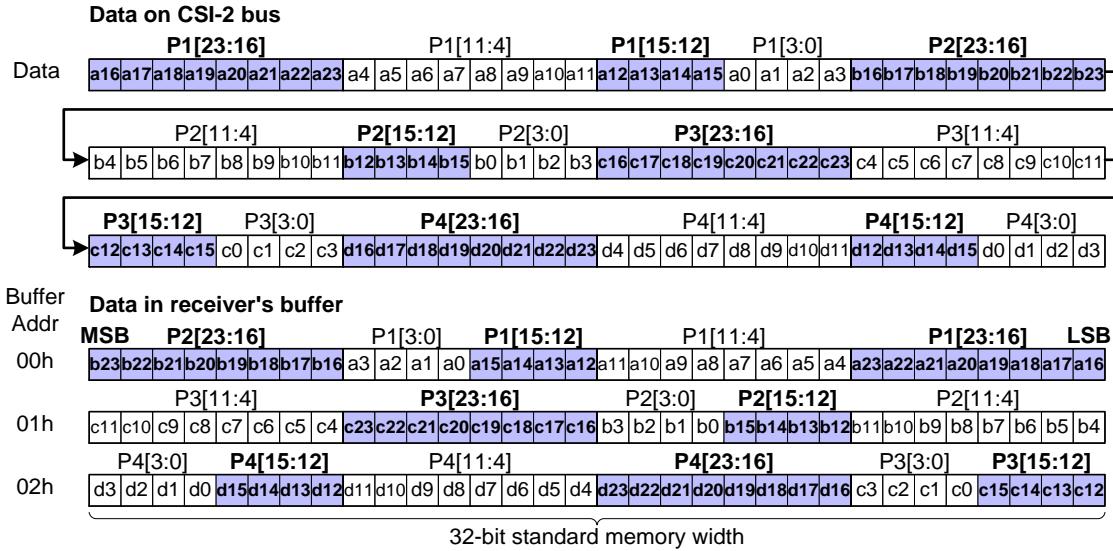


Figure 192 RAW20 Data Format Reception

12.20 RAW24 Data Reception

2682

The RAW24 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.



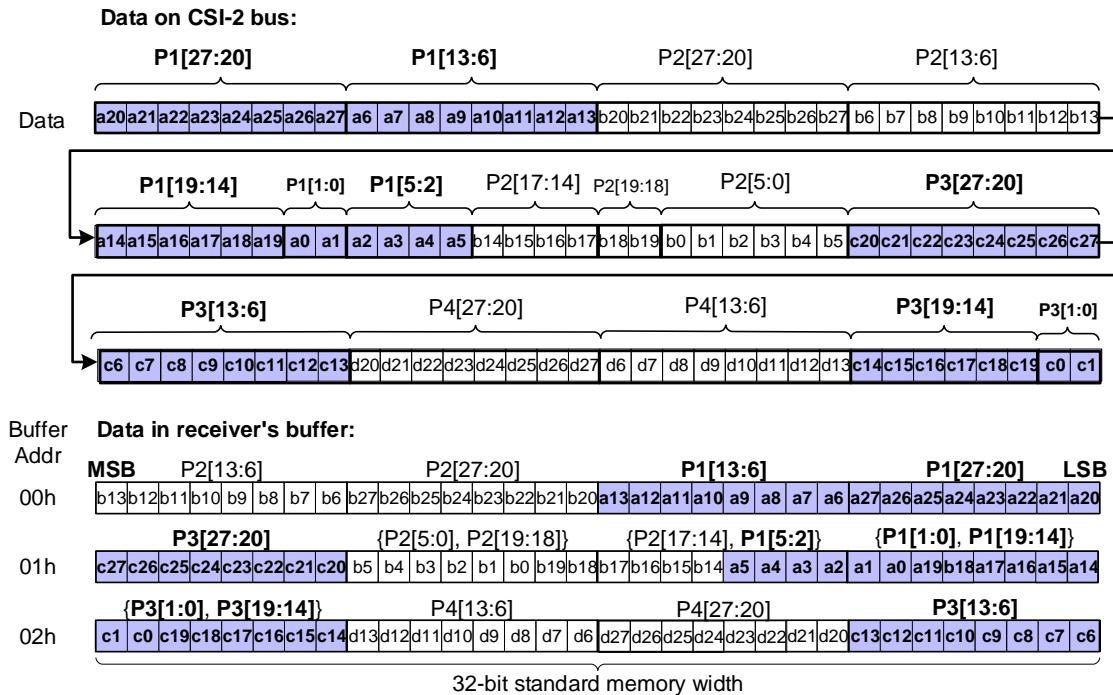
2683

Figure 193 RAW24 Data Format Reception

12.21 RAW28 Data Reception

2684

The RAW28 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.



2685

Figure 194 RAW28 Data Format Reception

This page intentionally left blank.

13 Always-On Sentinel Conduit (AOSC)

13.1 Introduction

Always-On Sentinel Conduit (AOSC) is an optional normative feature. AOSC provides an ultra-low power imaging conduit solution for broad range of always-on and always-aware applications using Vision Digital Signal Processor (VDSP). The AOSC solution entails CSI-2 protocol transport using I3C I/O, in which the VDSP is the I3C Controller, and SNS is the I3C Target. The CSI-2 Specification defines an interface between a peripheral device (SNS) and a host processor (APP) using MIPI D-PHY [[MIPI01](#)] or MIPI C-PHY [[MIPI02](#)] as the physical layer. AOSC defines how the CSI-2 protocol may instead be used with MIPI I3C [[MIPI03](#)] as the physical layer for the purpose of transporting low data rate pixels and/or other data from a SNS to a VDSP. AOSC shall support the I3C SDR and HDR-DDR data rates, and may optionally support the HDR-BT data rate. Imaging solutions requiring higher throughput should utilize Dual Mode and Quad Mode I3C Multi-Lane capabilities with bitwise striping.

A key difference between image transport as described in AOSC and traditional CSI-2 is that in AOSC, the host (VDSP) reads (or “pulls”) pixel data from an image sensor (SNS), whereas in CSI-2 the host passively receives pixel data transmitted (or “pushed”) to it by a SNS.

AOSC product solutions may include:

- SNS connected to an APP supporting VDSP functionality,
- SNS connected to an VDSP, or
- SNS connected to an APP and an external VDSP,

where APP and VDSP operations may or may not be mutually exclusive as illustrated in *Figure 195* through *Figure 197*.

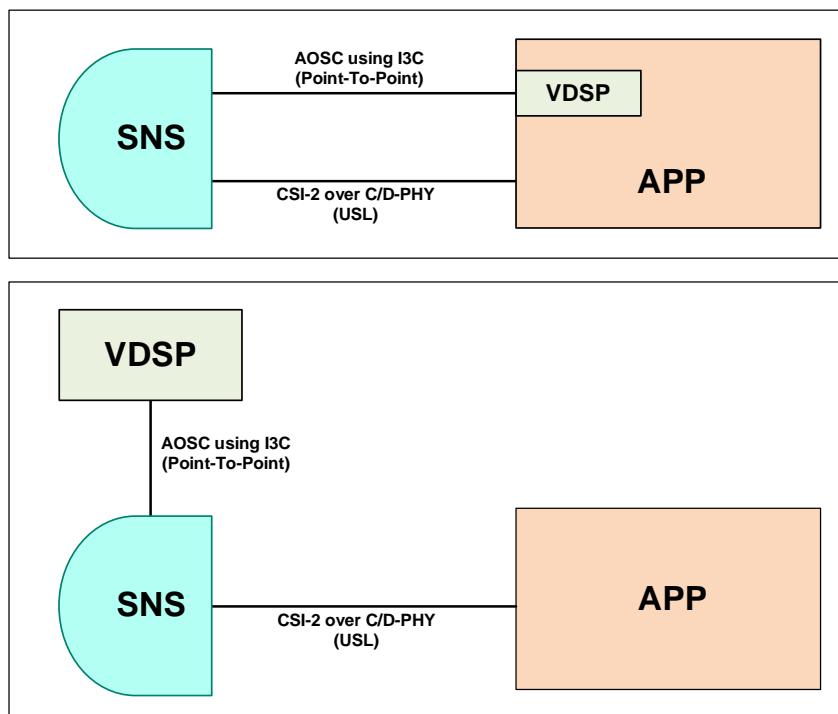
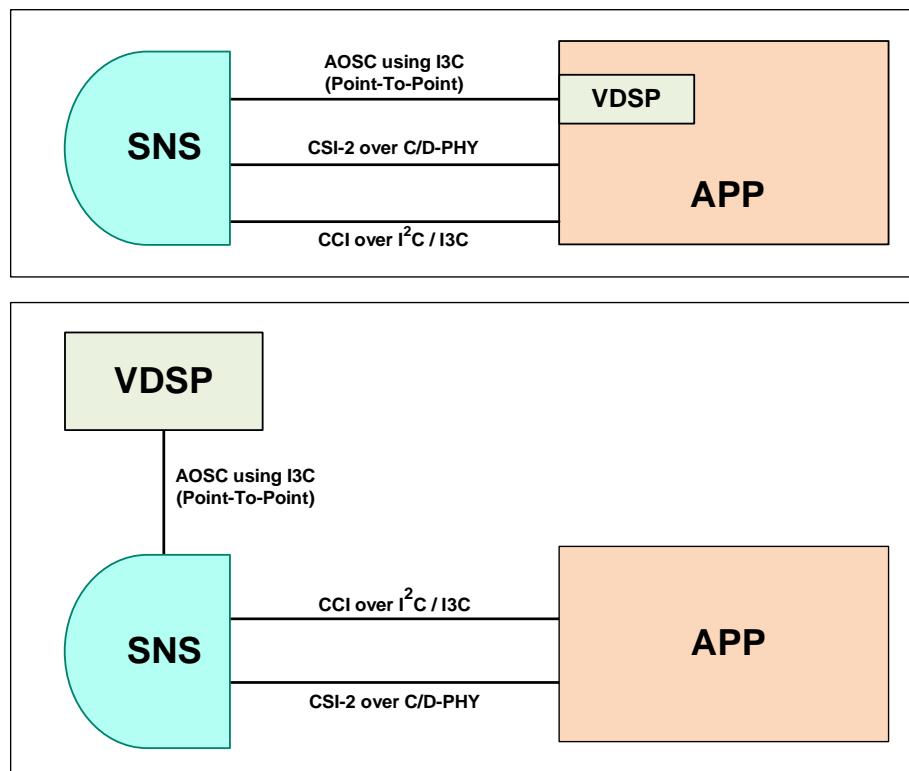
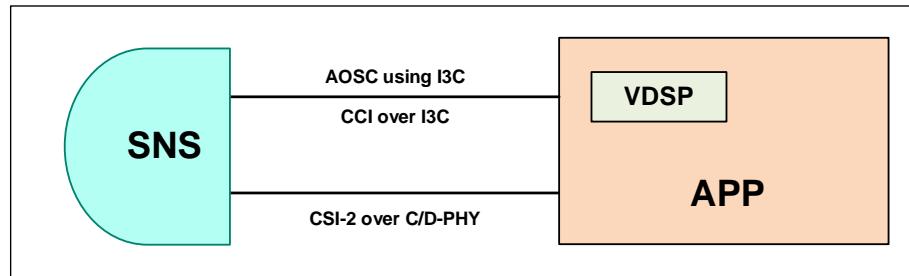


Figure 195 Point-To-Point AOSC Systems with USL Solutions



2706

Figure 196 Point-To-Point AOSC Systems with Non-USL Solutions



2707

Figure 197 System Supporting AOSC and CCI Operations Over Multi-Drop I3C

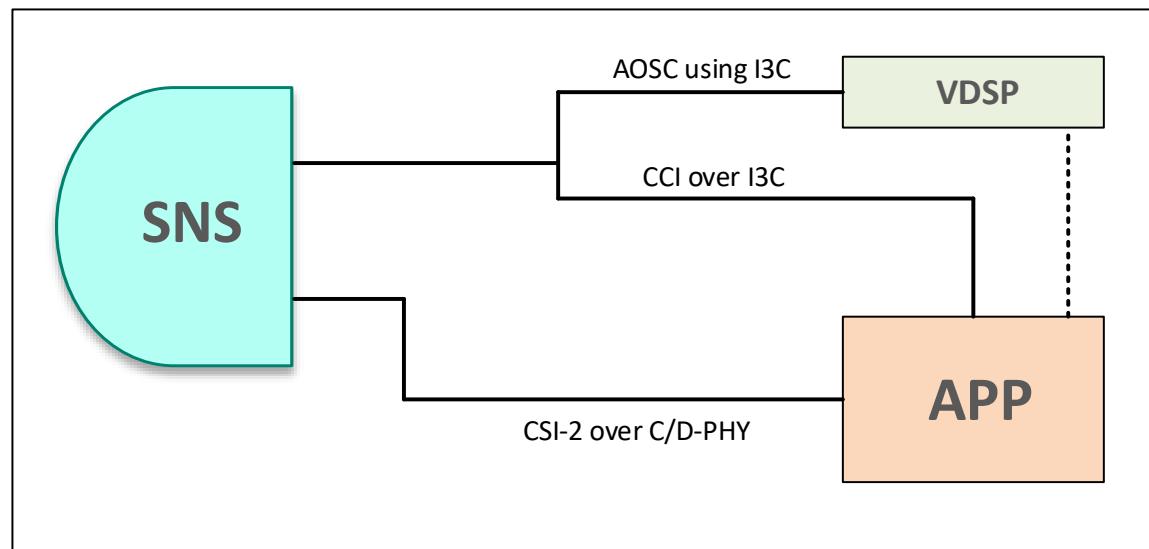


Figure 198 System Supporting Discrete Multicontrollers Mapped to Single SNS I3C Target Port

- The **Figure 195** system illustrates the USL (Unified Serial Link) encapsulated solution for APP and SNS communications as defined in **Section 9.11**.
- The **Figure 196** non-USL solution requires additional I²C / I3C / SPI wires for APP-initiated CCI operations to the SNS; in such operations, the SNS shall be designated as the Target and APP as the Controller.
- The **Figure 198** system enables non-integrated or discrete VDSP and APP supporting I3C host arbitration and handshake on the platform using Multi-Controller topology.

An AOSC SNS supporting CSI-2 transport over I3C and C/D-PHY may provide respective dedicated control registers (i.e., 3A control registers). An AOSC SNS may support simultaneous and mutually exclusive CSI-2 frame transport operations over I3C and C/D-PHY (as illustrated in the above figures). An AOSC SNS limited to low resolution operations may only support CSI-2 transport over I3C.

13.1.1 VDSP and APP In-Band Communication

Mailbox registers are utilized for systems with separate VDSP silicon and an APP silicon connected to an SNS, as illustrated in **Figure 195** and **Figure 196**. The SNS should provide two 32-bit mailbox registers for VDSP and APP in-band communications. Any possible sideband communication between VDSP and APP is implementation-specific and beyond the scope of this specification.

The two Mailbox registers are:

- **REG_AOSC_MAILBOX_VDSP_APP [31:0]** for VDSP-to-APP commands
- **REG_AOSC_MAILBOX_APP_VDSP [31:0]** for APP-to-VDSP commands

13.1.2 AOSC SNS Features and Capabilities

The SNS supporting AOSC shall support a 16-bit control register, **REG_AOSC_CONTROL[15:0]**. The AOSC features and capabilities are detailed in the sections below and mapped to bits **REG_AOSC_CONTROL[8:0]**. The remaining bits are reserved for future developments. The AOSC SNS may support multiple of 8-bit width register addressing.

- Bit **REG_AOSC_CONTROL[0]** (field **ENABLE_AOSC**) shall be used to enable the AOSC feature:
 - 1'b0: AOSC feature disabled
 - 1'b1: AOSC feature enabled

This specification defines two AOSC Transport Modes:

- **Optimal Transport Mode (OTM)** is intended to provide an optimal power-efficient AOSC transport solution for inference processing. Each LP shall map to a horizontal row of the photon collectors. The OTM shall be supported, see *Section 13.2*.
- **Smart Transport Mode (STM)** is intended for smart inferencing SNS solutions where the transport payload is limited to the I3C IBI. The STM should be supported, see *Section 13.3*.

For both OTM and STM, each image frame is transported over I3C as a predetermined number of fixed-length “long packets” (LPs), each of which consists of the pixels from a single image line. For the OTM, an LP may also be terminated by a variable number of “Interconnect Synchronizing Padding Bytes” (ISPB) which may be inserted by the SNS for bit rate synchronization purposes.

- Bits **REG_AOSC_CONTROL[7:6]** (field **AOSC_TRANSPORT_MODE**) shall be used to select the AOSC Transport Mode:
 - 2'b00: OTM
 - 2'b01: STM
 - 2'b10–2'b11: Values reserved for future AOSC Transport Modes

13.1.3 VDSP and APP Switching and Graceful Failure

The SNS should switch between VDSP and APP at CSI-2 frame boundaries, and the SNS should complete transmission of the present CSI-2 frame in flight prior to switching. In the event of an internal error within an SNS, any LP payload in transmission should be completed using multiples of 8'd0 padding; and any remaining LPs mapped to the CSI-2 frame shall be flushed. Upon flushing while in OTM, the SNS shall generate an IBI indicating an Internal Error as described in *Section 13.2.4*.

13.1.4 Support for Privacy

The SNS shall support privacy provisions to facilitate an electronic slider for always-on and always-aware imaging applications. An SNS targeting secure product platforms should support physical GPIO pin override to enable privacy. A platform may disable **PRIVACY** using GPIO or CCI as defined in this Section.

The GPIO pin HIGH signaling maps to **PRIVACY_ENABLED** functionality as defined in the paragraph below. CCI accessible register **REG_AOSC_CONTROL** is used to define privacy when GPIO pin signaling is configured to LOW. Physical GPIO privacy is disabled by default, and the SNS module should include the pull-down provision within the pad as appropriate.

Privacy changes made to the SNS using the physical GPIO pin or CCI registers shall not impact any frame(s) presently in transport by the SNS; and the privacy changes shall apply to the subsequent frame(s).

If **PRIVACY** is Enabled using the physical GPIO pin, then no digital representation(s) or interpretation(s) of streaming or still image frame(s) from the SNS photon collectors shall be made available.

If **PRIVACY** is Enabled using CCI accessible register **REG_AOSC_CONTROL[2:1]** (field **PRIVACY**), then permissible SNS operations include:

- **2'b00: PRIVACY_DISABLED:** Digital representation(s) or interpretation(s) of streaming or still image frame(s) from the SNS photon collectors may be made available to the VDSP and/or APP.

- **2'b01: PRIVACY_ENABLED_EVENT_DET_BYPASS:** No digital representation(s) of streaming or still image frame(s) from the SNS photon collectors shall be made available to the VDSP nor to the APP. Digital interpretation(s) of streaming or still image frame(s) from the SNS photon collectors may be made available to the VDSP and/or to the APP (i.e., Event Detection via I3C IBI).

System example of an event detection operation (hypothetical):

1. AOSC SNS detects an object of interest passing confidence threshold
 2. AOSC SNS generates an I3C IBI to the VDSP/APP
 3. VDSP/APP may optionally update PRIVACY configuration
 4. VDSP or APP may optionally read the ROI Image Frame(s) via I3C or C/D-PHY respectively
- **2'b10: PRIVACY_ENABLED:** No digital representation(s) or interpretation(s) of streaming or still image frame(s) from the SNS photon collectors shall be made available.
 - **2'b11: PRIVACY_RESERVED:** This value is reserved.

13.2 Optimal Transport Mode (OTM) Overview

The OTM is intended for transporting static CSI-2 frames configured by the VDSP or APP. The number of pixel bytes transported in each OTM LP by the SNS shall be determined by the CSI-2 Frame resolution and bits-per-pixel configured by the VDSP or APP as part of the SNS bring-up. The implementation specific SNS datasheet may be used to configure the SNS by the VDSP or APP. For example, a RAW10 format, 300 Horizontal Pixel by 320 Vertical Pixel CSI-2 Frame shall require 3,000 pixel bits (or 375 pixel bytes) per OTM LP. The SNS shall transport 320 LPs corresponding to each CSI-2 Frame.

The long packet 16-bit CRC checksum described in *Section 13.2.8* is optional for OTM image frames, in order to provide “ultra-low-power always-on” SNS and VDSP more flexibility in minimizing power. The OTM CRC does not apply to I3C HDR-BT mode. The SNS shall support generation of the OTM CRC for the I3C SDR and HDR-DDR modes.

- Bit **REG_AOSC_CONTROL[8]** (field **OTM_CRC_GEN**) shall be used to enable or disable the generation of the OTM CRC:
 - **1'b0:** The SNS shall not generate the 16-bit CRC.
 - **1'b1:** The SNS shall generate a single 16-bit CRC for all LP Payloads corresponding to an image frame, and shall append the 16-bit CRC to the last LP comprising an image frame.

An OTM SNS may optionally provide a register to enable the VDSP to configure the EHDR link rate in Bits Per Second as part of the power up configuration. The VDSP should configure register **REG_AOSC_EHDR_RATE_BPS** with an I3C supported data rate. The effective read data rate may be lower than the I3C link rate. The VDSP should ensure that the link rate is supported by the AOSC SNS as defined in the data sheet.

Example settings of **REG_AOSC_EHDR_RATE_BPS[31:0]** showing the targeted data rate (in Bits per Second) at which the VDSP will read the SNS FIFO (see *Section 13.2.2*):

32'd0:	The VDSP hasn't configured the SNS with the EHDR data rate
32'd1:	The VDSP should read the FIFO at 1 bps after EHDR
...	
32'd12_500_000:	The VDSP should read the FIFO at 12.5 Mbps
...	
32'd25_000_000:	The VDSP should read the FIFO at 25 Mbps

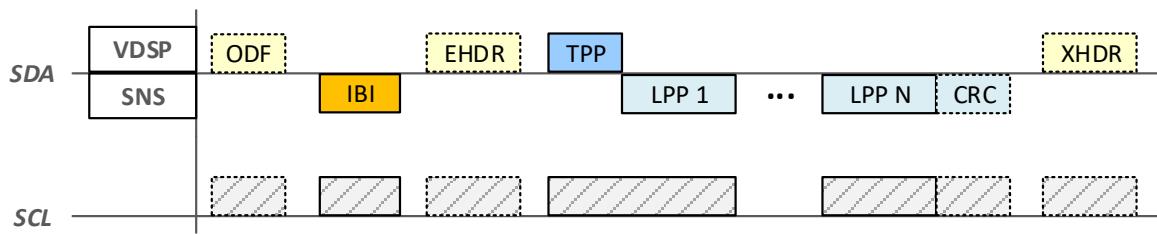


Figure 199 AOSC Optimal Transport Mode (OTM) Operation

Figure 199 illustrates OTM operation:

- Note that OTM operations in the dotted boxes are optional
- **ODF:** On-Demand Frame command to initiate frame exposure
- **IBI:** In-Band Interrupt generated by the SNS at the beginning of an image frame
- **EHDR:** The VDSP may optionally enter into an I3C HDR Mode that supports a higher data rate
- **TPP:** The VDSP generates a Transmit Packet Payload command to the SNS
- **LPP 1** through **LPP N:** The VDSP reads all Long Packets (1 to N) encompassing an image frame from the SNS. ISPB may be appended to select Long Packets.
- **CRC:** The VDSP reads the 16-bit CRC from the SNS if the **OTM_CRC_GEN** was enabled
- **XHDR:** The VDSP shall exit I3C HDR Mode (see **EHDR** step above)

OTM control registers are summarized in **Table 62**. Note that some register bits also apply to STM.

2823

Table 62 SNS Registers Required to Support OTM

SNS OTM Registers and Values <i>Bits marked with * also apply to STM</i>		Description RO: Read Only RW: Read / Write	See Section
REG_AOSC_CONTROL[15:0]		RW	–
Bit [0]*	1'b0: AOSC feature disabled 1'b1: AOSC feature enabled	–	13.1.2
Bit [2:1]*	2'b00: Privacy disabled 2'b01: Privacy enabled with event detection 2'b10: Privacy enabled 2'b01: Reserved	–	13.1.4
Bit [3]	1'b0: ODF Mode 1'b1: CSF Mode	–	13.2.5
Bit [4]	1'b0: The SNS shall not squelch Frame Start IBI 1'b1: The SNS shall squelch Frame Start IBI	–	13.2.6
Bit [5]	1'b0: Dynamic ISPB insertion disabled 1'b1: Dynamic ISPB insertion enabled	–	13.2.7
Bit [7:6]*	2'b00: OTM 2'b01: STM 2'b10: Reserved for future AOSC Transport Mode 2'b11: Reserved for future AOSC Transport Mode	–	13.1.2
Bit [8]	1'b0: 16-bit CRC disabled 1'b1: 16-bit CRC enabled	–	13.2
REG_AOSC_MAILBOX_VDSP_APP [31:0]*		RW	13.1.1
REG_AOSC_MAILBOX_APP_VDSP [31:0]*		RW	13.1.1
REG_AOSC_SQ_FS_IBI [15:0]		RW	13.2.6
REG_AOSC_ISPB_IBI_WMR[15:0]		RW	13.2.7
REG_AOSC_EHDR_RATE_BPS[31:0]		RW	13.2

13.2.1 Protocol Overview

As shown in **Figure 200** and **Table 63**, pixel bits in each OTM LP are mapped into I3C bytes in LS-bit-first order; the bit transmission order of each byte on the I3C bus is determined by the I3C specification for the selected I3C transfer mode. OTM line transport shall follow the formats defined in **Table 63** which are derived from **Section 11** but generally relax the latter's pixel granularity rules; i.e. OTM simply requires lines to consist of an integer number of pixels, with any remaining non-pixel bits in the last byte of a line being set to zero.

For example, an OTM line could consist of 130 RAW10 pixels transported in 163 bytes (i.e., not a multiple of four pixels or five bytes as described in **Section 11.4.4**), with the final byte consisting of the four MS bits of the last pixel followed by four zero bits.

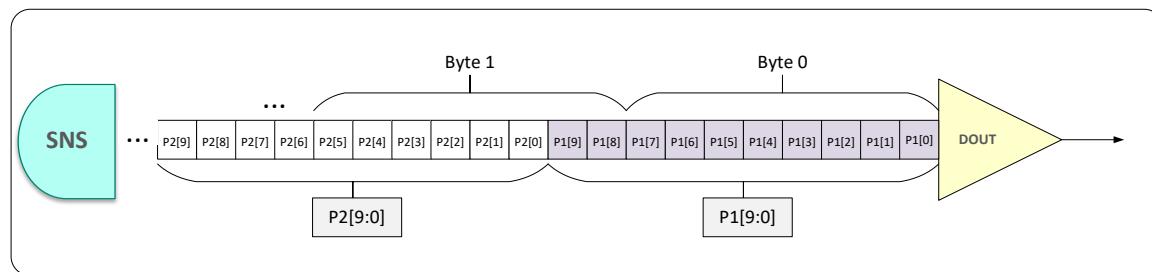


Figure 200 Example OTM RAW10 Format Transport (with I3C SDR Bit Transmission Order)

Table 63 provides comprehensive OTM SNS bitwise transmission mapping for all frame formats. These are mapped into data bytes for I3C read transfers, which shall be transmitted in the appropriate order based on the I3C Mode.

The SNS shall use an appropriate byte format and protocol for the chosen I3C Mode.

For example:

- **In I3C SDR Mode** the bytes shall be sent in the defined order, one byte at a time, starting with Byte 0 and using a T-Bit after each byte. The bits within each byte shall be sent in MSb-first order, per standard SDR Mode protocol (see [**MIPI03**] at **Section 5.1.2.3**).
- **In I3C HDR-DDR Mode** the bytes shall be sent in 2-byte HDR-DDR Data Words, starting with Bytes 0 and 1. The bits within each byte shall be sent according to the HDR-DDR Mode protocol (see [**MIPI03**] at **Section 5.2.2.1**). However, HDR-DDR Multi-Lane transfers might use additional Data Lanes and a modified protocol, per the configured I3C ML Frame Format and Data Transfer Coding (see [**MIPI03**] at **Section 5.3.2.2**).
- **In I3C HDR-BT Mode** the bytes shall be sent in 32-byte HDR-BT Data Blocks, unless the end of the data is ‘ragged’ and requires a Last Data Block. The protocol for HDR-BT data is a function of the number of additional Data Lanes (see [**MIPI03**] at **Section 5.2.4.1**), which depends on the configured I3C ML Frame Format and Data Transfer Coding (see [**MIPI03**] at **Section 5.3.2.4**).

2851

Table 63 OTM Bitwise Transport Mapping for All Frame Formats

I3C Bus Data Byte Order	Raw												RGB					Legacy YUV 420 8bit		YUV 420 8bit		YUV 420 10bit		YUV 422		User Def
	6	7	8	10	12	14	16	20	24	28	444	555	565	666	888	Odd Line	Even Line	Odd Line	Even Line	Odd Line	Even Line	8 bit	10 bit			
Byte 0	P1[0]	P1[0]	P1[0]	P1[0]	P1[0]	P1[0]	P1[0]	P1[0]	P1[0]	P1[0]	B1[0]	B1[0]	B1[0]	B1[0]	B1[0]	U1[0]	V1[0]	Y1[0]	U1[0]	Y1[0]	U1[0]	U1[0]	U1[0]	A[0]		
	P1[1]	P1[1]	P1[1]	P1[1]	P1[1]	P1[1]	P1[1]	P1[1]	P1[1]	P1[1]	B1[1]	B1[1]	B1[1]	B1[1]	B1[1]	U1[1]	V1[1]	Y1[1]	U1[1]	Y1[1]	U1[1]	U1[1]	U1[1]	A[1]		
	P1[2]	P1[2]	P1[2]	P1[2]	P1[2]	P1[2]	P1[2]	P1[2]	P1[2]	P1[2]	B1[2]	B1[2]	B1[2]	B1[2]	B1[2]	U1[2]	V1[2]	Y1[2]	U1[2]	Y1[2]	U1[2]	U1[2]	U1[2]	A[2]		
	P1[3]	P1[3]	P1[3]	P1[3]	P1[3]	P1[3]	P1[3]	P1[3]	P1[3]	P1[3]	B1[3]	B1[3]	B1[3]	B1[3]	B1[3]	U1[3]	V1[3]	Y1[3]	U1[3]	Y1[3]	U1[3]	U1[3]	U1[3]	A[3]		
	P1[4]	P1[4]	P1[4]	P1[4]	P1[4]	P1[4]	P1[4]	P1[4]	P1[4]	P1[4]	G1[0]	B1[4]	B1[4]	B1[4]	B1[4]	U1[4]	V1[4]	Y1[4]	U1[4]	Y1[4]	U1[4]	U1[4]	U1[4]	A[4]		
	P1[5]	P1[5]	P1[5]	P1[5]	P1[5]	P1[5]	P1[5]	P1[5]	P1[5]	P1[5]	P1[5]	G1[1]	G1[0]	G1[0]	B1[5]	B1[5]	U1[5]	V1[5]	Y1[5]	U1[5]	Y1[5]	U1[5]	U1[5]	U1[5]	A[5]	
	P2[0]	P1[6]	P1[6]	P1[6]	P1[6]	P1[6]	P1[6]	P1[6]	P1[6]	P1[6]	P1[6]	G1[2]	G1[1]	G1[1]	G1[0]	B1[6]	U1[6]	V1[6]	Y1[6]	U1[6]	Y1[6]	U1[6]	U1[6]	U1[6]	A[6]	
	P2[1]	P2[0]	P1[7]	P1[7]	P1[7]	P1[7]	P1[7]	P1[7]	P1[7]	P1[7]	G1[3]	G1[2]	G1[2]	G1[1]	B1[7]	U1[7]	V1[7]	Y1[7]	U1[7]	Y1[7]	U1[7]	U1[7]	U1[7]	A[7]		
	P2[2]	P2[0]	P1[8]	P1[8]	P1[8]	P1[8]	P1[8]	P1[8]	P1[8]	P1[8]	R1[0]	G1[3]	G1[3]	G1[2]	G1[0]	Y1[0]	Y1[0]	Y2[0]	Y1[0]	Y1[8]	U1[8]	Y1[0]	Y1[8]	B[0]		
Byte 1	P2[3]	P2[2]	P2[1]	P1[9]	P1[9]	P1[9]	P1[9]	P1[9]	P1[9]	P1[9]	R1[1]	G1[4]	G1[4]	G1[3]	G1[1]	Y1[1]	Y1[1]	Y2[1]	Y1[1]	Y1[9]	U1[9]	Y1[1]	Y1[9]	B[1]		
	P2[4]	P2[3]	P2[2]	P2[0]	P1[10]	P1[10]	P1[10]	P1[10]	P1[10]	P1[10]	R1[2]	R1[0]	G1[5]	G1[4]	G1[2]	Y1[2]	Y1[2]	Y2[2]	Y1[2]	Y2[0]	Y1[0]	Y1[2]	Y1[0]	B[2]		
	P2[5]	P2[4]	P2[3]	P2[1]	P1[11]	P1[11]	P1[11]	P1[11]	P1[11]	P1[11]	R1[3]	R1[1]	R1[0]	G1[5]	G1[3]	Y1[3]	Y1[3]	Y2[3]	Y1[3]	Y2[1]	Y1[1]	Y1[3]	Y1[1]	B[3]		
	P3[0]	P2[5]	P2[4]	P2[2]	P2[0]	P1[12]	P1[12]	P1[12]	P1[12]	P1[12]	B2[0]	R1[2]	R1[1]	R1[0]	G1[4]	Y1[4]	Y1[4]	Y2[4]	Y1[4]	Y2[2]	Y1[2]	Y1[4]	Y1[2]	B[4]		
	P3[1]	P2[6]	P2[5]	P2[3]	P2[1]	P1[13]	P1[13]	P1[13]	P1[13]	P1[13]	B2[1]	R1[3]	R1[2]	R1[1]	G1[5]	Y1[5]	Y1[5]	Y2[5]	Y1[5]	Y2[3]	Y1[3]	Y1[5]	Y1[3]	B[5]		
	P3[2]	P3[0]	P2[6]	P2[4]	P2[2]	P2[0]	P1[14]	P1[14]	P1[14]	P1[14]	B2[2]	R1[4]	R1[3]	R1[2]	G1[6]	Y1[6]	Y1[6]	Y2[6]	Y1[6]	Y2[4]	Y1[4]	Y1[6]	Y1[4]	B[6]		
	P3[3]	P3[1]	P2[7]	P2[5]	P2[3]	P2[1]	P1[15]	P1[15]	P1[15]	P1[15]	B2[3]	B2[0]	R1[4]	R1[3]	G1[7]	Y1[7]	Y1[7]	Y2[7]	Y1[7]	Y2[5]	Y1[5]	Y1[7]	Y1[5]	B[7]		
	P3[4]	P3[2]	P3[0]	P2[6]	P2[4]	P2[2]	P2[0]	P1[16]	P1[16]	P1[16]	G2[0]	B2[1]	B2[0]	R1[4]	R1[0]	Y2[0]	Y3[0]	V1[0]	Y2[6]	Y1[6]	V1[0]	Y1[6]	C[0]			
	P3[5]	P3[3]	P3[1]	P2[7]	P2[5]	P2[3]	P2[1]	P1[17]	P1[17]	P1[17]	G2[1]	B2[2]	B2[1]	R1[5]	R1[1]	Y2[1]	Y3[1]	V1[1]	Y2[7]	Y1[7]	V1[1]	Y1[7]	C[1]			
Byte 2	P4[0]	P3[4]	P3[2]	P2[8]	P2[6]	P2[4]	P2[2]	P1[18]	P1[18]	P1[18]	G2[2]	B2[3]	B2[2]	B2[0]	R1[2]	Y2[2]	Y3[2]	V1[2]	Y2[8]	Y1[8]	V1[2]	Y1[8]	C[2]			
	P4[1]	P3[5]	P3[3]	P2[9]	P2[7]	P2[5]	P2[3]	P1[19]	P1[19]	P1[19]	G2[3]	B2[4]	B2[3]	B2[1]	R1[3]	Y2[3]	Y3[3]	V1[3]	Y2[9]	Y1[9]	V1[3]	Y1[9]	C[3]			
	P4[2]	P3[6]	P3[4]	P3[0]	P2[8]	P2[6]	P2[4]	P2[0]	P1[20]	P1[20]	R2[0]	G2[0]	B2[4]	B2[2]	R1[4]	Y2[4]	Y2[4]	Y3[4]	V1[4]	Y3[0]	V1[0]	V1[4]	V1[0]	C[4]		
	P4[3]	P4[0]	P3[5]	P3[1]	P2[9]	P2[7]	P2[5]	P2[1]	P1[21]	P1[21]	R2[1]	G2[1]	B2[0]	B2[3]	R1[5]	Y2[5]	Y2[5]	Y3[5]	V1[5]	Y3[1]	V1[1]	V1[5]	V1[1]	C[5]		
	P4[4]	P4[1]	P3[6]	P3[2]	P2[10]	P2[8]	P2[6]	P2[2]	P1[22]	P1[22]	R2[2]	G2[2]	B2[1]	B2[4]	R1[6]	Y2[6]	Y2[6]	Y3[6]	V1[6]	Y3[2]	V1[2]	V1[6]	V1[2]	C[6]		
	P4[5]	P4[2]	P3[7]	P3[3]	P2[11]	P2[9]	P2[7]	P2[3]	P1[23]	P1[23]	R2[3]	G2[3]	B2[2]	B2[5]	R1[7]	Y2[7]	Y2[7]	Y3[7]	V1[7]	Y3[3]	V1[3]	V1[7]	V1[3]	C[7]		
	P5[0]	P4[3]	P4[0]	P3[4]	P3[0]	P2[10]	P2[8]	P2[4]	P2[0]	P1[24]	B3[0]	G2[4]	G2[3]	G2[0]	B2[0]	U3[0]	V3[0]	Y4[0]	Y2[0]	Y3[4]	V1[4]	Y2[0]	Y1[4]	D[0]		
	P5[1]	P4[4]	P4[1]	P3[5]	P3[1]	P2[11]	P2[9]	P2[5]	P2[1]	P1[25]	B3[1]	R2[0]	G2[4]	G2[1]	B2[1]	U3[1]	V3[1]	Y4[1]	Y2[1]	Y3[5]	V1[5]	Y2[1]	Y1[5]	D[1]		
	P5[2]	P4[5]	P4[2]	P3[6]	P3[2]	P2[12]	P2[10]	P2[6]	P2[2]	P1[26]	B3[2]	R2[1]	G2[5]	G2[2]	B2[2]	U3[2]	V3[2]	Y4[2]	Y2[2]	Y3[6]	V1[6]	Y2[2]	Y1[6]	D[2]		
Byte 3	P5[3]	P4[6]	P4[3]	P3[7]	P3[3]	P2[13]	P2[11]	P2[7]	P2[3]	P1[27]	B3[3]	R2[2]	R2[0]	G2[3]	B2[3]	U3[3]	V3[3]	Y4[3]	Y2[3]	Y3[7]	V1[7]	Y2[3]	Y1[7]	D[3]		
	P5[4]	P5[0]	P4[4]	P3[8]	P3[4]	P3[0]	P2[12]	P2[8]	P2[4]	P2[0]	G3[0]	R2[3]	R2[1]	G2[4]	B2[4]	U3[4]	V3[4]	Y4[4]	Y2[4]	Y3[8]	V1[8]	Y2[4]	Y1[8]	D[4]		
	P5[5]	P5[1]	P4[5]	P3[9]	P3[5]	P3[1]	P2[13]	P2[9]	P2[5]	P2[1]	G3[1]	R2[4]	R2[2]	G2[5]	B2[5]	U3[5]	V3[5]	Y4[5]	Y2[5]	Y3[9]	V1[9]	Y2[5]	Y1[9]	D[5]		
	P6[0]	P5[2]	P4[6]	P4[0]	P3[6]	P3[2]	P2[14]	P2[10]	P2[6]	P2[2]	G3[2]	B3[0]	R2[3]	R2[0]	B2[6]	U3[6]	V3[6]	Y4[6]	Y2[6]	Y4[0]	Y2[0]	Y2[6]	Y2[0]	D[6]		
	P6[1]	P5[3]	P4[7]	P4[1]	P3[7]	P3[3]	P2[15]	P2[11]	P2[7]	P2[3]	G3[3]	B3[1]	R2[4]	R2[1]	B2[7]	U3[7]	V3[7]	Y4[7]	Y2[7]	Y4[1]	Y2[1]	Y2[7]	Y2[1]	D[7]		

2852

13.2.2 SNS FIFO Requirement

As shown **Figure 201**, the SNS internally stores LP(s) using first-in first-out (FIFO) memory. The FIFO serves to decouple the rate at which the SNS internally produces pixel-bearing CSI-2 packets from the rate at which the VDSP is able to read them using the I3C bus. FIFO implementation details and storage capacity are beyond the scope of this specification.



Figure 201 SNS OTM FIFO

The FIFO shall be empty prior to the start of image streaming. After the SNS finishes writing all the pixel bytes from an LP payload into the FIFO, and the total payload consists of an odd number of bytes, then the SNS shall also write one additional “padding” byte with value 0x00 to the FIFO. The VDSP is able to detect and discard this padding byte upon reception because it knows in advance how many pixel bytes are transported by each OTM LP.

Because the I3C HDR-DDR word size is 16 bits, there is no byte granularity with 16-bit transport for HDR-DDR.

13.2.3 VDSP Initiated Commands to the SNS

As shown in **Table 64**, the following VDSP commands are used to initiate exposure, and to clear and read the SNS FIFO:

- **TPP: Transmit Packet Payload** (Mandatory)
- **OTM**: The VDSP reads all LP payloads corresponding to a single frame from the SNS.
- **DPF: Discard Present Frame** (Mandatory)
 - **OTM**: The VDSP uses **DPF** to direct the SNS to internally discard all remaining, unread bytes of a packet payload, including all payload bytes currently in the FIFO, as well as all payload bytes which have not yet been written to the FIFO. In addition, the **DPF** shall also discard any remaining payload corresponding to the present image frame. The time the image sensor requires to execute this command is implementation-dependent, but should in general be shorter than the time that the VDSP requires to read the data being discarded. The SNS shall ignore (i.e., shall NACK) any **TPP** command(s) received while a **DPF** command is executing; and the SNS shall also notify the VDSP once the **DPF** command execution has finished, using IBI.
- **ODF: On-Demand Frame** (Mandatory)
 - **OTM**: The VDSP uses **ODF** to direct the SNS to initiate frame exposure, and subsequently the SNS generates the Frame Start IBI.

In the I3C specification [**MIP103**], CCC values 0xC0 through 0xC3 have been reserved for CSI-2 AOSC applications, as summarized in **Table 64**.

Table 64 AOSC Operation CCC and Defining Byte Values

CCC Value	0xC0: Imaging & Vision Applications
Defining Byte Value	0x00: Reserved
	0x01: TPP
	0x02: DPF
	0x03: ODF
	0x04 – 0xFF: Reserved

13.2.4 SNS Status Communication to VDSP Using IBI

The SNS shall notify the VDSP via IBI when any of the following events occurs:

1. An image frame is available to be read by the VDSP
2. FIFO overflow, resulting in irretrievable loss of data
3. FIFO underflow, resulting from a fast VDSP-generated I3C clock
4. When a previously received DPF command has finished executing
5. A serious command-related error that will interfere with the SNS' ability to reliably continue transporting image frame data

The SNS shall support I3C IBI “Mandatory Data Byte” (MDB[7:5] = 3’b010) to communicate the IBI status. The SNS Mandatory Data Byte provides the VDSP additional information about the IBI. The SNS shall use the MDB[4:0]=5’h00 for OTM and subsequent payload(s) for the six mandatory operations and one optional operation (see **Table 65**). Upon generation of an IBI, the SNS may also provide status via CCI-compliant registers accessible by the VDSP.

Table 65 AOSC OTM Low-Latency Mandatory IBI MDB Codes

MDB[7:0]	Payload (Interrupt Group ID)	Field	Support	Description
8'h40	8'h00	Frame Start	Mandatory	Frame start indication from SNS, applicable only for new image frame transfer.
	8'h01	DPF Complete	Mandatory	Indication from SNS that DPF command has successfully completed and the internal pipelines are flushed.
	8'h02	Internal Error	Mandatory	Indication from SNS that an internal error occurred, and any remaining LPs mapped to the present frame will be flushed.
	8'h03	Frame Restart	Mandatory	Indication from SNS transmission of a new frame after completing DPF or encountering an internal error.
	8'h04	FIFO Underrun	Mandatory	Indication from SNS that the circular FIFO has encountered an underrun condition. Present frame transport may be paused.
	8'h05	FIFO Overrun	Mandatory	Indication from SNS that the circular FIFO has encountered an overrun condition. Such a condition may manifest as a result of slower VDSP bus transfer rate. This notification may be deemed as <i>fatal</i> by VDSP, due to loss of data.
	8'h06	Manufacturer Defined	Optional	One or more payload bytes to follow the Interrupt Group ID, to specify Manufacturer Defined Interrupts which may include Motion detection, ALS trigger, and FIFO hysteresis information.
8'h07 –8'hFF	Reserved	–	–	–

2897

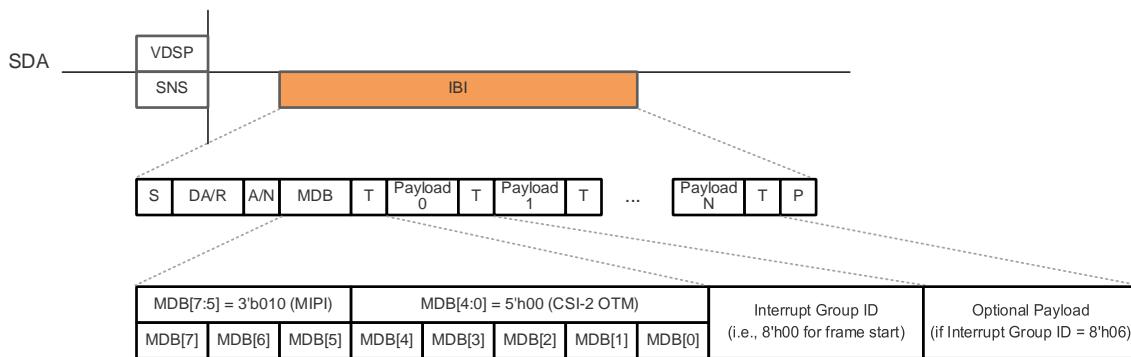


Figure 202 AOSC IBI MDB Codes

13.2.5 Support for On-Demand Frame and Streaming Frames

The SNS shall provide the option to generate either an On-Demand Frame (ODF) or Continuously Streaming Frames (CSF) to the VDSP. If the ODF option is enabled, then the SNS shall wait for an ODF command from the VDSP prior to initiating exposure and generating an In-Band-Interrupt (IBI) mapped to beginning of an image frame transfer (see *Figure 199*).

- Bit **REG_AOSC_CONTROL[3]** (field **EXPOSURE_CSF_ODF**) shall be used to select between ODF and CSF operation:
 - **1'b0: ODF Mode:** The SNS requires an ODF command from the VDSP to begin exposure
 - **1'b1: CSF Mode:** The SNS does not require an ODF command from the VDSP to begin exposure

13.2.6 Support for Frame Squelching

The frame squelching feature facilitates dynamic frame exposure duration. The SNS shall provide the option to squelch frames while in CSF mode, by generating interleaved Frame Start (FS) IBI as defined below. The SNS does not provide the option to squelch frames while in ODF mode. The VDSP shall handle frame slip(s) while the SNS is squelching frames. There may be frame slips due to PVT variations or jitter accumulation in the SNS.

- Bit **REG_AOSC_CONTROL[4]** (field **SQUELCH_FRAMES**) is used to enable frame squelching while in CSF mode:
 - **1'b1:** The SNS shall squelch Frame Start IBI as defined by register **REG_AOSC_SQ_FS_IBI[7:0]** (below).
 - **1'b0:** The SNS shall not squelch Frame Start IBI.
- Bits **REG_AOSC_SQ_FS_IBI[7:0]** (field **SQUELCH_FRAME_START_IBI**) control when the SNS generates an FS IBI:
 - **8'd0:** The SNS shall generate an FS IBI every second frame
 - **8'd1:** The SNS shall generate an FS IBI every third frame
 - ...
 - **8'dN:** The SNS shall generate an FS IBI every (**N+2**) frames
 - ...
 - **8'd255:** SNS shall generate FS IBI every 257th frame
- Bits **REG_AOSC_SQ_FS_IBI[15:8]** are reserved for future use.

13.2.7 Support for Interconnect Synchronization and Dynamic ISPB Insertion

An SNS supporting OTM shall support Dynamic Interconnect Synchronizing Padding Bytes (ISPB) insertion. The SNS may generate the ISPB using the same (pixel) clock that is used to generate the OTM Long Packet. The system may optionally utilize the ISPB capability to help align the SNS-generated pixel clock with the VDSP-generated I3C clock.

- Bit **REG_AOSC_CONTROL[5]** (field **ISPB_Insertion**) shall be used to enable ISPB:
 - 1'b0: Dynamic ISPB insertion is disabled
 - 1'b1: Dynamic ISPB insertion is enabled

The dynamic ISPB consists of zero, or a non-zero even number of, 8'h00 bytes, immediately followed by two 8'hFF bytes. When enabled, the SNS shall generate a dynamic ISPB (which may consist of only the two 8'hFF bytes) after each LP in an image frame except for the last LP (see *Figure 203*). Due to the LP padding byte defined in *Section 13.2.2*, the beginning of the ISPB is always 16-bit word-aligned. The length of the dynamic ISPB shall be determined by the SNS. The number of ISPB bytes transmitted after each LP may be automatically varied by the SNS as needed in order to prevent FIFO underrun or overrun.

Once the OTM Frame Start IBI is initiated, the VDSP shall initiate TPP within the T_RESP duration as illustrated in *Figure 203*. The duration of T_RESP is implementation specific and should be included in the AOSC SNS data sheet.

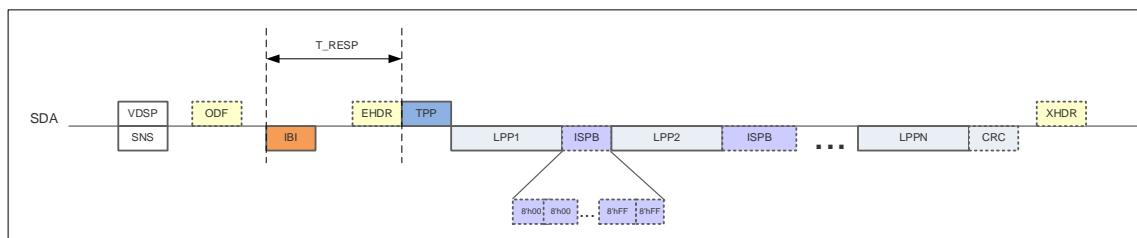


Figure 203 OTM ISPB Insertion and Frame Start IBI T_RESP Parameter

The AOSC SNS should support **REG_AOSC_ISPB_IBI_WM[15:0]** to help facilitate appropriate VDSP transport link rate. The VDSP configures **REG_AOSC_ISPB_IBI_WM[15:0]** (field **ISPB IBI WATERMARK**) with one of the FIFO watermark levels shown below in order to direct the SNS as to when it should generate the frame-start IBI, based on FIFO fullness. An SNS datasheet may restrict watermark support to only a subset of these levels:

- {12'd0, 4'b0000}: The SNS shall generate an IBI when FIFO is not empty
- {12'd0, 4'b0001}: The SNS shall generate an IBI when FIFO WATERMARK is 1/16th full
- {12'd0, 4'b0010}: The SNS shall generate an IBI when FIFO WATERMARK is 2/16th full
- ...
- {4'b0100, 12'd0}: The SNS shall generate IBI when FIFO WATERMARK is 15/16th full
- {4'b1000, 12'd0}: The SNS shall generate IBI when FIFO is full

13.2.8 OTM Errors Detected by the VDSP

The VDSP may use the CSI-2 16-bit CRC described in [Section 9.6](#) to detect packet payload errors; however, this form of error detection, while highly reliable, requires the VDSP to wait until the CRC has been received at the end of the payload before determining whether an error occurred.

When the OTM CRC is enabled, the SNS initializes it to 0xFFFF at the start of every image frame and updates it with every pixel byte, padding byte, and ISPB read by the VDSP during the frame. The CRC is conceptually updated one byte at a time over the entire image frame using the serial algorithm shown in the C-code function of [Figure 69](#); note that each byte is conceptually shifted LS-bit-first into the corresponding CRC shift register shown in [Figure 68](#). The SNS transfers the calculated CRC over the I3C Bus in MS-byte-first order.

The OTM CRC shall not be generated by the SNS if the HDR-BT mode is selected for I3C transfers. This is because the HDR-BT mode includes its own mandatory 16-bit CRC and optional 32-bit CRC calculations.

13.3 Smart Transport Mode (STM) Overview

AOSC STM support is optional. STM allows the SNS to transport payload over a single I3C IBI without the need for a Read Request from the VDSP. The AOSC STM is limited to the I3C SDR data rate. As shown in *Figure 204*, the structure for STM may include the equivalent of the Long Packet Structure for D-PHY Physical Layer Option shown in *Figure 52*, but here the structure is considered solely as payload, with the D-PHY generic STM types where **STM Type** is set to **D-PHY Generic Use 0** or **D-PHY Generic Use 1** as shown in *Table 66*.

Note that the **BCR[2]** bit in the I3C SNS's BCR register shall be set to 1'b1 in order to take advantage of transporting one or more data bytes (MDB).

Figure 204 illustrates STM operation for the D-PHY generic STM types.

Note:

- The MDB defined for STM in *Table 66* differs from the one shown in *Table 65* for OTM.
- STM operations in the dotted boxes are optional.

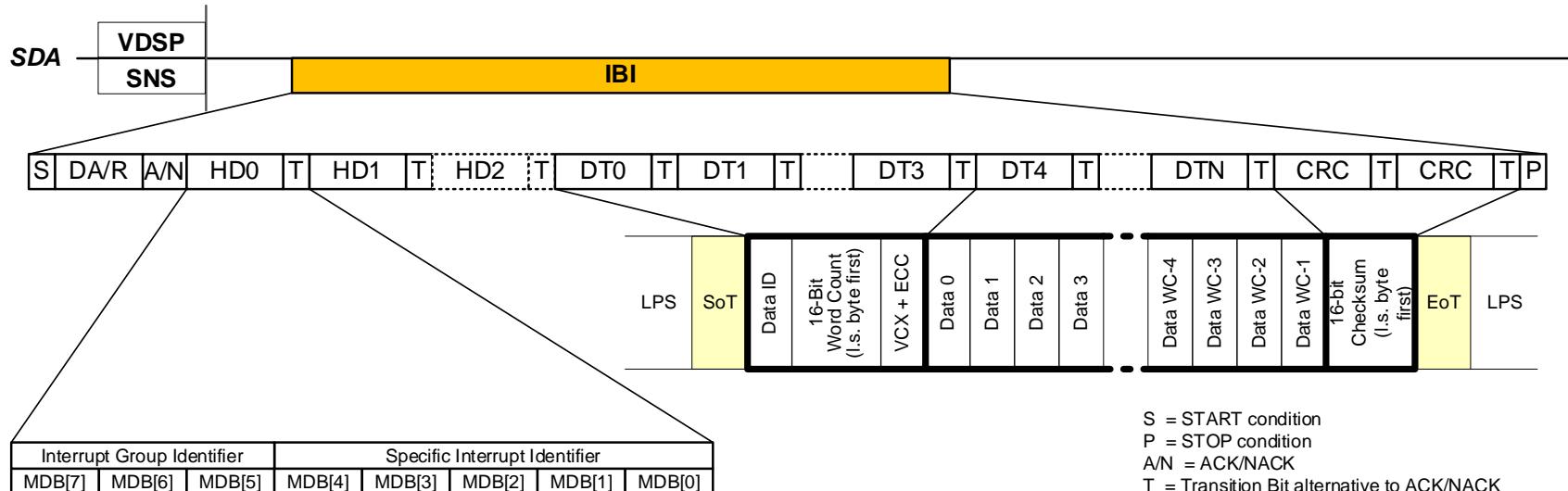


Figure 204 AOSC Smart Transport Mode (STM) Operation for the D-PHY Generic STM Types

- **IBI:** In-Band Interrupt generated by the SNS at the beginning of an image line or frame
- The IBI consists of the following data:
- **DAR:** The Dynamic Address with RnW bit set to 1'b1 (i.e., **READ**)
 - **HD0:** MDB as the first Header Byte for STM shown in *Table 66*
 - **MDB:** The Mandatory Data Byte of the IBI payload is the data that follows the Dynamic Address when the SNS sends an IBI request:
 - **MDB[7:5]:** Interrupt Group Identifier
 - **3'b010: MIPI Groups**
 - **MDB[4:0]:** Specific Interrupt Identifier Value
 - **5'h01: AOSC Transport Mode for STM**
 - **HD1:** The second Header Byte that defines the Word Count Extension Enable (**WCX_EN**) and the STM Type for the STM specific payload data, which provides some flexibility depending on the applications:
 - **Bit[7]:** Reserved for future use
 - **WCX_EN[6]: Word Count Extension Enable**
 - **1'b0:** The Word Count Extension is disabled, so the next Header Byte **HD2** may be omitted. Only **DT1** and **DT2** are valid as 16-bit Word Count (**WC**).
 - **1'b1:** the Word Count Extension is enabled, so the next Header Byte **HD2** represents the most significant byte of the 24-bit Word Count (**WC**) along with **DT1** and **DT2**
 - **STMTYP[5:0]: STM Type**
 - **6'h00:** Reserved for future use
 - **6'h01–6'h1F:** User Defined 1–31
 - **6'h20:** D-PHY Generic Use 0, which is equivalent to Long Packet Structure for D-PHY Physical Layer Option shown in *Figure 52*, while the 32-bit Packet Header (**PH**) element is considered as payload in STM. The Word Count Extension (**WCX_EN[6]**) is ignored in this case, or handled as 1'b0 as if it's disabled so that the following third Header Byte (**HD2**) is omitted.
 - **6'h21:** D-PHY Generic Use 1, which is defined as an extension of D-PHY Generic Use 0 above that allows the Word Count Extension, making the Word Count (**WC**) a 24-bit (not 16-bit) field in order to support transport of more data than D-PHY Generic Use 0.
 - **6'h22–6'h3F:** Reserved for future use
 - **HD2:** The third Header Byte, which defines the most significant byte of the 24-bit Word Count (**WC**) along with **DT1** and **DT2** when **WCX_EN[6]** = 1'b1
 - **WCX[7:0]:** The most significant byte of the 24-bit Word Count (**WC**)
 - **DT0:** The 8-bit Data Identifier (**DI**) as the first byte of the 32-bit Packet Header (**PH**) shown in *Figure 52* when **STMTYP[5:0]** = 6'h20 or 6'h21
 - **VC[7:6]:** The 2-bit Virtual Channel (**VC**) that defines the least significant two bits of the 4-bit Virtual Channel Identifier for the D-PHY physical layer option
 - **DT[5:0]:** The 6-bit Data Type that denotes the format/content of the Application Specific Payload Data used by the application specific layer
 - **DT1:** The least significant byte of the 16-bit or 24-bit Word Count (**WC**) as the second byte of the 32-bit Packet Header (**PH**) shown in *Figure 52* when **STMTYP[5:0]** = 6'h20 or 6'h21
 - **WC[7:0]:** The least significant byte of the 16-bit or 24-bit Word Count (**WC**)
 - **DT2:** The second least significant byte of the 16-bit or 24-bit Word Count (**WC**) as the third byte of the 32-bit Packet Header (**PH**) shown in *Figure 52* when **STMTYP[5:0]** = 6'h20 or 6'h21
 - **WC[15:8]:** The second least significant byte of the 16-bit or 24-bit Word Count (**WC**)

- **DT3:** The 6-bit Error Correction Code (**ECC**) and the 2-bit Virtual Channel Extension (**VCX**) as the fourth byte of the 32-bit Packet Header (**PH**) shown in *Figure 52* when **STMTYP[5:0]** = 6'h20 or 6'h21
- **ECC[5:0]:** The 6-bit ECC enables 1-bit errors within the packet header to be corrected, and 2-bit errors to be detected
- **VCX[1:0]:** The 2-bit Virtual Channel Extension (**VCX**) that defines the most significant two bits of the 4-bit Virtual Channel Identifier for the D-PHY physical layer option
- **DT4–DTN:** The SNS transports the Application Specific Payload Data shown in *Figure 52* when **STMTYP[5:0]** = 6'h20 or 6'h21
- **CRC:** The SNS generates the 16-bit Checksum/CRC (the least significant byte first) shown in *Figure 52* when **STMTYP[5:0]** = 6'h20 or 6'h21

See *Section 9.1.1* for further descriptions.

Table 66 Header Definitions for AOSC Smart Transport Mode (STM)

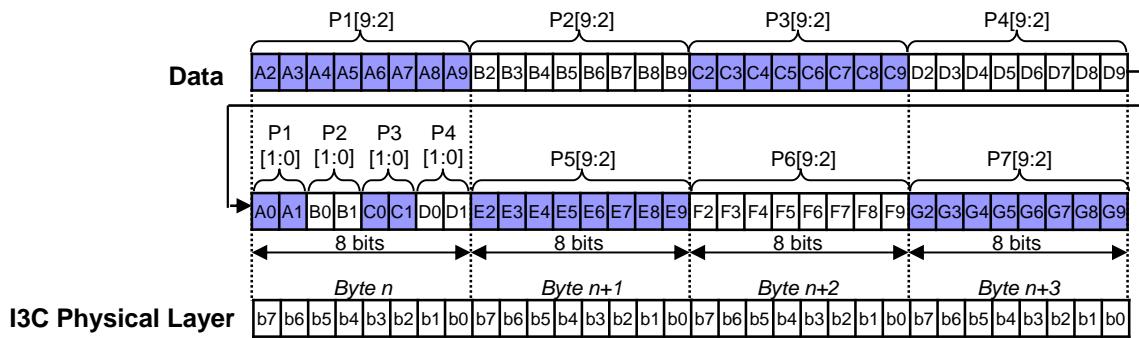
Header Byte	Bits	Field	Values
0 (HD0)	MDB[7:5]	MDB Interrupt Group Identifier	3'b010: MIPI Groups
	MDB[4:0]	MDB Specific Interrupt Identifier Value	5'h00: AOSC Transport Mode for OTM 5'h01: AOSC Transport Mode for STM 5'h02–5'h1B: MIPI Alliance Reserved 5'h1C–5'h1D: MIPI Alliance Debug WG Reserved 5'h1E–5'h1F: MIPI Alliance Reserved
1 (HD1)	Bit[7]	–	Reserved
	WCX_EN[6]	Word Count Extension Enable	1'b0: Word Count Extension Disabled 1'b1: Word Count Extension Enabled
	STMTYP[5:0]	STM Type	6'h00: Reserved 6'h01–6'h1F: User Defined 1 - 31 6'h20: D-PHY Generic Use 0 6'h21: D-PHY Generic Use 1 6'h22–6'h3F: Reserved
2 (HD2: Optional)	WCX[7:0]	Word Count Extension	8'h00–8'hFF: Most significant byte of the 24-bit Word Count when WCX_EN[6] = 1'b1

3035 AOSC STM shall follow the data packing defined in **Section 11** with the D-PHY generic STM types, when
 3036 **STMTYP[5:0]** = 6'h20 or 6'h21.

3037 **Figure 205** illustrates an example of RAW10 format bitwise transport with the D-PHY generic STM types
 3038 above.

3039 **Note:**

3040 *Data packing for the D-PHY generic STM types differs from that shown in **Table 63** for OTM.*



3041 **Figure 205 Example AOSC STM RAW10 Data Bitwise Transport Illustration with the D-PHY Generic STM Types**

This page intentionally left blank.

Annex A JPEG8 Data Format (informative)

A.1 Introduction

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

JPEG8 has two non-standard extensions:

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

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

The JPEG8 data flow is illustrated in **Figure 206** and **Figure 207**.

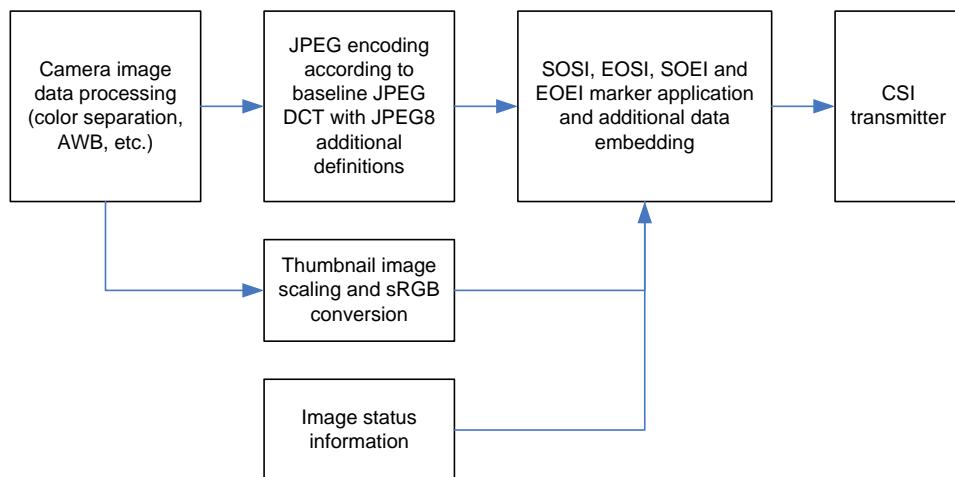


Figure 206 JPEG8 Data Flow in the Encoder

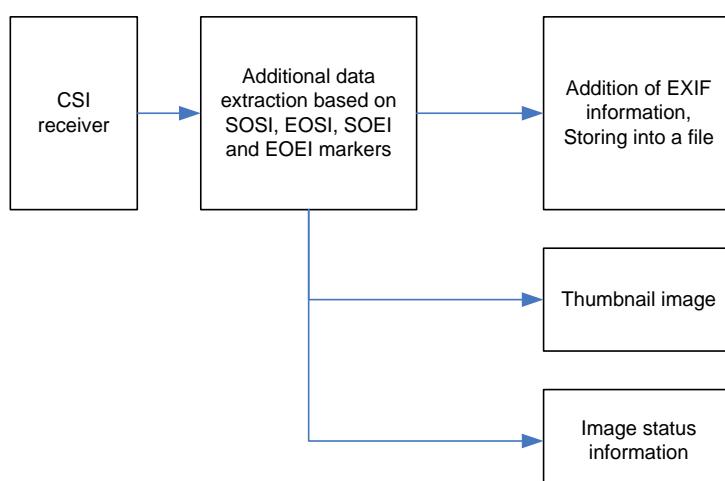


Figure 207 JPEG8 Data Flow in the Decoder

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 208**.
- A status line is added in the end of JPEG data as defined in **Section A.3**.
- If needed, an embedded image is interlaced in order which is free of choice as defined in **Section A.4**.
- Prior to storing into a file, the CSI-2 JPEG data is processed by the data separation process described in **Section A.1**.

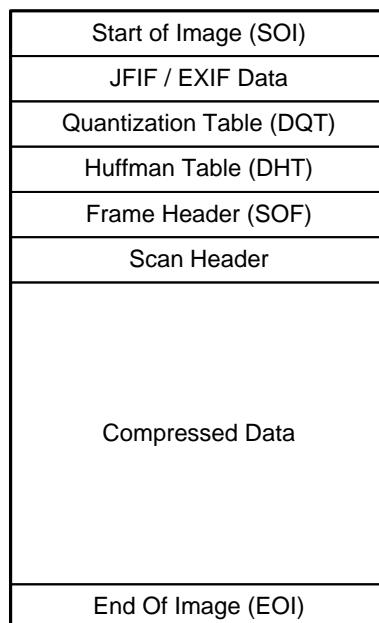


Figure 208 EXIF Compatible Baseline JPEG DCT Format

A.3 Image Status Information

Information of at least the following items has to be stored in the end of the JPEG sequence as illustrated in **Figure 209**:

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

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

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

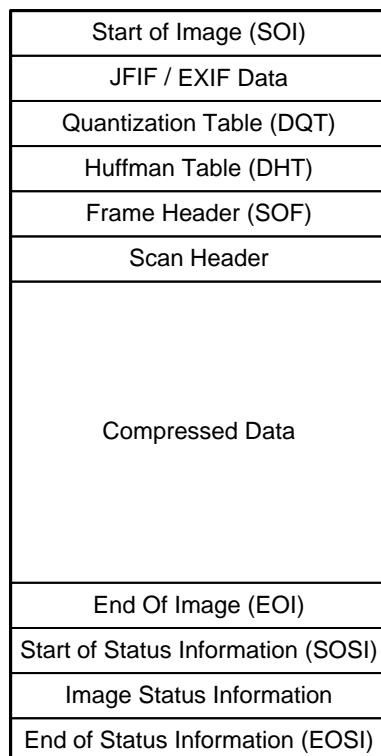
The status information content has to be defined in the product specification of each camera module containing the JPEG8 feature. The format and content is manufacturer specific.

The image status data should be arranged so that each byte is split into two 4-bit nibbles and "1010" padding sequence is added to MSB, as presented in **Table 67**. This ensures that no JPEG escape sequences (0xFF 0x00) are present in the status data.

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

Table 67 Status Data Padding

Data Word	After Padding
D7D6D5D4 D3D2D1D0	1010D7D6D5D4 1010D3D2D1D0



3090

Figure 209 Status Information Field in the End of Baseline JPEG Frame

A.4 Embedded Images

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

The philosophy of embedded / interleaved thumbnail additions is to minimize the needed frame memory. The EI (Embedded Image) data can be included in any part of the compressed image data segment and in as many pieces as needed. See **Figure 210**.

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

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

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

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

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

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

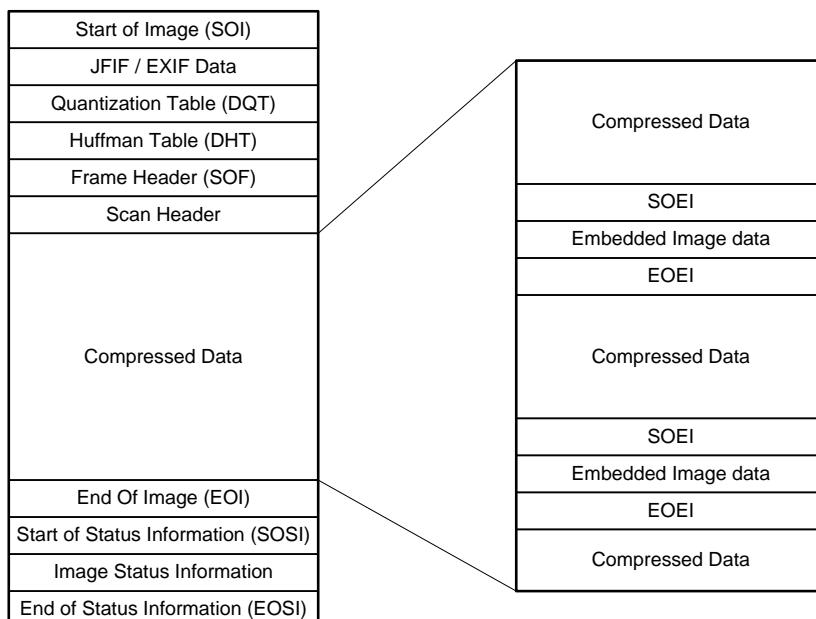


Figure 210 Example of TN Image Embedding Inside the Compressed JPEG Data Block

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 68 JPEG8 Additional Marker Codes Listing

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

A.6 JPEG8 Data Reception

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

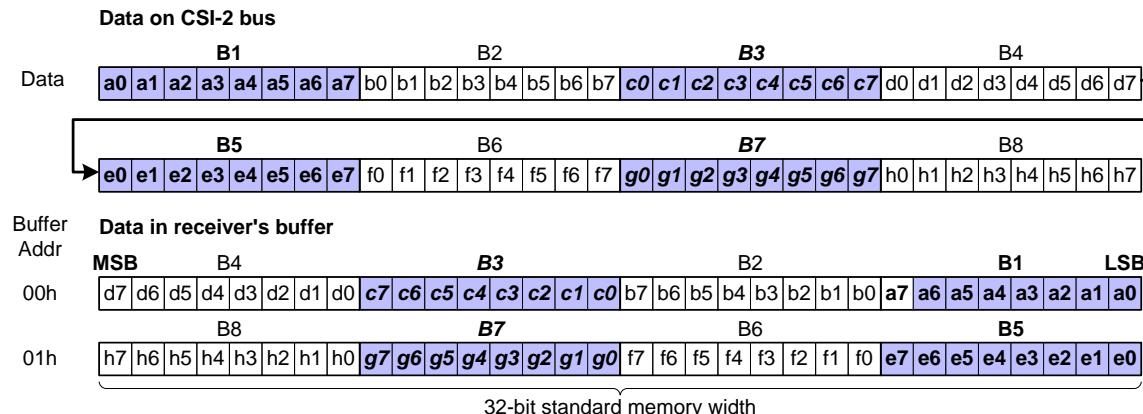


Figure 211 JPEG8 Data Format Reception

Annex B CSI-2 Implementation Example (informative)

B.1 Overview

The CSI-2 implementation example assumes that the interface comprises of D-PHY unidirectional Clock and Data, with forward Escape Mode and optional deskew functionality. The scope in this implementation example refers only to the unidirectional data link without any references to the CCI interface, as it can be seen in *Figure 212*. This implementation example varies from the informative PPI example in [MIPI01].

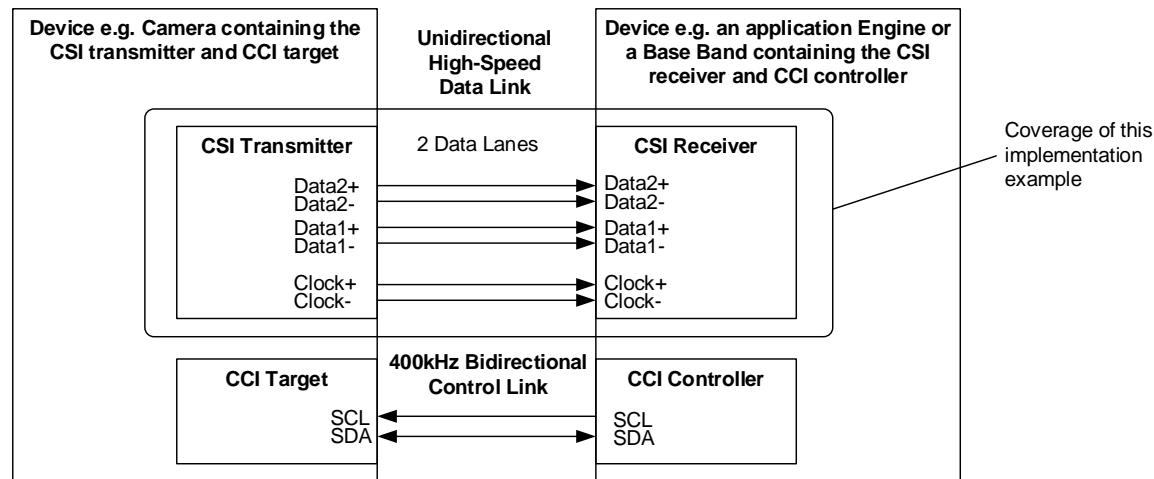


Figure 212 Implementation Example Block Diagram and Coverage

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

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

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

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

B.2 CSI-2 Transmitter Detailed Block Diagram

3136
3137

Using the layered structure described in the overview the CSI-2 transmitter could have the block diagram in **Figure 213**.

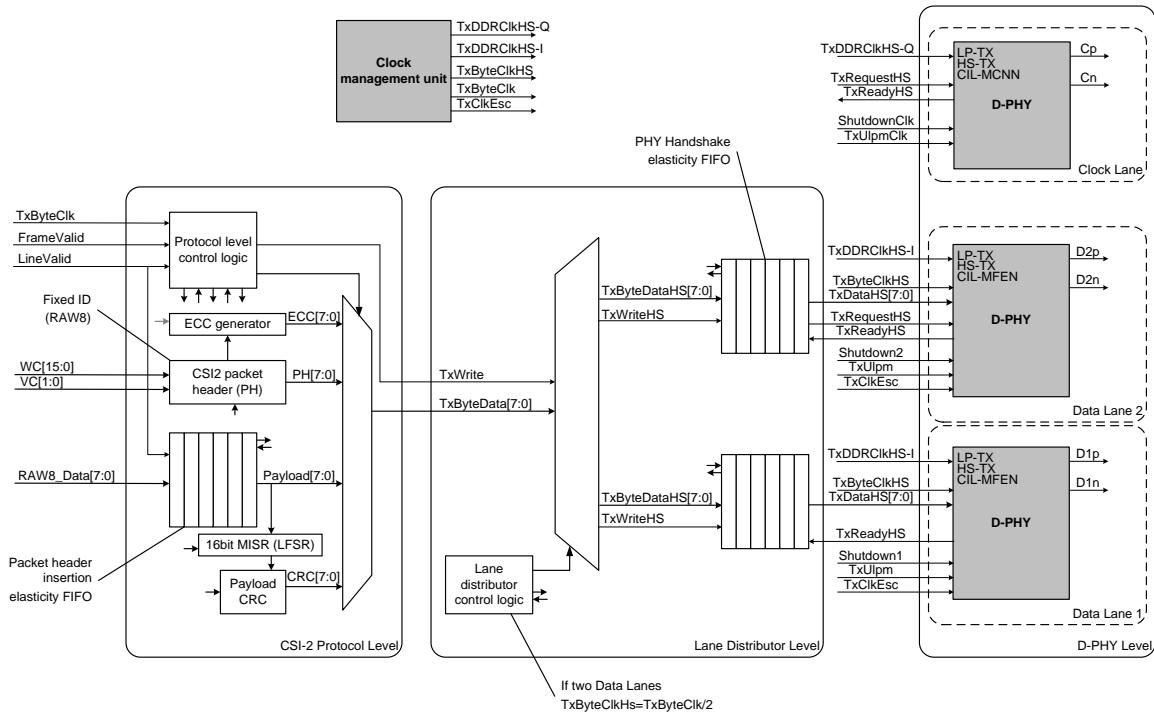


Figure 213 CSI-2 Transmitter Block Diagram

B.3 CSI-2 Receiver Detailed Block Diagram

3139 Using the layered structure described in the overview, the CSI-2 receiver could have the block diagram in
3140 **Figure 214**.

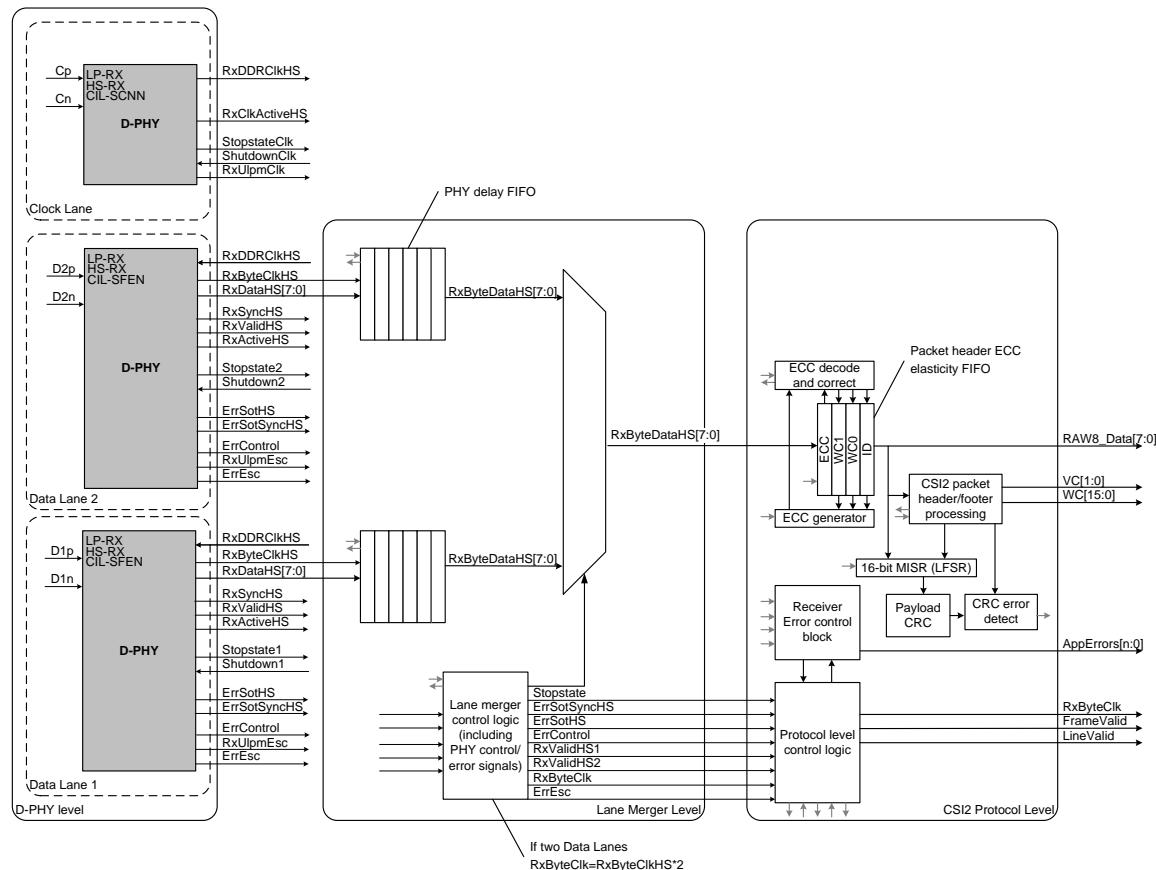


Figure 214 CSI-2 Receiver Block Diagram

B.4 Details on the D-PHY Implementation

3142

The PHY level of implementation has the top level structure as seen in *Figure 215*.

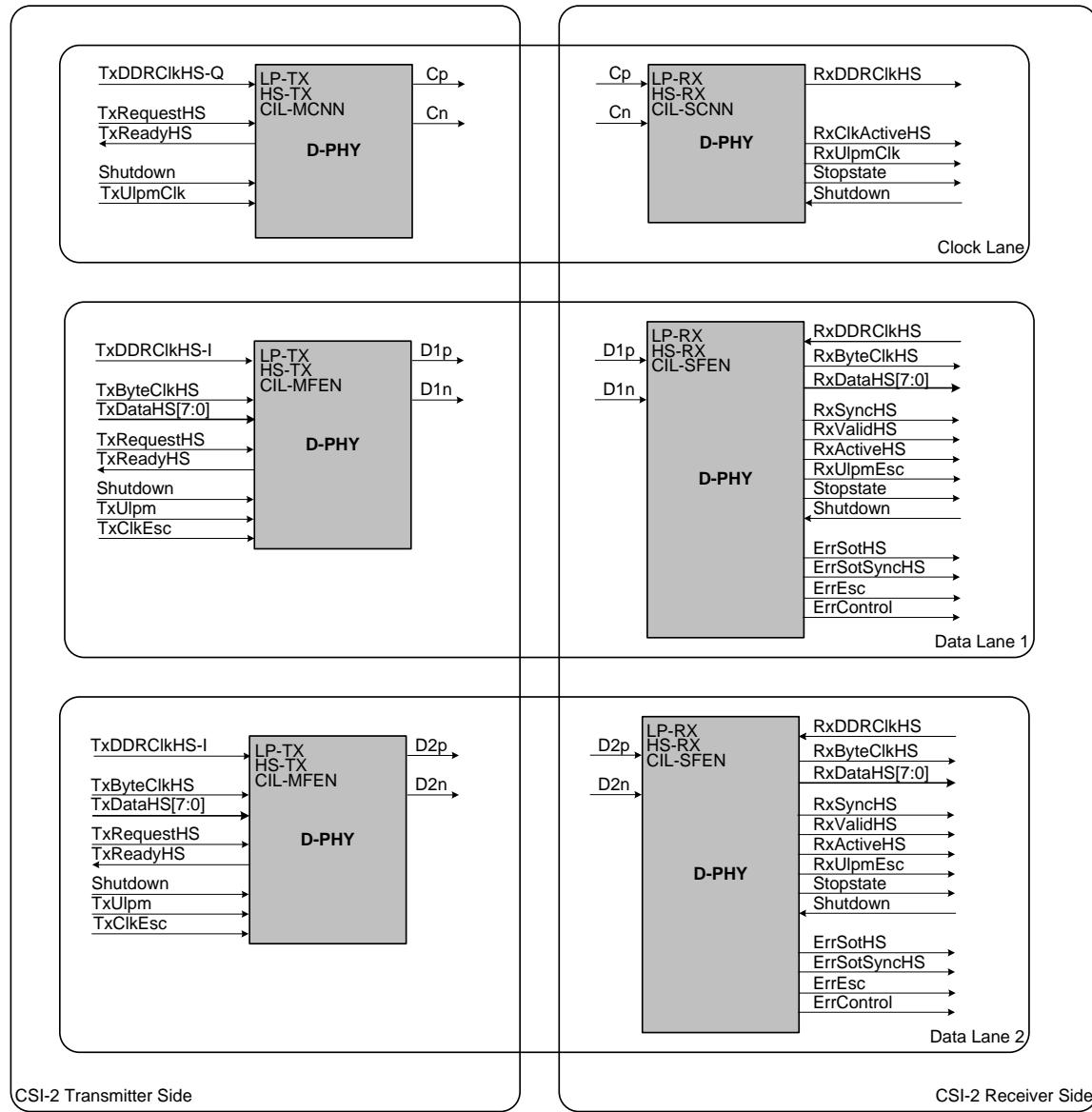


Figure 215 D-PHY Level Block Diagram

3144

The components can be categorized as:

3145

- CSI-2 Transmitter side:

3146

- Clock lane (Transmitter)
- Data1 lane (Transmitter)
- Data2 lane (Transmitter)

3147

- CSI-2 Receiver side:

3148

- Clock lane (Receiver)
- Data1 lane (Receiver)

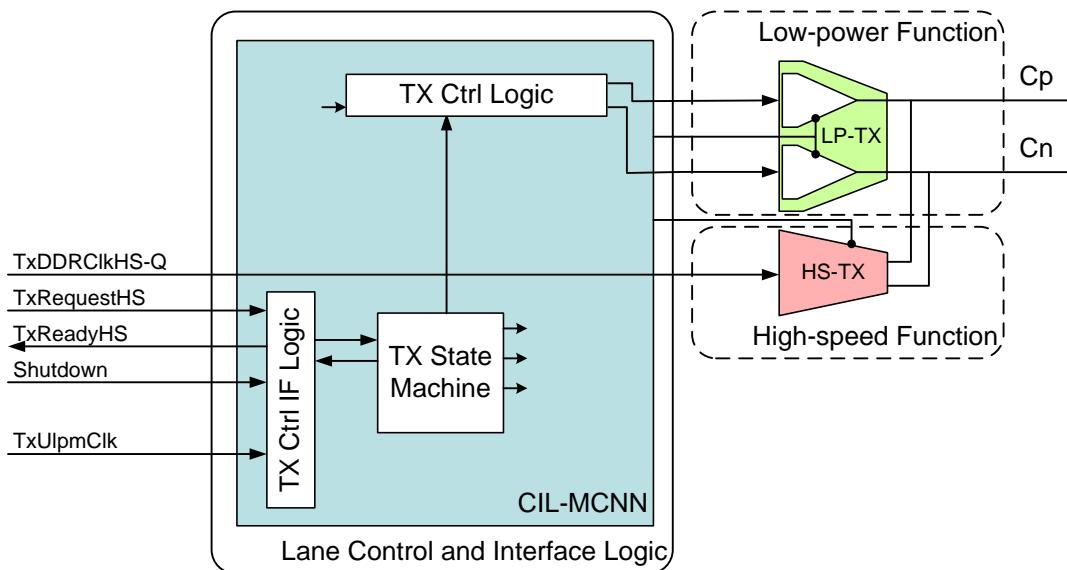
3149

258

- 3152 • Data2 lane (Receiver)

B.4.1 CSI-2 Clock Lane Transmitter

3153 The suggested implementation can be seen in *Figure 216*.



3154 **Figure 216 CSI-2 Clock Lane Transmitter**

3155 The modular D-PHY components used to build a CSI-2 clock lane transmitter are:

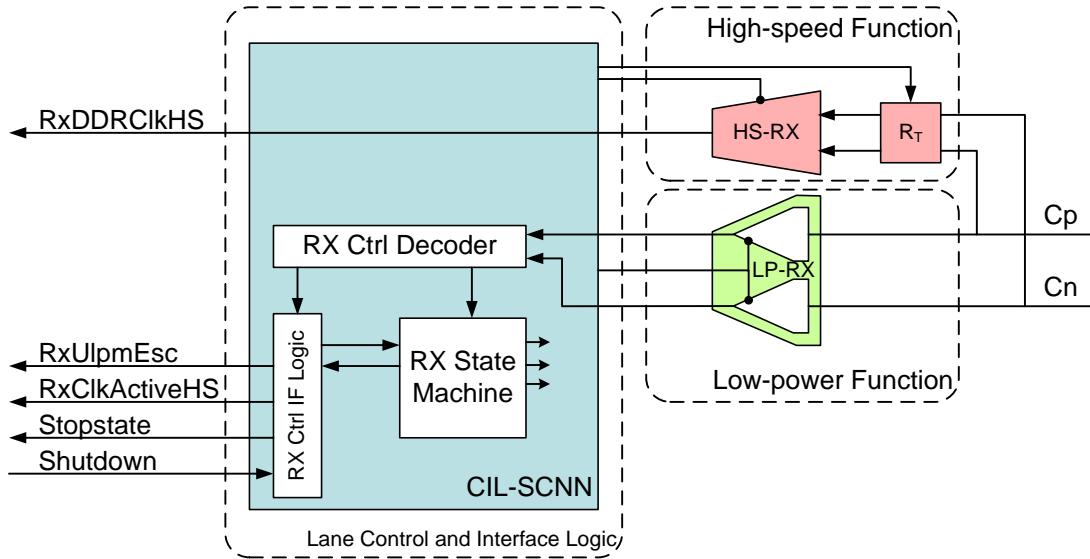
- 3156 • **LP-TX** for the Low-power function
- 3157 • **HS-TX** for the High-speed function
- 3158 • **CIL-MCNN** for the Lane control and interface logic

3159 The PPI interface signals to the CSI-2 clock lane transmitter are:

- 3160 • **TxDDRClkHS-Q** (Input): High-Speed Transmit DDR Clock (Quadrature).
- 3161 • **TxRequestHS** (Input): High-Speed Transmit Request. This active high signal causes the lane module to begin transmitting a high-speed clock.
- 3162 • **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that the clock lane is transmitting HS clock.
- 3163 • **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.
- 3164 • **TxUlpmClk** (Input): Transmit Ultra Low-Power mode on Clock Lane This active high signal is asserted to cause a Clock Lane module to enter the Ultra Low-Power mode. The lane module remains in this mode until TxUlpmClk is de-asserted.

B.4.2 CSI-2 Clock Lane Receiver

3173 The suggested implementation can be seen in *Figure 217*.



3174 **Figure 217 CSI-2 Clock Lane Receiver**

3175 The modular D-PHY components used to build a CSI-2 clock lane receiver are:

- 3176 • **LP-RX** for the Low-power function
- 3177 • **HS-RX** for the High-speed function
- 3178 • **CIL-SCNN** for the Lane control and interface logic

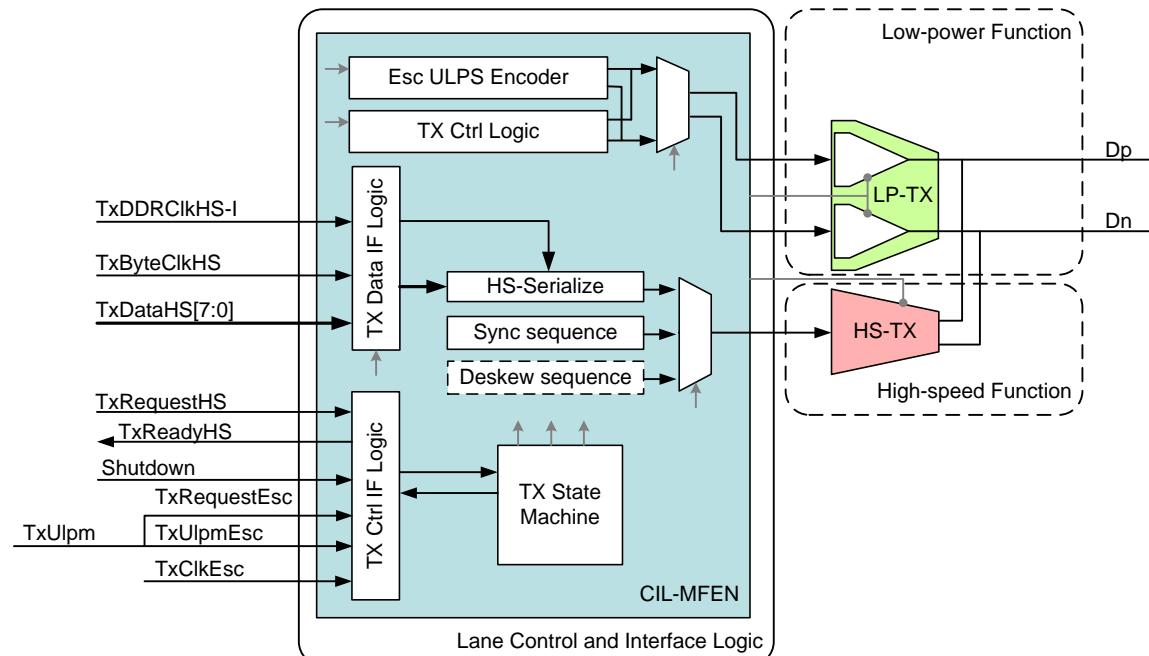
3179 The PPI interface signals to the CSI-2 clock lane receiver are:

- 3180 • **RxDDRClkHS** (Output): High-Speed Receive DDR Clock used to sample the data in all data
lanes.
- 3181 • **RxClkActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the
clock lane is receiving valid clock. This signal is asynchronous.
- 3182 • **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.
- 3183 • **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.
- 3184 • **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

3194

The suggested implementation can be seen in **Figure 218**.



3195

Figure 218 CSI-2 Data Lane Transmitter

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

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

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

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

- 3219 • **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that
3220 TxDataHS is accepted by the lane module to be serially transmitted. TxReadyHS is valid on rising
3221 edges of TxByteClkHS. Valid data has to be provided for the whole duration of active
3222 TxReadyHS.
- 3223 • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
3224 “shutdown”, disabling all activity. All line drivers, including terminators, are turned off when
3225 Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs
3226 are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on
3227 any clock.
- 3228 • **TxUlpmEsc** (Input): Escape Mode Transmit Ultra Low Power. This active high signal is asserted
3229 with TxRequestEsc to cause the lane module to enter the Ultra Low-Power mode. The lane
3230 module remains in this mode until TxRequestEsc is de-asserted.
- 3231 • **TxRequestEsc** (Input): This active high signal, asserted together with TxUlpmEsc is used to
3232 request entry into Escape Mode. Once in Escape Mode, the lane stays in Escape Mode until
3233 TxRequestEsc is de-asserted. TxRequestEsc is only asserted by the protocol while TxRequestHS
3234 is low.
- 3235 • **TxClkEsc** (Input): Escape Mode Transmit Clock. This clock is directly used to generate escape
3236 sequences. The period of this clock determines the symbol time for low power signals. It is
3237 therefore constrained by the normative part of the *[MIPI01]*.

B.4.4 CSI-2 Data Lane Receiver

3238

The suggested implementation can be seen in *Figure 219*.

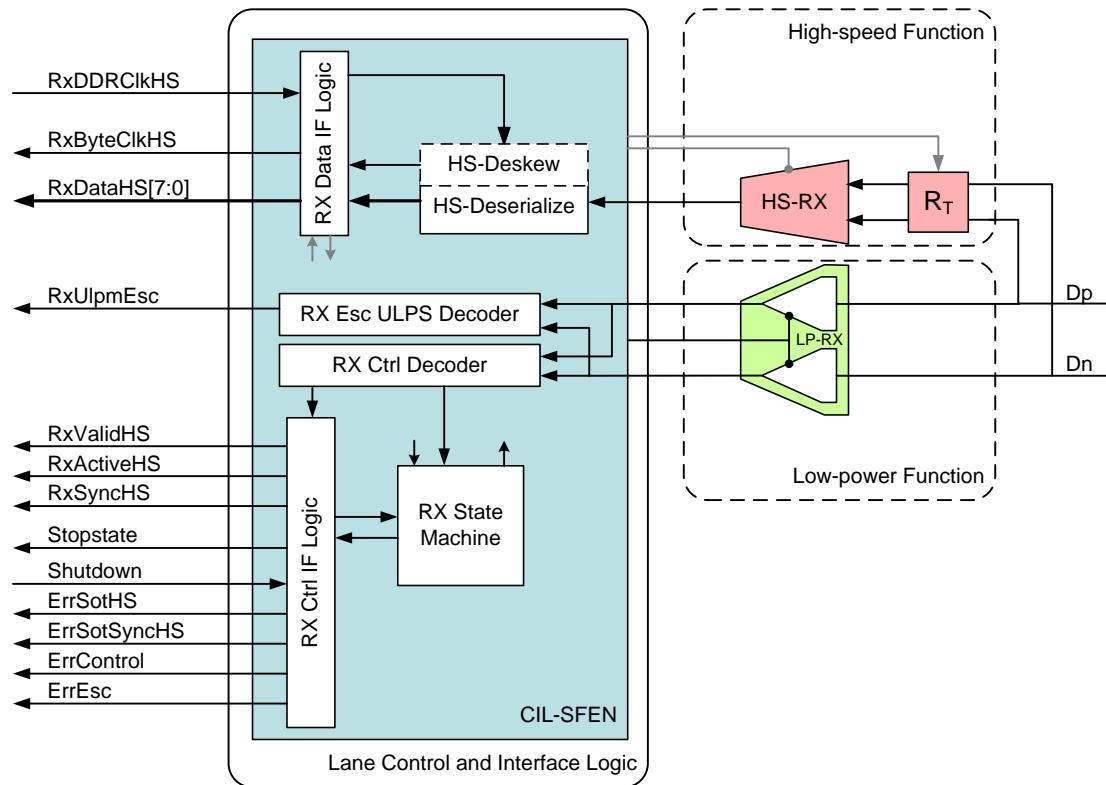


Figure 219 CSI-2 Data Lane Receiver

3239

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.

- 3260 • **RxActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the
3261 lane module is actively receiving a high-speed transmission from the lane interconnect.
- 3262 • **RxSyncHS** (Output): Receiver Synchronization Observed. This active high signal indicates that
3263 the lane module has seen an appropriate synchronization event. In a typical high-speed
3264 transmission, RxSyncHS is high for one cycle of RxByteClkHS at the beginning of a high-speed
3265 transmission when RxActiveHS is first asserted. This signal missing is signaled using
3266 ErrSotSyncHS.
- 3267 • **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is
3268 asserted to indicate that the lane module has entered the Ultra Low-Power mode. The lane module
3269 remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane
3270 interconnect.
- 3271 • **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is
3272 currently in Stop state. This signal is asynchronous.
- 3273 • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
3274 “shutdown”, disabling all activity. All line drivers, including terminators, are turned off when
3275 Shutdown is asserted. When Shutdown is high, all PPI outputs are driven to the default inactive
3276 state. Shutdown is a level sensitive signal and does not depend on any clock.
- 3277 • **ErrSotHS** (Output): Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is
3278 corrupted, but in such a way that proper synchronization can still be achieved, this error signal is
3279 asserted for one cycle of RxByteClkHS. This is considered to be a “soft error” in the leader
3280 sequence and confidence in the payload data is reduced.
- 3281 • **ErrSotSyncHS** (Output): Start-of-Transmission Synchronization Error. If the high-speed SoT
3282 leader sequence is corrupted in a way that proper synchronization cannot be expected, this error is
3283 asserted for one cycle of RxByteClkHS.
- 3284 • **ErrControl** (Output): Control Error. This signal is asserted when an incorrect line state sequence
3285 is detected.
- 3286 • **ErrEsc** (Output): Escape Entry Error. If an unrecognized escape entry command is received, this
3287 signal is asserted and remains high until the next change in line state. The only escape entry
3288 command supported by the receiver is the ULPS.

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

C.1 Overview

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

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

- **D-PHY Level errors**

Refers to any PHY related transmission error and is unrelated to the transmission’s contents:

- Start of Transmission (SoT) errors, which can be:

- Recoverable, if the PHY successfully identifies the Sync code but an error was detected.
- Unrecoverable, if the PHY does not successfully identify the sync code but does detect a HS transmission.

- **Control Error**, which signals that the PHY has detected a control sequence that should not be present in this implementation of the Link.

- **Packet Level errors**

This type of error refers strictly to data integrity of the received Packet Header and payload data:

- **Packet Header errors**, signaled through the ECC code, that result in:

- A single bit-error, which can be detected and corrected by the ECC code
- Two bit-errors in the header, which can be detected but not corrected by the ECC code, resulting in a corrupt header

- **Packet payload errors**, signaled through the CRC code

- **Protocol Decoding Level errors**

This type of error refers to errors present in the decoded Packet Header or errors resulting from an incomplete sequence of events:

- **Frame Sync Error**, caused when a FS could not be successfully paired with a FE on a given virtual channel

- **Unrecognized ID**, caused by the presence of an unimplemented or unrecognized ID in the header

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

C.2 D-PHY Level Error

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

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

The recommended receiver error behavior for this level is:

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

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

C.3 Packet Level Error

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

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

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

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

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

The recommended receiver error behavior for this level is:

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

C.4 Protocol Decoding Level Error

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

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

The recommended receiver error behavior for this level is:

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

Annex D CSI-2 Sleep Mode (informative)

D.1 Overview

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

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

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

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

In general, when in SLM, both sides of the interface will be in ULPS, as defined in [\[MIPI01\]](#) or [\[MIPI02\]](#).

D.2 SLM Command Phase

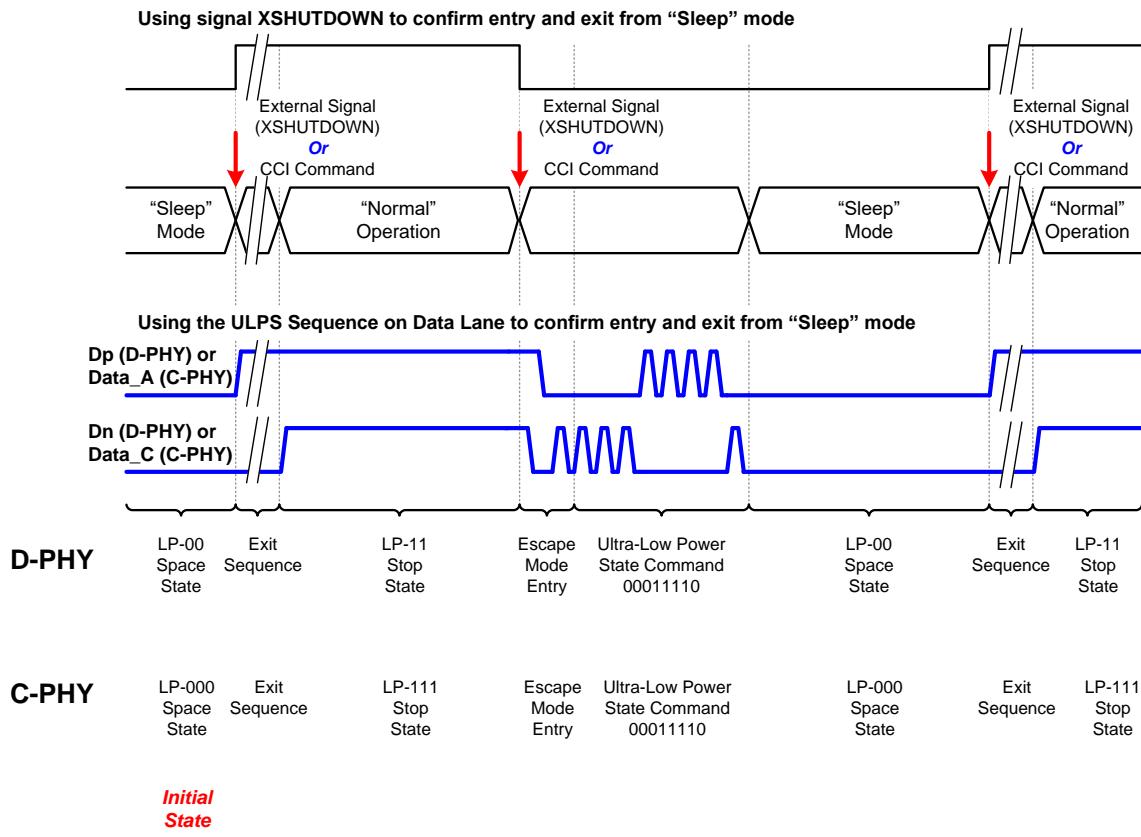
For the first phase, initiation of SLM occurs by a mechanism outside the scope of CSI-2. Of the many mechanisms available, two examples would be:

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

D.3 SLM Entry Phase

3433 For the second phase, consider one option:

3434 Only the TX side enters SLM and propagates the ULPS to the RX side by sending a D-PHY or C-PHY
 3435 ‘ULPS’ command on each Lane. In **Figure 220**, only the Data Lane ‘ULPS’ command is used as an example.
 3436 The D-PHY D_p, D_n, and C-PHY Data_A, Data_C are logical signal names and do not imply specific
 3437 multiplexing on dual mode (combined D-PHY and C-PHY) implementations.



3438 **Figure 220 SLM Synchronization**

D.4 SLM Exit Phase

3439 For the third phase, three options are presented and assume the camera peripheral is in ULPS or Sleep mode
 3440 at power-up:

- 3441 1. Use a SLEEP signal to power-up both sides of the interface.
- 3442 2. Detect any CCI activity on the I²C control Link, which was in the 00 state ({SCL, SDA}), after
 3443 receiving the I²C instruction to enter ULPS command as per **Section D.2**, option 2. Any change on
 3444 those lines should wake up the camera peripheral. The drawback of this method is that I²C lines
 3445 are used exclusively for control of the camera.
- 3446 3. Detect a wake-up sequence on the I²C lines. This sequence, which may vary by implementation,
 3447 shall not disturb the I²C interface so that it can be used by other devices. One example sequence is:
 3448 StopI2C-StartI2C-StopI2C. See **Section 6** for details on CCI.

3449 A handshake using the ‘ULPS’ mechanism as described in **[MIPI01]** or **[MIPI02]**, as appropriate, should be
 3450 used for powering up the interface.

Annex E Data Compression for RAW Data Types (normative)

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

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

The following data compression schemes are defined:

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

To identify the type of data on the CSI-2 interface, packets with compressed data shall have a User Defined Data Type value as indicated in *Table 61*. Note that User Defined data type codes are not reserved for compressed data types. Therefore, a CSI-2 device shall be able to communicate over the CCI the data compression scheme represented by a particular User Defined data type code for each scheme supported by the device. Note that the method to communicate the data compression scheme to Data Type code mapping is beyond the scope of this document.

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

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

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

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

3485 The data compression system consists of encoder, decoder and predictor blocks as shown in **Figure 221**.

3486

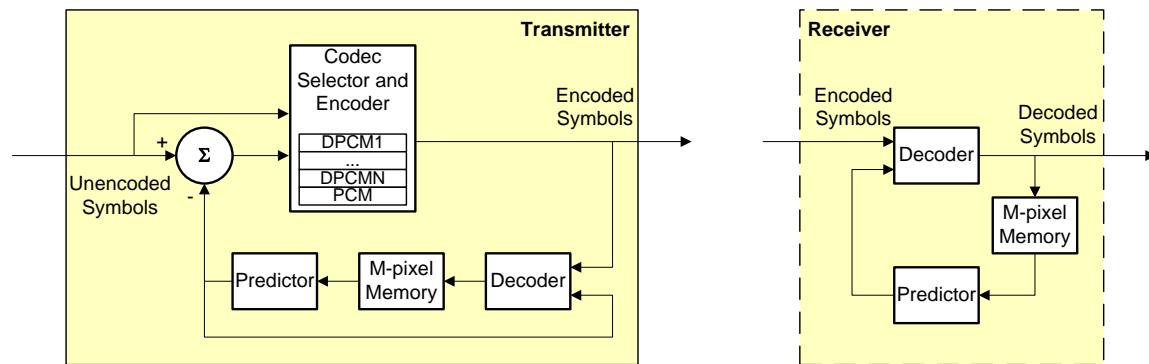


Figure 221 Data Compression System Block Diagram

3487 The encoder uses a simple algorithm to encode the pixel values. A fixed number of pixel values at the
 3488 beginning of each line are encoded without using prediction. These first few values are used to initialize the
 3489 predictor block. The remaining pixel values on the line are encoded using prediction.

3490 If the predicted value of the pixel (**Xpred**) is close enough to the original value of the pixel (**Xorig**)
 3491 ($\text{abs}(\text{Xorig} - \text{Xpred}) < \text{difference limit}$), its difference value (**Xdiff**) is quantized using a DPCM codec.
 3492 Otherwise, **Xorig** is quantized using a PCM codec. The quantized value is combined with a code word
 3493 describing the codec used to quantize the pixel and the sign bit, if applicable, to create the encoded value
 3494 (**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 222*.

C0 ₀	C1 ₁	C0 ₂	C1 ₃	C0 ₄	C1 ₅	C0 ₆	C1 ₇	...
C2 ₀	C3 ₁	C2 ₂	C3 ₃	C2 ₄	C3 ₅	C2 ₆	C3 ₇	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 222 Pixel Order of the Original Image

Figure 223 shows an example of the pixel order with RGB data.

G ₀	R ₁	G ₂	R ₃	G ₄	R ₅	G ₆	R ₇	...
B ₀	G ₁	B ₂	G ₃	B ₄	G ₅	B ₆	G ₇	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 223 Example Pixel Order of the Original Image

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

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

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

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

E.1.1 Predictor1

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

The first two pixels (C0₀, C1₁ / C2₀, C3₁ or as in example G₀, R₁ / B₀, G₁) in a line are encoded without prediction.

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

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

E.1.2 Predictor2

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

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

The first pixel (C_{0_0} / C_{2_0} , or as in example G_0 / B_0) in a line is coded without prediction.

The second pixel (C_{1_1} / C_{3_1} or as in example R_1 / G_1) in a line is predicted using the previous decoded different color value as a prediction value. The second pixel is predicted with the following equation:

$$X_{pred}(n) = X_{deco}(n-1)$$

The third pixel (C_{0_2} / C_{2_2} or as in example G_2 / B_2) in a line is predicted using the previous decoded same color value as a prediction value. The third pixel is predicted with the following equation:

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

The fourth pixel (C_{1_3} / C_{3_3} or as in example R_3 / G_3) in a line is predicted using the following equation:

```
if ((Xdeco(n-1) <= Xdeco(n-2) AND Xdeco(n-2) <= Xdeco(n-3)) OR
    (Xdeco(n-1) >= Xdeco(n-2) AND Xdeco(n-2) >= Xdeco(n-3))) then
    Xpred(n) = Xdeco(n-1)
else
    Xpred(n) = Xdeco(n-2)
endif
```

Other pixels in all lines are predicted using the equation:

```
if ((Xdeco(n-1) <= Xdeco(n-2) AND Xdeco(n-2) <= Xdeco(n-3)) OR
    (Xdeco(n-1) >= Xdeco(n-2) AND Xdeco(n-2) >= Xdeco(n-3))) then
    Xpred(n) = Xdeco(n-1)
else if ((Xdeco(n-1) <= Xdeco(n-3) AND Xdeco(n-2) <= Xdeco(n-4)) OR
         (Xdeco(n-1) >= Xdeco(n-3) AND Xdeco(n-2) >= Xdeco(n-4))) then
    Xpred(n) = Xdeco(n-2)
else
    Xpred(n) = (Xdeco(n-2) + Xdeco(n-4) + 1) / 2
endif
```

E.2 Encoders

3546 There are seven different encoders available, one for each data compression scheme.

3547 For all encoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
3548 for predicted pixels.

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

3549 The 10–8–10 coder offers a 20% bit rate reduction with very high image quality.

3550 Pixels without prediction are encoded using the following formula:

```
3551     Xenco( n ) = Xorig( n ) / 4
```

3552 To avoid a full-zero encoded value, the following check is performed:

```
3553     if (Xenco( n ) == 0) then
3554         Xenco( n ) = 1
3555     endif
```

3556 Pixels with prediction are encoded using the following formula:

```
3557     if (abs(Xdiff( n )) < 32) then
3558         use DPCM1
3559     else if (abs(Xdiff( n )) < 64) then
3560         use DPCM2
3561     else if (abs(Xdiff( n )) < 128) then
3562         use DPCM3
3563     else
3564         use PCM
3565     endif
```

E.2.1.1 DPCM1 for 10–8–10 Coder

3566 **Xenco(n)** has the following format:

```
3567     Xenco( n ) = "00 s xxxxx"
```

3568 where,

```
3569     "00" is the code word
3570     "s" is the sign bit
3571     "xxxxx" is the five bit value field
```

3572 The coder equation is described as follows:

```
3573     if (Xdiff( n ) <= 0) then
3574         sign = 1
3575     else
3576         sign = 0
3577     endif
3578     value = abs(Xdiff( n ))
```

3579 Note: Zero code has been avoided (0 is sent as -0).

E.2.1.2 DPCM2 for 10–8–10 Coder

3580 **Xenco(n)** has the following format:

3581 **Xenco(n)** = "010 s xxxx"

3582 where,

3583 "010" is the code word

3584 "s" is the **sign** bit

3585 "xxxx" is the four bit **value** field

3586 The coder equation is described as follows:

```
3587       if (Xdiff( n ) < 0) then
3588           sign = 1
3589       else
3590           sign = 0
3591       endif
3592       value = (abs(Xdiff( n )) - 32) / 2
```

E.2.1.3 DPCM3 for 10–8–10 Coder

3593 **Xenco(n)** has the following format:

3594 **Xenco(n)** = "011 s xxxx"

3595 where,

3596 "011" is the code word

3597 "s" is the **sign** bit

3598 "xxxx" is the four bit **value** field

3599 The coder equation is described as follows:

```
3600       if (Xdiff( n ) < 0) then
3601           sign = 1
3602       else
3603           sign = 0
3604       endif
3605       value = (abs(Xdiff( n )) - 64) / 4
```

E.2.1.4 PCM for 10–8–10 Coder

3606 **Xenco(n)** has the following format:

3607 **Xenco(n)** = "1 xxxxxxxx"

3608 where,

3609 "1" is the code word

3610 the **sign** bit is not used

3611 "xxxxxxxx" is the seven bit **value** field

3612 The coder equation is described as follows:

3613 **value** = **Xorig(n)** / 8

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

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

Pixels without prediction are encoded using the following formula:

```
3614     Xenco( n ) = Xorig( n ) / 8
```

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

```
3618     if (Xenco( n ) == 0) then
3619         Xenco( n ) = 1
```

Pixels with prediction are encoded using the following formula:

```
3620     if (abs(Xdiff( n )) < 8) then
3621         use DPCM1
3622     else if (abs(Xdiff( n )) < 16) then
3623         use DPCM2
3624     else if (abs(Xdiff( n )) < 32) then
3625         use DPCM3
3626     else if (abs(Xdiff( n )) < 160) then
3627         use DPCM4
3628     else
3629         use PCM
3630     endif
```

E.2.2.1 DPCM1 for 10–7–10 Coder

Xenco(n) has the following format:

```
3632     Xenco( n ) = "000 s xxx"
```

where,

```
3633     "000" is the code word
3634     "s" is the sign bit
3635     "xxx" is the three bit value field
```

The coder equation is described as follows:

```
3636     if (Xdiff( n ) <= 0) then
3637         sign = 1
3638     else
3639         sign = 0
3640     endif
3641     value = abs(Xdiff( n ))
```

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

E.2.2.2 DPCM2 for 10-7-10 Coder

Xenco(n) has the following format:

```
3647     Xenco( n ) = "0010 s xx"
```

where,

```
3649     "0010" is the code word
3650     "s" is the sign bit
3651     "xx" is the two bit value field
```

The coder equation is described as follows:

```
3652     if (Xdiff( n ) < 0) then
3653         sign = 1
3654     else
3655         sign = 0
3656     endif
3657     value = (abs(Xdiff( n )) - 8) / 2
```

E.2.2.3 DPCM3 for 10-7-10 Coder

Xenco(n) has the following format:

```
3659     Xenco( n ) = "0011 s xx"
```

where,

```
3662     "0011" is the code word
3663     "s" is the sign bit
3664     "xx" is the two bit value field
```

The coder equation is described as follows:

```
3665     if (Xdiff( n ) < 0) then
3666         sign = 1
3667     else
3668         sign = 0
3669     endif
3670     value = (abs(Xdiff( n )) - 16) / 4
```

E.2.2.4 DPCM4 for 10-7-10 Coder

Xenco(n) has the following format:

```
3673     Xenco( n ) = "01 s xxxx"
```

where,

```
3675     "01" is the code word
3676     "s" is the sign bit
3677     "xxxx" is the four bit value field
```

The coder equation is described as follows:

```
3678     if (Xdiff( n ) < 0) then
3679         sign = 1
3680     else
3681         sign = 0
3682     endif
3683     value = (abs(Xdiff( n )) - 32) / 8
```

E.2.2.5 PCM for 10-7-10 Coder

3685 **Xenco(n)** has the following format:

3686 **Xenco(n)** = "1 xxxxxx"

3687 where,

3688 "1" is the code word

3689 the **sign** bit is not used

3690 "xxxxxx" is the six bit **value** field

3691 The coder equation is described as follows:

3692 **value** = **Xorig(n)** / 16

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

3693 The 10–6–10 coder offers 40% bit rate reduction with acceptable image quality.

3694 Pixels without prediction are encoded using the following formula:

3695 **Xenco(n) = Xorig(n) / 16**

3696 To avoid a full-zero encoded value, the following check is performed:

```
3697 if (Xenco( n ) == 0) then
3698     Xenco( n ) = 1
3699 endif
```

3700 Pixels with prediction are encoded using the following formula:

```
3701 if (abs(Xdiff( n )) < 1) then
3702     use DPCM1
3703 else if (abs(Xdiff( n )) < 3) then
3704     use DPCM2
3705 else if (abs(Xdiff( n )) < 11) then
3706     use DPCM3
3707 else if (abs(Xdiff( n )) < 43) then
3708     use DPCM4
3709 else if (abs(Xdiff( n )) < 171) then
3710     use DPCM5
3711 else
3712     use PCM
3713 endif
```

E.2.3.1 DPCM1 for 10–6–10 Coder

3714 **Xenco(n)** has the following format:

3715 **Xenco(n) = "00000 s"**

3716 where,

3717 "00000" is the code word
3718 "s" is the **sign** bit
3719 the **value** field is not used

3720 The coder equation is described as follows:

```
3721 sign = 1
3722 Note: Zero code has been avoided (0 is sent as -0).
```

E.2.3.2 DPCM2 for 10–6–10 Coder

3723 **Xenco(n)** has the following format:

3724 **Xenco(n) = "00001 s"**

3725 where,

3726 "00001" is the code word
3727 "s" is the **sign** bit
3728 the **value** field is not used

3729 The coder equation is described as follows:

```
3730 if (Xdiff( n ) < 0) then
3731     sign = 1
3732 else
3733     sign = 0
3734 endif
```

E.2.3.3 DPCM3 for 10–6–10 Coder

Xenco(n) has the following format:

```
3735     Xenco( n ) = "0001 s x"
```

where,

```
3738     "0001" is the code word
3739     "s" is the sign bit
3740     "x" is the one bit value field
```

The coder equation is described as follows:

```
3741     if (Xdiff( n ) < 0) then
3742         sign = 1
3743     else
3744         sign = 0
3745     value = (abs(Xdiff( n )) - 3) / 4
3747 endif
```

E.2.3.4 DPCM4 for 10–6–10 Coder

Xenco(n) has the following format:

```
3748     Xenco( n ) = "001 s xx"
```

where,

```
3751     "001" is the code word
3752     "s" is the sign bit
3753     "xx" is the two bit value field
```

The coder equation is described as follows:

```
3755     if (Xdiff( n ) < 0) then
3756         sign = 1
3757     else
3758         sign = 0
3759     endif
3760     value = (abs(Xdiff( n )) - 11) / 8
```

E.2.3.5 DPCM5 for 10–6–10 Coder

Xenco(n) has the following format:

```
3762     Xenco( n ) = "01 s xxx"
```

where,

```
3764     "01" is the code word
3765     "s" is the sign bit
3766     "xxx" is the three bit value field
```

The coder equation is described as follows:

```
3768     if (Xdiff( n ) < 0) then
3769         sign = 1
3770     else
3771         sign = 0
3772     endif
3773     value = (abs(Xdiff( n )) - 43) / 16
```

E.2.3.6 PCM for 10–6–10 Coder

3774 **Xenco(n)** has the following format:

3775 **Xenco(n)** = "1 xxxx"

3776 where,

3777 "1" is the code word

3778 the **sign** bit is not used

3779 "xxxxx" is the five bit **value** field

3780 The coder equation is described as follows:

3781 **value** = **Xorig(n)** / 32

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

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

Pixels without prediction are encoded using the following formula:

```
3784     Xenco( n ) = Xorig( n ) / 4
```

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

```
3786     if (Xenco( n ) == 0) then
3787         Xenco( n ) = 1
3788     endif
```

Pixels with prediction are encoded using the following formula:

```
3790     if (abs(Xdiff( n )) < 128) then
3791         use DPCM1
3792     else if (abs(Xdiff( n )) < 256) then
3793         use DPCM2
3794     else if (abs(Xdiff( n )) < 512) then
3795         use DPCM3
3796     else
3797         use PCM
3798     endif
```

E.2.4.1 DPCM1 for 12-10-12 Coder

Xenco(n) has the following format:

```
3800     Xenco( n ) = "00 s xxxxxxx"
```

where,

```
3802     "00" is the code word
3803     "s" is the sign bit
3804     "xxxxxxxx" is the seven bit value field
```

The coder equation is described as follows:

```
3806     if (Xdiff( n ) <= 0) then
3807         sign = 1
3808     else
3809         sign = 0
3810     endif
3811     value = abs(Xdiff( n ))
```

Note:

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

E.2.4.2 DPCM2 for 12-10-12 Coder

Xenco(n) has the following format:

```
3814      Xenco( n ) = "010 s xxxxxxx"
```

where,

```
3815      "010" is the code word
3816      "s" is the sign bit
3817      "xxxxxx" is the six bit value field
```

The coder equation is described as follows:

```
3821      if (Xdiff( n ) < 0) then
3822          sign = 1
3823      else
3824          sign = 0
3825      endif
3826      value = (abs(Xdiff( n )) - 128) / 2
```

E.2.4.3 DPCM3 for 12-10-12 Coder

Xenco(n) has the following format:

```
3827      Xenco( n ) = "011 s xxxxxxx"
```

where,

```
3829      "011" is the code word
3830      "s" is the sign bit
3831      "xxxxxx" is the six bit value field
```

The coder equation is described as follows:

```
3834      if (Xdiff( n ) < 0) then
3835          sign = 1
3836      else
3837          sign = 0
3838      endif
3839      value = (abs(Xdiff( n )) - 256) / 4
```

E.2.4.4 PCM for 12-10-12 Coder

Xenco(n) has the following format:

```
3840      Xenco( n ) = "1 xxxxxxxxxxx"
```

where,

```
3843      "1" is the code word
3844      the sign bit is not used
3845      "xxxxxxxx" is the nine bit value field
```

The coder equation is described as follows:

```
3846      value = Xorig( n ) / 8
```

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

3848 The 12–8–12 coder offers 33% bit rate reduction with very high image quality.

3849 Pixels without prediction are encoded using the following formula:

3850 **Xenco(n) = Xorig(n) / 16**

3851 To avoid a full-zero encoded value, the following check is performed:

```
3852   if (Xenco( n ) == 0) then  
3853       Xenco( n ) = 1  
3854   endif
```

3855 Pixels with prediction are encoded using the following formula:

```
3856   if (abs(Xdiff( n )) < 8) then  
3857       use DPCM1  
3858   else if (abs(Xdiff( n )) < 40) then  
3859       use DPCM2  
3860   else if (abs(Xdiff( n )) < 104) then  
3861       use DPCM3  
3862   else if (abs(Xdiff( n )) < 232) then  
3863       use DPCM4  
3864   else if (abs(Xdiff( n )) < 360) then  
3865       use DPCM5  
3866   else  
3867       use PCM
```

E.2.5.1 DPCM1 for 12–8–12 Coder

3868 **Xenco(n)** has the following format:

3869 **Xenco(n) = "0000 s xxx"**

3870 where,

3871 "0000" is the code word
3872 "s" is the **sign** bit
3873 "xxx" is the three bit **value** field

3874 The coder equation is described as follows:

```
3875   if (Xdiff( n ) <= 0) then  
3876       sign = 1  
3877   else  
3878       sign = 0  
3879   endif  
3880   value = abs(Xdiff( n ))
```

3881 Note: Zero code has been avoided (0 is sent as -0).

E.2.5.2 DPCM2 for 12–8–12 Coder

Xenco(n) has the following format:

```
3882     Xenco( n ) = "011 s xxxx"
```

where,

```
3885     "011" is the code word
3886     "s" is the sign bit
3887     "xxxx" is the four bit value field
```

The coder equation is described as follows:

```
3888     if (Xdiff( n ) < 0) then
3889         sign = 1
3890     else
3891         sign = 0
3892     endif
3893     value = (abs(Xdiff( n )) - 8) / 2
```

E.2.5.3 DPCM3 for 12–8–12 Coder

Xenco(n) has the following format:

```
3895     Xenco( n ) = "010 s xxxx"
```

where,

```
3898     "010" is the code word
3899     "s" is the sign bit
3900     "xxxx" is the four bit value field
```

The coder equation is described as follows:

```
3902     if (Xdiff( n ) < 0) then
3903         sign = 1
3904     else
3905         sign = 0
3906     endif
3907     value = (abs(Xdiff( n )) - 40) / 4
```

E.2.5.4 DPCM4 for 12–8–12 Coder

Xenco(n) has the following format:

```
3909     Xenco( n ) = "001 s xxxx"
```

where,

```
3911     "001" is the code word
3912     "s" is the sign bit
3913     "xxxx" is the four bit value field
```

The coder equation is described as follows:

```
3915     if (Xdiff( n ) < 0) then
3916         sign = 1
3917     else
3918         sign = 0
3919     endif
3920     value = (abs(Xdiff( n )) - 104) / 8
```

E.2.5.5 DPCM5 for 12–8–12 Coder

Xenco(n) has the following format:

```
3921     Xenco( n ) = "0001 s xxx"
```

where,

```
3922     "0001" is the code word  
3923     "s" is the sign bit  
3924     "xxx" is the three bit value field
```

The coder equation is described as follows:

```
3925     if (Xdiff( n ) < 0) then  
3926         sign = 1  
3927     else  
3928         sign = 0  
3929     endif  
3930     value = (abs(Xdiff( n )) - 232) / 16
```

E.2.5.6 PCM for 12–8–12 Coder

Xenco(n) has the following format:

```
3931     Xenco( n ) = "1 xxxxxxxx"
```

where,

```
3932     "1" is the code word  
3933     the sign bit is not used  
3934     "xxxxxxxx" is the seven bit value field
```

The coder equation is described as follows:

```
3935     value = Xorig( n ) / 32
```

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

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

Pixels without prediction are encoded using the following formula:

```
3944     Xenco( n ) = Xorig( n ) / 32
```

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

```
3946     if (Xenco( n ) == 0) then
3947         Xenco( n ) = 1
3948     endif
```

Pixels with prediction are encoded using the following formula:

```
3950     if (abs(Xdiff( n )) < 4) then
3951         use DPCM1
3952     else if (abs(Xdiff( n )) < 12) then
3953         use DPCM2
3954     else if (abs(Xdiff( n )) < 28) then
3955         use DPCM3
3956     else if (abs(Xdiff( n )) < 92) then
3957         use DPCM4
3958     else if (abs(Xdiff( n )) < 220) then
3959         use DPCM5
3960     else if (abs(Xdiff( n )) < 348) then
3961         use DPCM6
3962     else
3963         use PCM
3964     endif
```

E.2.6.1 DPCM1 for 12–7–12 Coder

Xenco(n) has the following format:

```
3966     Xenco( n ) = "0000 s xx"
```

where,

```
3968     "0000" is the code word
3969     "s" is the sign bit
3970     "xx" is the two bit value field
```

The coder equation is described as follows:

```
3972     if (Xdiff( n ) <= 0) then
3973         sign = 1
3974     else
3975         sign = 0
3976     endif
3977     value = abs(Xdiff( n ))
```

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

E.2.6.2 DPCM2 for 12-7-12 Coder

Xenco(n) has the following format:

```
Xenco( n ) = "0001 s xx"
```

where,

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

The coder equation is described as follows:

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

E.2.6.3 DPCM3 for 12-7-12 Coder

Xenco(n) has the following format:

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

where,

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

The coder equation is described as follows:

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

E.2.6.4 DPCM4 for 12-7-12 Coder

Xenco(n) has the following format:

```
Xenco( n ) = "010 s xxx"
```

where,

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

The coder equation is described as follows:

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

E.2.6.5 DPCM5 for 12-7-12 Coder

4018 **Xenco(n)** has the following format:

4019 **Xenco(n)** = "011 s xxx"

4020 where,

4021 "011" is the code word

4022 "s" is the **sign** bit

4023 "xxx" is the three bit **value** field

4024 The coder equation is described as follows:

```
4025       if (Xdiff( n ) < 0) then
4026           sign = 1
4027       else
4028           sign = 0
4029       endif
4030       value = (abs(Xdiff( n )) - 92) / 16
```

E.2.6.6 DPCM6 for 12-7-12 Coder

4031 **Xenco(n)** has the following format:

4032 **Xenco(n)** = "0011 s xx"

4033 where,

4034 "0011" is the code word

4035 "s" is the **sign** bit

4036 "xx" is the two bit **value** field

4037 The coder equation is described as follows:

```
4038       if (Xdiff( n ) < 0) then
4039           sign = 1
4040       else
4041           sign = 0
4042       endif
4043       value = (abs(Xdiff( n )) - 220) / 32
```

E.2.6.7 PCM for 12-7-12 Coder

4044 **Xenco(n)** has the following format:

4045 **Xenco(n)** = "1 xxxxxxxx"

4046 where,

4047 "1" is the code word

4048 the **sign** bit is not used

4049 "xxxxxxxx" is the six bit **value** field

4050 The coder equation is described as follows:

4051 **value** = **Xorig(n)** / 64

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

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

Pixels without prediction are encoded using the following formula:

```
4052     Xenco( n ) = Xorig( n ) / 64
```

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

```
4056     if (Xenco( n ) == 0) then
4057         Xenco( n ) = 1
4058     endif
```

Pixels with prediction are encoded using the following formula:

```
4060     if (abs(Xdiff( n )) < 2) then
4061         use DPCM1
4062     else if (abs(Xdiff( n )) < 10) then
4063         use DPCM3
4064     else if (abs(Xdiff( n )) < 42) then
4065         use DPCM4
4066     else if (abs(Xdiff( n )) < 74) then
4067         use DPCM5
4068     else if (abs(Xdiff( n )) < 202) then
4069         use DPCM6
4070     else if (abs(Xdiff( n )) < 330) then
4071         use DPCM7
4072     else
4073         use PCM
4074     endif
```

Note: DPCM2 is not used.

E.2.7.1 DPCM1 for 12–6–12 Coder

Xenco(n) has the following format:

```
4076     Xenco( n ) = "0000 s x"
```

where,

```
4079     "0000" is the code word
4080     "s" is the sign bit
4081     "x" is the one bit value field
```

The coder equation is described as follows:

```
4083     if (Xdiff( n ) <= 0) then
4084         sign = 1
4085     else
4086         sign = 0
4087     endif
4088     value = abs(Xdiff( n ))
```

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

E.2.7.2 DPCM3 for 12–6–12 Coder

Xenco(n) has the following format:

```
4090      Xenco( n ) = "0001 s x"
```

where,

```
4093      "0001" is the code word
4094      "s" is the sign bit
4095      "x" is the one bit value field
```

The coder equation is described as follows:

```
4096      if (Xdiff( n ) < 0) then
4097          sign = 1
4098      else
4099          sign = 0
4100      endif
4101      value = (abs(Xdiff( n )) - 2) / 4
```

E.2.7.3 DPCM4 for 12–6–12 Coder

Xenco(n) has the following format:

```
4103      Xenco( n ) = "010 s xx"
```

where,

```
4106      "010" is the code word
4107      "s" is the sign bit
4108      "xx" is the two bit value field
```

The coder equation is described as follows:

```
4109      if (Xdiff( n ) < 0) then
4110          sign = 1
4111      else
4112          sign = 0
4113      endif
4114      value = (abs(Xdiff( n )) - 10) / 8
```

E.2.7.4 DPCM5 for 12–6–12 Coder

Xenco(n) has the following format:

```
4116      Xenco( n ) = "0010 s x"
```

where,

```
4119      "0010" is the code word
4120      "s" is the sign bit
4121      "x" is the one bit value field
```

The coder equation is described as follows:

```
4123      if (Xdiff( n ) < 0) then
4124          sign = 1
4125      else
4126          sign = 0
4127      endif
4128      value = (abs(Xdiff( n )) - 42) / 16
```

E.2.7.5 DPCM6 for 12–6–12 Coder

Xenco(n) has the following format:

```
4129     Xenco( n ) = "011 s xx"
```

where,

```
4132     "011" is the code word
4133     "s" is the sign bit
4134     "xx" is the two bit value field
```

The coder equation is described as follows:

```
4136     if (Xdiff( n ) < 0) then
4137         sign = 1
4138     else
4139         sign = 0
4140     endif
4141     value = (abs(Xdiff( n )) - 74) / 32
```

E.2.7.6 DPCM7 for 12–6–12 Coder

Xenco(n) has the following format:

```
4142     Xenco( n ) = "0011 s x"
```

where,

```
4145     "0011" is the code word
4146     "s" is the sign bit
4147     "x" is the one bit value field
```

The coder equation is described as follows:

```
4149     if (Xdiff( n ) < 0) then
4150         sign = 1
4151     else
4152         sign = 0
4153     endif
4154     value = (abs(Xdiff( n )) - 202) / 64
```

E.2.7.7 PCM for 12–6–12 Coder

Xenco(n) has the following format:

```
4155     Xenco( n ) = "1 xxxxx"
```

where,

```
4158     "1" is the code word
4159     the sign bit is not used
4160     "xxxxx" is the five bit value field
```

The coder equation is described as follows:

```
4161     value = Xorig( n ) / 128
```

E.3 Decoders

4163 There are six different decoders available, one for each data compression scheme.

4164 For all decoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
4165 for predicted pixels.

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

4166 Pixels without prediction are decoded using the following formula:

```
4167     Xdeco( n ) = 4 * Xenco( n ) + 2
```

4168 Pixels with prediction are decoded using the following formula:

```
4169     if (Xenco( n ) & 0xc0 == 0x00) then
4170         use DPCM1
4171     else if (Xenco( n ) & 0xe0 == 0x40) then
4172         use DPCM2
4173     else if (Xenco( n ) & 0xe0 == 0x60) then
4174         use DPCM3
4175     else
4176         use PCM
4177     endif
```

E.3.1.1 DPCM1 for 10–8–10 Decoder

4178 **Xenco(n)** has the following format:

```
4179     Xenco( n ) = "00 s xxxxx"
```

4180 where,

```
4181     "00" is the code word
4182     "s" is the sign bit
4183     "xxxxx" is the five bit value field
```

4184 The decoder equation is described as follows:

```
4185     sign = Xenco( n ) & 0x20
4186     value = Xenco( n ) & 0x1f
4187     if (sign > 0) then
4188         Xdeco( n ) = Xpred( n ) - value
4189     else
4190         Xdeco( n ) = Xpred( n ) + value
4191     endif
```

E.3.1.2 DPCM2 for 10–8–10 Decoder

Xenco(n) has the following format:

```
4192     Xenco( n ) = "010 s xxxx"
```

where,

```
4195     "010" is the code word
4196     "s" is the sign bit
4197     "xxxx" is the four bit value field
```

The decoder equation is described as follows:

```
4199     sign = Xenco( n ) & 0x10
4200     value = 2 * (Xenco( n ) & 0xf) + 32
4201     if (sign > 0) then
4202         Xdeco( n ) = Xpred( n ) - value
4203     else
4204         Xdeco( n ) = Xpred( n ) + value
4205     endif
```

E.3.1.3 DPCM3 for 10–8–10 Decoder

Xenco(n) has the following format:

```
4206     Xenco( n ) = "011 s xxxx"
```

where,

```
4209     "011" is the code word
4210     "s" is the sign bit
4211     "xxxx" is the four bit value field
```

The decoder equation is described as follows:

```
4213     sign = Xenco( n ) & 0x10
4214     value = 4 * (Xenco( n ) & 0xf) + 64 + 1
4215     if (sign > 0) then
4216         Xdeco( n ) = Xpred( n ) - value
4217         if (Xdeco( n ) < 0) then
4218             Xdeco( n ) = 0
4219         endif
4220     else
4221         Xdeco( n ) = Xpred( n ) + value
4222         if (Xdeco( n ) > 1023) then
4223             Xdeco( n ) = 1023
4224         endif
4225     endif
```

E.3.1.4 PCM for 10–8–10 Decoder

4226 **Xenco(n)** has the following format:

4227 **Xenco(n)** = "1 xxxxxxxx"

4228 where,

4229 "1" is the code word

4230 the **sign** bit is not used

4231 "xxxxxxxx" is the seven bit **value** field

4232 The codec equation is described as follows:

```
4233    value = 8 * (Xenco( n ) & 0x7f)
4234    if (value > Xpred( n )) then
4235       Xdeco( n ) = value + 3
4236     endif
4237    else
4238       Xdeco( n ) = value + 4
4239     endif
```

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

4240 Pixels without prediction are decoded using the following formula:

```
4241     Xdeco( n ) = 8 * Xenco( n ) + 4
```

4242 Pixels with prediction are decoded using the following formula:

```
4243     if (Xenco( n ) & 0x70 == 0x00) then
4244         use DPCM1
4245     else if (Xenco( n ) & 0x78 == 0x10) then
4246         use DPCM2
4247     else if (Xenco( n ) & 0x78 == 0x18) then
4248         use DPCM3
4249     else if (Xenco( n ) & 0x60 == 0x20) then
4250         use DPCM4
4251     else
4252         use PCM
4253     endif
```

E.3.2.1 DPCM1 for 10–7–10 Decoder

4254 **Xenco(n)** has the following format:

```
4255     Xenco( n ) = "000 s xxx"
```

4256 where,

```
4257     "000" is the code word
4258     "s" is the sign bit
4259     "xxx" is the three bit value field
```

4260 The codec equation is described as follows:

```
4261     sign = Xenco( n ) & 0x8
4262     value = Xenco( n ) & 0x7
4263     if (sign > 0) then
4264         Xdeco( n ) = Xpred( n ) - value
4265     else
4266         Xdeco( n ) = Xpred( n ) + value
4267     endif
```

E.3.2.2 DPCM2 for 10–7–10 Decoder

4268 **Xenco(n)** has the following format:

```
4269     Xenco( n ) = "0010 s xx"
```

4270 where,

```
4271     "0010" is the code word
4272     "s" is the sign bit
4273     "xx" is the two bit value field
```

4274 The codec equation is described as follows:

```
4275     sign = Xenco( n ) & 0x4
4276     value = 2 * (Xenco( n ) & 0x3) + 8
4277     if (sign > 0) then
4278         Xdeco( n ) = Xpred( n ) - value
4279     else
4280         Xdeco( n ) = Xpred( n ) + value
4281     endif
```

E.3.2.3 DPCM3 for 10–7–10 Decoder

4282 **Xenco(n)** has the following format:

4283 **Xenco(n)** = "0011 s xx"

4284 where,

4285 "0011" is the code word

4286 "s" is the **sign** bit

4287 "xx" is the two bit **value** field

4288 The codec equation is described as follows:

```
4289    sign = Xenco( n ) & 0x4
4290    value = 4 * (Xenco( n ) & 0x3) + 16 + 1
4291    if (sign > 0) then
4292      Xdeco( n ) = Xpred( n ) - value
4293      if (Xdeco( n ) < 0) then
4294        Xdeco( n ) = 0
4295      endif
4296    else
4297      Xdeco( n ) = Xpred( n ) + value
4298      if (Xdeco( n ) > 1023) then
4299        Xdeco( n ) = 1023
4300      endif
4301  endif
```

E.3.2.4 DPCM4 for 10–7–10 Decoder

4302 **Xenco(n)** has the following format:

4303 **Xenco(n)** = "01 s xxxx"

4304 where,

4305 "01" is the code word

4306 "s" is the **sign** bit

4307 "xxxx" is the four bit **value** field

4308 The codec equation is described as follows:

```
4309    sign = Xenco( n ) & 0x10
4310    value = 8 * (Xenco( n ) & 0xf) + 32 + 3
4311    if (sign > 0) then
4312      Xdeco( n ) = Xpred( n ) - value
4313      if (Xdeco( n ) < 0) then
4314        Xdeco( n ) = 0
4315      endif
4316  else
4317      Xdeco( n ) = Xpred( n ) + value
4318      if (Xdeco( n ) > 1023) then
4319        Xdeco( n ) = 1023
4320      endif
4321  endif
```

E.3.2.5 PCM for 10–7–10 Decoder

4322 **Xenco(n)** has the following format:

4323 **Xenco(n)** = "1 xxxxxx"

4324 where,

4325 "1" is the code word

4326 the **sign** bit is not used

4327 "xxxxxx" is the six bit **value** field

4328 The codec equation is described as follows:

4329 **value** = 16 * (**Xenco(n)** & 0x3f)

4330 if (**value** > **Xpred(n)**) then

4331 **Xdeco(n)** = **value** + 7

4332 else

4333 **Xdeco(n)** = **value** + 8

4334 endif

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

4335 Pixels without prediction are decoded using the following formula:

$$4336 \quad \mathbf{Xdeco}(\mathbf{n}) = 16 * \mathbf{Xenco}(\mathbf{n}) + 8$$

4337 Pixels with prediction are decoded using the following formula:

```
4338     if (Xenco(n) & 0x3e == 0x00) then
4339         use DPCM1
4340     else if (Xenco(n) & 0x3e == 0x02) then
4341         use DPCM2
4342     else if (Xenco(n) & 0x3c == 0x04) then
4343         use DPCM3
4344     else if (Xenco(n) & 0x38 == 0x08) then
4345         use DPCM4
4346     else if (Xenco(n) & 0x30 == 0x10) then
4347         use DPCM5
4348     else
4349         use PCM
4350     endif
```

E.3.3.1 DPCM1 for 10–6–10 Decoder

4351 **Xenco(n)** has the following format:

$$4352 \quad \mathbf{Xenco}(\mathbf{n}) = "00000 \mathbf{s}"$$

4353 where,

4354 "00000" is the code word

4355 "s" is the **sign** bit

4356 the **value** field is not used

4357 The codec equation is described as follows:

$$4358 \quad \mathbf{Xdeco}(\mathbf{n}) = \mathbf{Xpred}(\mathbf{n})$$

E.3.3.2 DPCM2 for 10–6–10 Decoder

4359 **Xenco(n)** has the following format:

$$4360 \quad \mathbf{Xenco}(\mathbf{n}) = "00001 \mathbf{s}"$$

4361 where,

4362 "00001" is the code word

4363 "s" is the **sign** bit

4364 the **value** field is not used

4365 The codec equation is described as follows:

```
4366     sign = Xenco(n) & 0x1
4367     value = 1
4368     if (sign > 0) then
4369         Xdeco(n) = Xpred(n) - value
4370     else
4371         Xdeco(n) = Xpred(n) + value
4372     endif
```

E.3.3.3 DPCM3 for 10–6–10 Decoder

Xenco(n) has the following format:

```
4373     Xenco( n ) = "0001 s x"
```

where,

```
4376     "0001" is the code word
4377     "s" is the sign bit
4378     "x" is the one bit value field
```

The codec equation is described as follows:

```
4380     sign = Xenco( n ) & 0x2
4381     value = 4 * (Xenco( n ) & 0x1) + 3 + 1
4382     if (sign > 0) then
4383         Xdeco( n ) = Xpred( n ) - value
4384         if (Xdeco( n ) < 0) then
4385             Xdeco( n ) = 0
4386         endif
4387     else
4388         Xdeco( n ) = Xpred( n ) + value
4389         if (Xdeco( n ) > 1023) then
4390             Xdeco( n ) = 1023
4391         endif
4392     endif
```

E.3.3.4 DPCM4 for 10–6–10 Decoder

Xenco(n) has the following format:

```
4394     Xenco( n ) = "001 s xx"
```

where,

```
4396     "001" is the code word
4397     "s" is the sign bit
4398     "xx" is the two bit value field
```

The codec equation is described as follows:

```
4400     sign = Xenco( n ) & 0x4
4401     value = 8 * (Xenco( n ) & 0x3) + 11 + 3
4402     if (sign > 0) then
4403         Xdeco( n ) = Xpred( n ) - value
4404         if (Xdeco( n ) < 0) then
4405             Xdeco( n ) = 0
4406         endif
4407     else
4408         Xdeco( n ) = Xpred( n ) + value
4409         if (Xdeco( n ) > 1023) then
4410             Xdeco( n ) = 1023
4411         endif
4412     endif
```

E.3.3.5 DPCM5 for 10–6–10 Decoder

4413 **Xenco(n)** has the following format:

4414 **Xenco(n)** = "01 s xxx"

4415 where,

4416 "01" is the code word

4417 "s" is the **sign** bit

4418 "xxx" is the three bit **value** field

4419 The codec equation is described as follows:

```
4420    sign = Xenco( n ) & 0x8
4421    value = 16 * (Xenco( n ) & 0x7) + 43 + 7
4422    if (sign > 0) then
4423      Xdeco( n ) = Xpred( n ) - value
4424      if (Xdeco( n ) < 0) then
4425        Xdeco( n ) = 0
4426      endif
4427    else
4428      Xdeco( n ) = Xpred( n ) + value
4429      if (Xdeco( n ) > 1023) then
4430        Xdeco( n ) = 1023
4431      endif
4432  endif
```

E.3.3.6 PCM for 10–6–10 Decoder

4433 **Xenco(n)** has the following format:

4434 **Xenco(n)** = "1 xxxxx"

4435 where,

4436 "1" is the code word

4437 the **sign** bit is not used

4438 "xxxxx" is the five bit **value** field

4439 The codec equation is described as follows:

```
4440    value = 32 * (Xenco( n ) & 0x1f)
4441    if (value > Xpred( n )) then
4442      Xdeco( n ) = value + 15
4443    else
4444      Xdeco( n ) = value + 16
4445  endif
```

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

Pixels without prediction are decoded using the following formula:

```
4446      Xdeco( n ) = 4 * Xenco( n ) + 2
```

Pixels with prediction are decoded using the following formula:

```
4449      if (Xenco( n ) & 0x300 == 0x000) then
4450          use DPCM1
4451      else if (Xenco( n ) & 0x380 == 0x100) then
4452          use DPCM2
4453      else if (Xenco( n ) & 0x380 == 0x180) then
4454          use DPCM3
4455      else
4456          use PCM
4457      endif
```

E.3.4.1 DPCM1 for 12–10–12 Decoder

Xenco(n) has the following format:

```
4459      Xenco( n ) = "00 s xxxxxxxx"
```

where,

```
4461      "00" is the code word
4462      "s" is the sign bit
4463      "xxxxxxxx" is the seven bit value field
```

The decoder equation is described as follows:

```
4465      sign = Xenco( n ) & 0x80
4466      value = Xenco( n ) & 0x7f
4467      if (sign > 0) then
4468          Xdeco( n ) = Xpred( n ) - value
4469      else
4470          Xdeco( n ) = Xpred( n ) + value
4471      endif
```

E.3.4.2 DPCM2 for 12–10–12 Decoder

4472 **Xenco(n)** has the following format:

4473 **Xenco(n)** = "010 s xxxxxxx"

4474 where,

4475 "010" is the code word

4476 "s" is the **sign** bit

4477 "xxxxxx" is the six bit **value** field

4478 The decoder equation is described as follows:

```
4479   sign = Xenco( n ) & 0x40
4480   value = 2 * (Xenco( n ) & 0x3f) + 128
4481   if (sign > 0) then
4482     Xdeco( n ) = Xpred( n ) - value
4483   else
4484     Xdeco( n ) = Xpred( n ) + value
4485   endif
```

E.3.4.3 DPCM3 for 12–10–12 Decoder

4486 **Xenco(n)** has the following format:

4487 **Xenco(n)** = "011 s xxxxxxx"

4488 where,

4489 "011" is the code word

4490 "s" is the **sign** bit

4491 "xxxxxx" is the six bit **value** field

4492 The decoder equation is described as follows:

```
4493   sign = Xenco( n ) & 0x40
4494   value = 4 * (Xenco( n ) & 0x3f) + 256 + 1
4495   if (sign > 0) then
4496     Xdeco( n ) = Xpred( n ) - value
4497     if (Xdeco( n ) < 0) then
4498       Xdeco( n ) = 0
4499     endif
4500   else
4501     Xdeco( n ) = Xpred( n ) + value
4502     if (Xdeco( n ) > 4095) then
4503       Xdeco( n ) = 4095
4504     endif
4505   endif
```

E.3.4.4 PCM for 12–10–12 Decoder

4506 **Xenco(n)** has the following format:

4507 **Xenco(n)** = "1 xxxxxxxx"

4508 where,

4509 "1" is the code word

4510 the **sign** bit is not used

4511 "xxxxxxxx" is the nine bit **value** field

4512 The codec equation is described as follows:

4513 **value** = 8 * (**Xenco(n)** & 0x1ff)

4514 if (**value** > **Xpred(n)**) then

4515 **Xdeco(n)** = **value** + 3

4516 endif

4517 else

4518 **Xdeco(n)** = **value** + 4

4519 endif

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

4520 Pixels without prediction are decoded using the following formula:

```
4521     Xdeco( n ) = 16 * Xenco( n ) + 8
```

4522 Pixels with prediction are decoded using the following formula:

```
4523     if (Xenco( n ) & 0xf0 == 0x00) then
4524         use DPCM1
4525     else if (Xenco( n ) & 0xe0 == 0x60) then
4526         use DPCM2
4527     else if (Xenco( n ) & 0xe0 == 0x40) then
4528         use DPCM3
4529     else if (Xenco( n ) & 0xe0 == 0x20) then
4530         use DPCM4
4531     else if (Xenco( n ) & 0xf0 == 0x10) then
4532         use DPCM5
4533     else
4534         use PCM
4535     endif
```

E.3.5.1 DPCM1 for 12–8–12 Decoder

4536 **Xenco(n)** has the following format:

```
4537     Xenco( n ) = "0000 s xxx"
```

4538 where,

```
4539     "0000" is the code word
4540     "s" is the sign bit
4541     "xxx" is the three bit value field
```

4542 The codec equation is described as follows:

```
4543     sign = Xenco( n ) & 0x8
4544     value = Xenco( n ) & 0x7
4545     if (sign > 0) then
4546         Xdeco( n ) = Xpred( n ) - value
4547     else
4548         Xdeco( n ) = Xpred( n ) + value
4549     endif
```

E.3.5.2 DPCM2 for 12–8–12 Decoder

4550 **Xenco(n)** has the following format:

```
4551     Xenco( n ) = "011 s xxxx"
```

4552 where,

```
4553     "011" is the code word
4554     "s" is the sign bit
4555     "xxxx" is the four bit value field
```

4556 The codec equation is described as follows:

```
4557     sign = Xenco( n ) & 0x10
4558     value = 2 * (Xenco( n ) & 0xf) + 8
4559     if (sign > 0) then
4560         Xdeco( n ) = Xpred( n ) - value
4561     else
4562         Xdeco( n ) = Xpred( n ) + value
4563     endif
```

E.3.5.3 DPCM3 for 12–8–12 Decoder

Xenco(n) has the following format:

```
4564     Xenco( n ) = "010 s xxxx"
```

where,

```
4567     "010" is the code word
4568     "s" is the sign bit
4569     "xxxx" is the four bit value field
```

The codec equation is described as follows:

```
4571     sign = Xenco( n ) & 0x10
4572     value = 4 * (Xenco( n ) & 0xf) + 40 + 1
4573     if (sign > 0) then
4574         Xdeco( n ) = Xpred( n ) - value
4575         if (Xdeco( n ) < 0) then
4576             Xdeco( n ) = 0
4577         endif
4578     else
4579         Xdeco( n ) = Xpred( n ) + value
4580         if (Xdeco( n ) > 4095) then
4581             Xdeco( n ) = 4095
4582         endif
4583     endif
```

E.3.5.4 DPCM4 for 12–8–12 Decoder

Xenco(n) has the following format:

```
4584     Xenco( n ) = "001 s xxxx"
```

where,

```
4587     "001" is the code word
4588     "s" is the sign bit
4589     "xxxx" is the four bit value field
```

The codec equation is described as follows:

```
4591     sign = Xenco( n ) & 0x10
4592     value = 8 * (Xenco( n ) & 0xf) + 104 + 3
4593     if (sign > 0) then
4594         Xdeco( n ) = Xpred( n ) - value
4595         if (Xdeco( n ) < 0) then
4596             Xdeco( n ) = 0
4597         endif
4598     else
4599         Xdeco( n ) = Xpred( n ) + value
4600         if (Xdeco( n ) > 4095)
4601             Xdeco( n ) = 4095
4602         endif
4603     endif
```

E.3.5.5 DPCM5 for 12–8–12 Decoder

4604 **Xenco(n)** has the following format:

4605 **Xenco(n)** = "0001 s xxx"

4606 where,

4607 "0001" is the code word
 4608 "s" is the **sign** bit
 4609 "xxx" is the three bit **value** field

4610 The codec equation is described as follows:

```
4611      sign = Xenco( n ) & 0x8
4612      value = 16 * (Xenco( n ) & 0x7) + 232 + 7
4613      if (sign > 0) then
4614        Xdeco( n ) = Xpred( n ) - value
4615        if (Xdeco( n ) < 0) then
4616          Xdeco( n ) = 0
4617        endif
4618      else
4619        Xdeco( n ) = Xpred( n ) + value
4620        if (Xdeco( n ) > 4095) then
4621          Xdeco( n ) = 4095
4622        endif
4623      endif
```

E.3.5.6 PCM for 12–8–12 Decoder

4624 **Xenco(n)** has the following format:

4625 **Xenco(n)** = "1 xxxxxxx"

4626 where,

4627 "1" is the code word
 4628 the **sign** bit is not used
 4629 "xxxxxxxx" is the seven bit **value** field

4630 The codec equation is described as follows:

```
4631      value = 32 * (Xenco( n ) & 0x7f)
4632      if (value > Xpred( n )) then
4633        Xdeco( n ) = value + 15
4634      else
4635        Xdeco( n ) = value + 16
4636      endif
```

E.3.6 Decoder for 12-7-12 Data Compression

4637 Pixels without prediction are decoded using the following formula:

```
4638     Xdeco( n ) = 32 * Xenco( n ) + 16
```

4639 Pixels with prediction are decoded using the following formula:

```
4640     if (Xenco( n ) & 0x78 == 0x00) then
4641         use DPCM1
4642     else if (Xenco( n ) & 0x78 == 0x08) then
4643         use DPCM2
4644     else if (Xenco( n ) & 0x78 == 0x10) then
4645         use DPCM3
4646     else if (Xenco( n ) & 0x70 == 0x20) then
4647         use DPCM4
4648     else if (Xenco( n ) & 0x70 == 0x30) then
4649         use DPCM5
4650     else if (Xenco( n ) & 0x78 == 0x18) then
4651         use DPCM6
4652     else
4653         use PCM
4654     endif
```

E.3.6.1 DPCM1 for 12-7-12 Decoder

4655 **Xenco(n)** has the following format:

```
4656     Xenco( n ) = "0000 s xx"
```

4657 where,

```
4658     "0000" is the code word
4659     "s" is the sign bit
4660     "xx" is the two bit value field
```

4661 The codec equation is described as follows:

```
4662     sign = Xenco( n ) & 0x4
4663     value = Xenco( n ) & 0x3
4664     if (sign > 0) then
4665         Xdeco( n ) = Xpred( n ) - value
4666     else
4667         Xdeco( n ) = Xpred( n ) + value
4668     endif
```

E.3.6.2 DPCM2 for 12–7–12 Decoder

4669 **Xenco(n)** has the following format:

4670 **Xenco(n)** = "0001 s xx"

4671 where,

4672 "0001" is the code word

4673 "s" is the **sign** bit

4674 "xx" is the two bit **value** field

4675 The codec equation is described as follows:

```
4676       sign = Xenco( n ) & 0x4
4677       value = 2 * (Xenco( n ) & 0x3) + 4
4678       if (sign > 0) then
4679           Xdeco( n ) = Xpred( n ) - value
4680       else
4681           Xdeco( n ) = Xpred( n ) + value
4682       endif
```

E.3.6.3 DPCM3 for 12–7–12 Decoder

4683 **Xenco(n)** has the following format:

4684 **Xenco(n)** = "0010 s xx"

4685 where,

4686 "0010" is the code word

4687 "s" is the **sign** bit

4688 "xx" is the two bit **value** field

4689 The codec equation is described as follows:

```
4690       sign = Xenco( n ) & 0x4
4691       value = 4 * (Xenco( n ) & 0x3) + 12 + 1
4692       if (sign > 0) then
4693           Xdeco( n ) = Xpred( n ) - value
4694           if (Xdeco( n ) < 0) then
4695               Xdeco( n ) = 0
4696           endif
4697       else
4698           Xdeco( n ) = Xpred( n ) + value
4699           if (Xdeco( n ) > 4095) then
4700               Xdeco( n ) = 4095
4701           endif
4702       endif
```

E.3.6.4 DPCM4 for 12–7–12 Decoder

Xenco(n) has the following format:

```
4703     Xenco( n ) = "010 s xxx"
```

where,

```
4706     "010" is the code word
4707     "s" is the sign bit
4708     "xxx" is the three bit value field
```

The codec equation is described as follows:

```
4710     sign = Xenco( n ) & 0x8
4711     value = 8 * (Xenco( n ) & 0x7) + 28 + 3
4712     if (sign > 0) then
4713         Xdeco( n ) = Xpred( n ) - value
4714         if (Xdeco( n ) < 0) then
4715             Xdeco( n ) = 0
4716         endif
4717     else
4718         Xdeco( n ) = Xpred( n ) + value
4719         if (Xdeco( n ) > 4095) then
4720             Xdeco( n ) = 4095
4721         endif
4722     endif
```

E.3.6.5 DPCM5 for 12–7–12 Decoder

Xenco(n) has the following format:

```
4724     Xenco( n ) = "011 s xxx"
```

where,

```
4726     "011" is the code word
4727     "s" is the sign bit
4728     "xxx" is the three bit value field
```

The codec equation is described as follows:

```
4730     sign = Xenco( n ) & 0x8
4731     value = 16 * (Xenco( n ) & 0x7) + 92 + 7
4732     if (sign > 0) then
4733         Xdeco( n ) = Xpred( n ) - value
4734         if (Xdeco( n ) < 0) then
4735             Xdeco( n ) = 0
4736         endif
4737     else
4738         Xdeco( n ) = Xpred( n ) + value
4739         if (Xdeco( n ) > 4095) then
4740             Xdeco( n ) = 4095
4741         endif
4742     endif
```

E.3.6.6 DPCM6 for 12-7-12 Decoder

Xenco(n) has the following format:

Xenco(n) = "0011 s xx"

where,

"0011" is the code word
 "s" is the **sign** bit
 "xx" is the two bit **value** field

The codec equation is described as follows:

```
sign = Xenco( n ) & 0x4
value = 32 * (Xenco( n ) & 0x3) + 220 + 15
if (sign > 0) then
  Xdeco( n ) = Xpred( n ) - value
  if (Xdeco( n ) < 0) then
    Xdeco( n ) = 0
  endif
else
  Xdeco( n ) = Xpred( n ) + value
  if (Xdeco( n ) > 4095) then
    Xdeco( n ) = 4095
  endif
endif
```

E.3.6.7 PCM for 12-7-12 Decoder

Xenco(n) has the following format:

Xenco(n) = "1 xxxxxx"

where,

"1" is the code word
 the **sign** bit is not used
 "xxxxxx" is the six bit **value** field

The codec equation is described as follows:

```
value = 64 * (Xenco( n ) & 0x3f)
if (value > Xpred( n )) then
  Xdeco( n ) = value + 31
else
  Xdeco( n ) = value + 32
endif
```

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

Pixels without prediction are decoded using the following formula:

```
4776 Xdeco( n ) = 64 * Xenco( n ) + 32
```

Pixels with prediction are decoded using the following formula:

```
4779 if (Xenco( n ) & 0x3c == 0x00) then
4780     use DPCM1
4781 else if (Xenco( n ) & 0x3c == 0x04) then
4782     use DPCM3
4783 else if (Xenco( n ) & 0x38 == 0x10) then
4784     use DPCM4
4785 else if (Xenco( n ) & 0x3c == 0x08) then
4786     use DPCM5
4787 else if (Xenco( n ) & 0x38 == 0x18) then
4788     use DPCM6
4789 else if (Xenco( n ) & 0x3c == 0x0c) then
4790     use DPCM7
4791 else
4792     use PCM
4793 endif
```

Note: DPCM2 is not used.

E.3.7.1 DPCM1 for 12–6–12 Decoder

Xenco(n) has the following format:

```
4795 Xenco( n ) = "0000 s x"
```

where,

```
4798 "0000" is the code word
4799 "s" is the sign bit
4800 "x" is the one bit value field
```

The codec equation is described as follows:

```
4802 sign = Xenco( n ) & 0x2
4803 value = Xenco( n ) & 0x1
4804 if (sign > 0) then
4805     Xdeco( n ) = Xpred( n ) - value
4806 else
4807     Xdeco( n ) = Xpred( n ) + value
4808 endif
```

E.3.7.2 DPCM3 for 12–6–12 Decoder

4809 **Xenco(n)** has the following format:

4810 **Xenco(n)** = "0001 s x"

4811 where,

4812 "0001" is the code word

4813 "s" is the **sign** bit

4814 "x" is the one bit **value** field

4815 The codec equation is described as follows:

```
4816   sign = Xenco( n ) & 0x2
4817   value = 4 * (Xenco( n ) & 0x1) + 2 + 1
4818   if (sign > 0) then
4819     Xdeco( n ) = Xpred( n ) - value
4820     if (Xdeco( n ) < 0) then
4821       Xdeco( n ) = 0
4822     endif
4823   else
4824     Xdeco( n ) = Xpred( n ) + value
4825     if (Xdeco( n ) > 4095) then
4826       Xdeco( n ) = 4095
4827     endif
4828   endif
```

E.3.7.3 DPCM4 for 12–6–12 Decoder

4829 **Xenco(n)** has the following format:

4830 **Xenco(n)** = "010 s xx"

4831 where,

4832 "010" is the code word

4833 "s" is the **sign** bit

4834 "xx" is the two bit **value** field

4835 The codec equation is described as follows:

```
4836   sign = Xenco( n ) & 0x4
4837   value = 8 * (Xenco( n ) & 0x3) + 10 + 3
4838   if (sign > 0) then
4839     Xdeco( n ) = Xpred( n ) - value
4840     if (Xdeco( n ) < 0) then
4841       Xdeco( n ) = 0
4842     endif
4843   else
4844     Xdeco( n ) = Xpred( n ) + value
4845     if (Xdeco( n ) > 4095) then
4846       Xdeco( n ) = 4095
4847     endif
4848   endif
```

E.3.7.4 DPCM5 for 12–6–12 Decoder

4849 **Xenco(n)** has the following format:

4850 **Xenco(n)** = "0010 s x"

4851 where,

4852 "0010" is the code word

4853 "s" is the **sign** bit

4854 "x" is the one bit **value** field

4855 The codec equation is described as follows:

```
4856   sign = Xenco( n ) & 0x2
4857   value = 16 * (Xenco( n ) & 0x1) + 42 + 7
4858   if (sign > 0) then
4859     Xdeco( n ) = Xpred( n ) - value
4860     if (Xdeco( n ) < 0) then
4861       Xdeco( n ) = 0
4862     endif
4863   else
4864     Xdeco( n ) = Xpred( n ) + value
4865     if (Xdeco( n ) > 4095) then
4866       Xdeco( n ) = 4095
4867     endif
4868   endif
```

E.3.7.5 DPCM6 for 12–6–12 Decoder

4869 **Xenco(n)** has the following format:

4870 **Xenco(n)** = "011 s xx"

4871 where,

4872 "011" is the code word

4873 "s" is the **sign** bit

4874 "xx" is the two bit **value** field

4875 The codec equation is described as follows:

```
4876   sign = Xenco( n ) & 0x4
4877   value = 32 * (Xenco( n ) & 0x3) + 74 + 15
4878   if (sign > 0) then
4879     Xdeco( n ) = Xpred( n ) - value
4880     if (Xdeco( n ) < 0) then
4881       Xdeco( n ) = 0
4882     endif
4883   else
4884     Xdeco( n ) = Xpred( n ) + value
4885     if (Xdeco( n ) > 4095) then
4886       Xdeco( n ) = 4095
4887     endif
4888   endif
```

E.3.7.6 DPCM7 for 12–6–12 Decoder

4889 **Xenco(n)** has the following format:

4890 **Xenco(n)** = "0011 s x"

4891 where,

4892 "0011" is the code word

4893 "s" is the **sign** bit

4894 "x" is the one bit **value** field

4895 The codec equation is described as follows:

```
4896   sign = Xenco( n ) & 0x2
4897   value = 64 * (Xenco( n ) & 0x1) + 202 + 31
4898   if (sign > 0) then
4899     Xdeco( n ) = Xpred( n ) - value
4900     if (Xdeco( n ) < 0) then
4901       Xdeco( n ) = 0
4902     endif
4903   else
4904     Xdeco( n ) = Xpred( n ) + value
4905     if (Xdeco( n ) > 4095) then
4906       Xdeco( n ) = 4095
4907     endif
4908   endif
```

E.3.7.7 PCM for 12–6–12 Decoder

4909 **Xenco(n)** has the following format:

4910 **Xenco(n)** = "1 xxxxx"

4911 where,

4912 "1" is the code word

4913 the **sign** bit is not used

4914 "xxxxx" is the five bit **value** field

4915 The codec equation is described as follows:

```
4916   value = 128 * (Xenco( n ) & 0x1f)
4917   if (value > Xpred( n )) then
4918     Xdeco( n ) = value + 63
4919   else
4920     Xdeco( n ) = value + 64
4921   Endif
```

Annex F JPEG Interleaving (informative)

This annex illustrates how the standard features of the CSI-2 protocol should be used to interleave (multiplex) JPEG image data with other types of image data, e.g. RGB565 or YUV422, without requiring a custom JPEG format such as JPEG8.

The Virtual Channel Identifier and Data Type value in the CSI-2 Packet Header provide simple methods of interleaving multiple data streams or image data types at the packet level. Interleaving at the packet level minimizes the amount of buffering required in the system.

The Data Type value in the CSI-2 Packet Header should be used to multiplex different image data types at the CSI-2 transmitter and de-multiplex the data types at the CSI-2 receiver.

The Virtual Channel Identifier in the CSI-2 Packet Header should be used to multiplex different data streams (channels) at the CSI-2 transmitter and de-multiplex the streams at the CSI-2 receiver.

The main difference between the two interleaving methods is that images with different Data Type values within the same Virtual Channel use the same frame and line synchronization information, whereas multiple Virtual Channels (data streams) each have their own independent frame and line synchronization information and thus potentially each channel may have different frame rates.

Since the predefined Data Type values represent only YUV, RGB and RAW data types, one of the User Defined Data Type values should be used to represent JPEG image data.

Figure 224 illustrates interleaving JPEG image data with YUV422 image data using Data Type values.

Figure 225 illustrates interleaving JPEG image data with YUV422 image data using both Data Type values and Virtual Channel Identifiers.

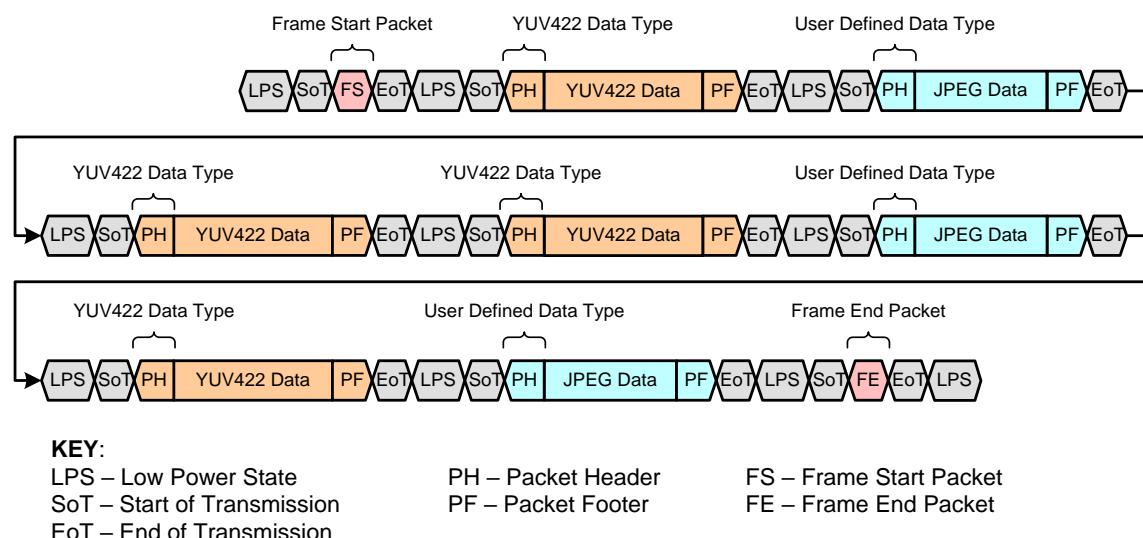


Figure 224 Data Type Interleaving: Concurrent JPEG and YUV Image Data

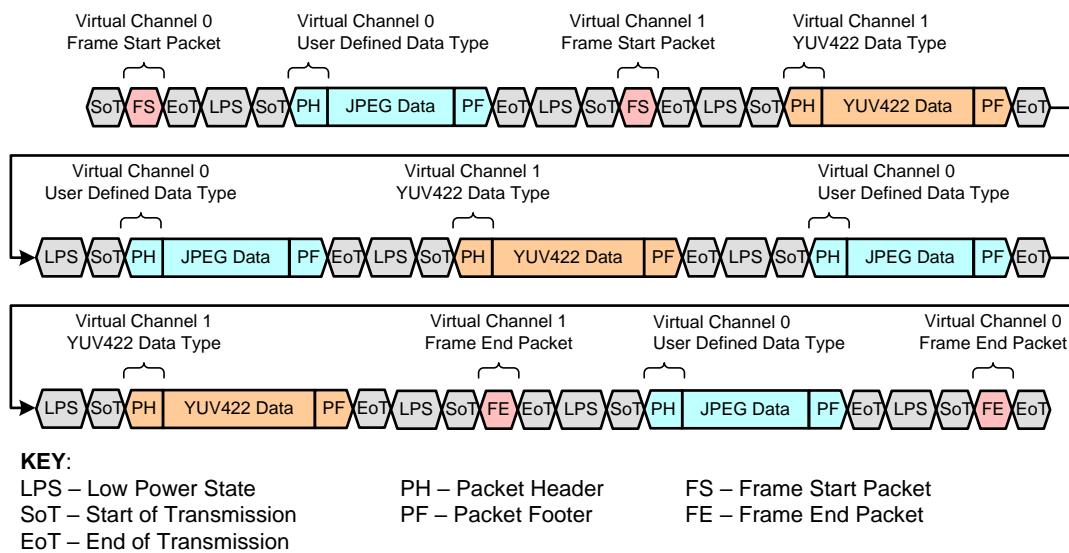


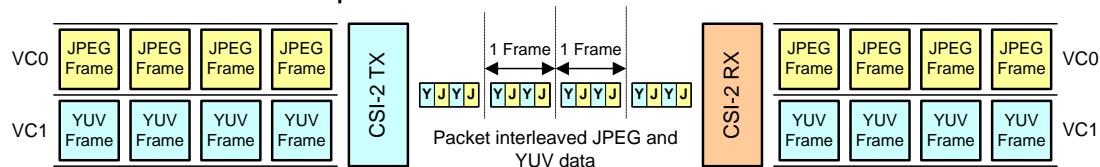
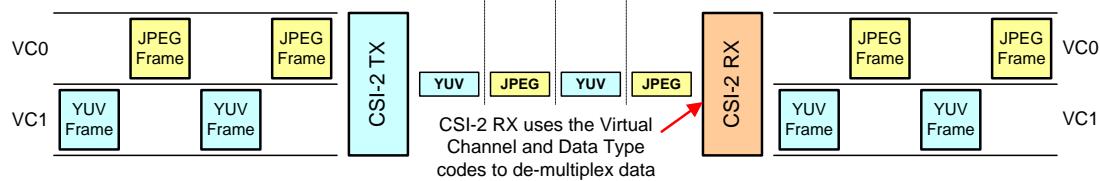
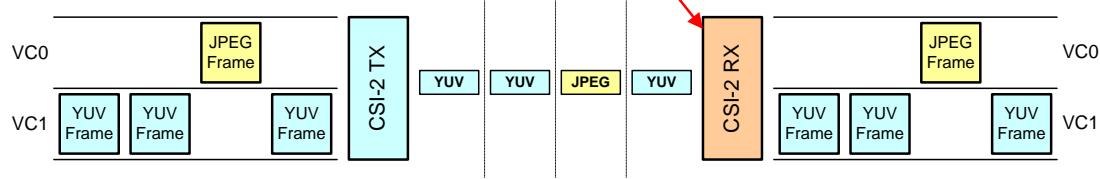
Figure 225 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data

Both **Figure 224** and **Figure 225** can be similarly extended to the interleaving of JPEG image data with any other type of image data, e.g. RGB565.

Figure 226 illustrates the use of Virtual Channels to support three different JPEG interleaving usage cases:

- Concurrent JPEG and YUV422 image data.
- Alternating JPEG and YUV422 output - one frame JPEG, then one frame YUV
- Streaming YUV22 with occasional JPEG for still capture

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

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

4950

Figure 226 Example JPEG and YUV Interleaving Use Cases

This page intentionally left blank.

Annex G Scrambler Seeds for Lanes 9 and Above

(See also: *Section 9.12*).

For Links of 9 to 32 Lanes, the Scrambler PRBS registers of Lanes 9 through 32 should be initialized with the initial seed values as listed in *Table 69*.

For Links of more than 32 Lanes, the Scrambler PRBS registers of Lanes 33 and higher shall use the same initial seed value that is used for the Lane number modulo 32. (See *Section 9.12* and *Table 69*.)

Examples

- Lane 33 shall use the same initial seed value as Lane 1
- Lane 34 shall use the same initial seed value as Lane 2
- Lane 64 shall use the same initial seed value as Lane 32
- Lane 65 shall use the same initial seed value as Lane 1

Table 69 Initial Seed Values for Lanes 9 through 32

Lane	Initial Seed Value
9	0x1818
10	0x1998
11	0x1a59
12	0x1bd8
13	0x1c38
14	0x1db8
15	0x1e78
16	0x1ff8
17	0x0001
18	0x0180
19	0x0240
20	0x03c0
21	0x0420
22	0x05a0
23	0x0660
24	0x07e0
25	0x0810
26	0x0990
27	0x0a51
28	0x0bd0
29	0x0c30
30	0x0db0
31	0x0e70
32	0x0ff0

Note that the binary representation of each initial seed value is symmetrical with respect to the forwards and backwards directions, with the exceptions of Lanes 11, 17, and 27. The initial seed values can be created easily using a Lane index value (i.e., Lane number minus one).

This page intentionally left blank.

4965

Annex H Guidance on CSI-2 Over C-PHY ALP and PPI

H.1 CSI-2 with C-PHY ALP Mode

C-PHY Alternate Low Power (ALP) Mode is an alternative to the legacy LP mode of C-PHY. ALP Mode uses solely High-Speed signaling with a special state where the signals can cease toggling and collapse to zero. The legacy LP Mode signaling and escape sequences have equivalent ALP Mode functions so that the high-voltage low power signaling can be replaced by ALP Mode signaling if that is beneficial in specific systems. ALP Mode replaces the legacy LP Mode line levels by the transmission of unique code words that are used only for Lane signaling events. These unique codes are never produced by the 3-Phase mapping function, so there is never ambiguity in the interpretation of these codes at the receiver.

Reasons to replace the legacy LP mode with equivalent ALP Mode functions are to begin a transitionary path to the future so that legacy LP mode might someday be eliminated in some devices. Another reason to choose ALP Mode over Legacy LP mode is to support systems that have long interconnect between the Primary and Secondary devices.

H.1.1 Concepts of ALP Mode and Legacy LP Mode

In ALP mode, the conventional LP receivers are not used to detect signaling states. Instead, all communication is performed using High-Speed signaling levels. The system level functions performed by ALP signaling are quite similar to the functional behavior of legacy LP mode. The intent of this is to cause the least amount of disruption to systems that support both ALP Mode and legacy LP mode. **Figure 227** shows a comparison of a High-Speed data burst with LP Mode versus ALP Mode. The purpose of this diagram is to show that each of the intervals in the High-Speed data burst with LP mode correspond to similar intervals in the High-Speed data burst with ALP mode.

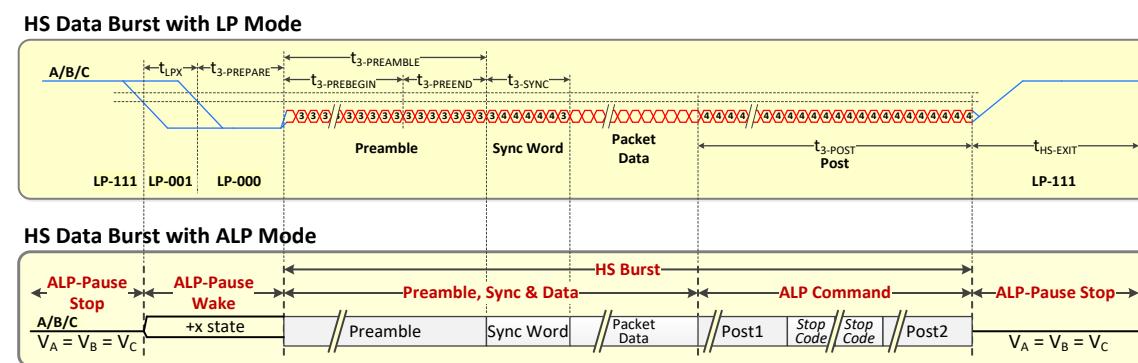


Figure 227 Comparing Data Burst Timing of Legacy LP Mode versus ALP Mode

ALP Mode supports the transmission of High-Speed data bursts as well as the transmission of control sequences that are traditionally transmitted using legacy LP mode Escape Mode sequences. The format of all ALP mode bursts is like the timing diagram in **Figure 228**.

The burst begins and ends in an ALP-Pause state. There are two types of ALP-Pause: ALP-Pause Stop and ALP-Pause ULPS. ALP-Pause Stop is analogous to the legacy LP mode Stop state; ALP-Pause ULPS is analogous to the legacy LP mode ULPS state. The only difference between these two types of ALP-Pause states is the time allowed to wake up from each, which is the duration of the ALP-Pause Wake interval. The nominal time allowed to wake from ALP-Pause Stop is 100 ns, which is about the same time as the duration of the LP-001 and LP-000 states at the beginning of a HS Data Burst using legacy LP mode. The nominal time to wake from the ALP-Pause ULPS state is 1 msec, which is approximately the time allowed in legacy LP mode for t_{WAKEUP} . (The time that a transmitter drives a Mark-1 state prior to a Stop state to initiate an exit from ULPS.) The longer wake-up time from ALP-Pause ULPS compared to ALP-Pause Stop allows a lower power consumption while in the ALP-Pause ULPS state.

4998 The ALP-Pause Stop and ALP-Pause ULPS line states are defined by the following relationships of the Line
 4999 levels: $V_A = V_B = V_C$, and $V_{OD_AB} = V_{OD_BC} = V_{OD_CA} = 0$. Examples of the ALP-Pause and the ALP-Pause
 5000 Wake states are illustrated at the beginning and end of the waveform in **Figure 228**. The ALP-Pause Wake
 5001 state, which is very long compared to a High-Speed Unit Interval, is detected by the low-power wake-up
 5002 receiver. This causes the system to leave one of the ALP-Pause states and to begin receiving a High-Speed
 5003 signal.

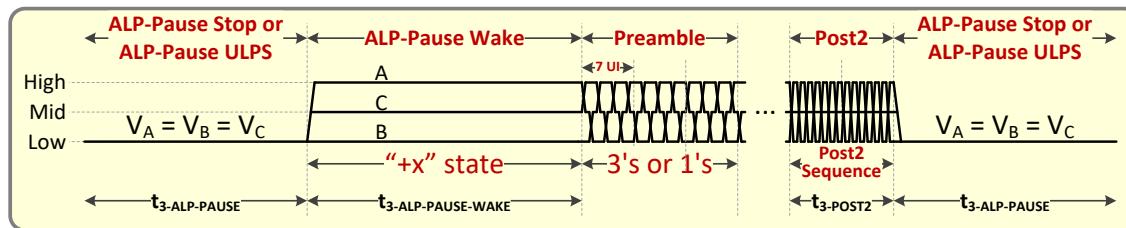


Figure 228 ALP Mode General Burst Format

5004 To minimize power consumption while Lane activity has ceased during one of the ALP-Pause states, a special
 5005 low-speed and low-power differential receiver circuit is present, in addition to the three High-Speed
 5006 differential receivers for A-B, B-C and C-A. This special low-speed and low-power differential receiver has
 5007 a nominal +80 mV offset input threshold voltage that detects the difference in differential levels between the
 5008 ALP-Pause state ($V_{OD} = 0$) and ALP-Pause Wake state ($V_{OD} = |V_{OD}| \text{ Strong}$). This allows the line signals to
 5009 collapse to zero with the $100\Omega Z_{ID}$ termination still connected, and still have a well-defined method to detect
 5010 the difference between the ALP-Pause and ALP-Pause Wake line conditions. Collapsing to zero with the
 5011 terminations still connected makes it possible for implementations to have very low power consumption
 5012 during the ALP-Pause states. The ALP-Pause Wake pulse is very long compared to a High-Speed Unit
 5013 Interval so that the wake receiver can be slow and consume very little power compared to the High-Speed
 5014 differential receivers.
 5015

5016
 5017
 5018
 5019
 5020 An example of the differential receiver circuit to support ALP mode is shown in **Figure 229**. Two different offset receivers are shown for wake from stop versus wake from ULPS, because the power consumption in the ALP-Pause ULPS state is expected to be lower than in ALP-Pause Stop state. The ALP-Pause Wake pulse from the ULPS state can be longer than waking from ALP-Pause Stop, so the ALP ULPS receiver can be slower and consume less power compared to the ALP Stop receiver.

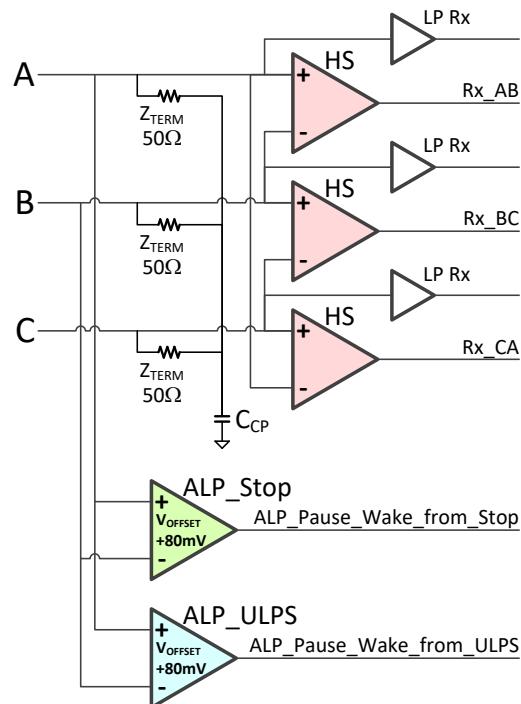


Figure 229 High-Speed and ALP-Pause Wake Receiver Example

5022 The C-PHY specification defines thirteen unique 7-symbol ALP Code Words that are the functional
 5023 equivalent of the LP pulse sequences of legacy LP mode. In some cases, a single 7-symbol ALP Code Word
 5024 can replace the transmission of a long sequence of legacy LP mode pulses, such as for the transmission of
 5025 Escape Mode triggers or low-power data transmission. The CSI-2 specification needs only three of these LP
 5026 mode pulse sequences to emulate the functionality of legacy LP mode: Stop Code, ULPS Code, and Post. A
 5027 fourth code, the TAC Code, is used for Fast Bus Turnaround.

5028 Exit from and entry into the ALP-Pause state, which is the functional equivalent of the legacy LP mode Stop
 5029 state, requires a special ALP Mode sequence consisting of one or more Stop Codes or ULPS codes followed
 5030 by a string of Post codes followed by setting the voltage of all three Lines of a Lane to the same value.

5031 As illustrated in **Figure 227**, the burst starting sequence of the legacy LP mode consisting of: LP-111, LP-
 5032 001, and LP-000 followed by preamble, has a functional equivalent sequence in ALP Mode consisting of:
 5033 ALP-Pause Stop followed by ALP Pause Wake followed by preamble. Similarly, the burst ending sequence
 5034 of legacy LP mode consisting of Post sequence followed by LP-111, has a functional equivalent sequence in
 5035 ALP Mode consisting of: the Post1 field by two or more Stop Codes followed by the Post2 field followed by
 5036 ALP-Pause Stop.

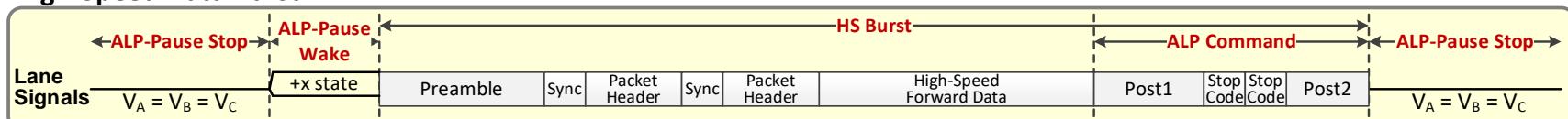
H.1.2 Burst Examples Using ALP Mode

5037 *Figure 230* shows examples of the three types of High-Speed bursts that can be sent in ALP mode. Many
5038 combinations of ALP code sequences are possible, but *Figure 230* shows three sequences that adequately
5039 perform the functions necessary to support CSI-2 that are currently performed using legacy LP mode. The
5040 ALP state machine from the C-PHY Specification has been highlighted in *Figure 231*, *Figure 232*, and
5041 *Figure 233* to show how transmission of these three sequences should occur.

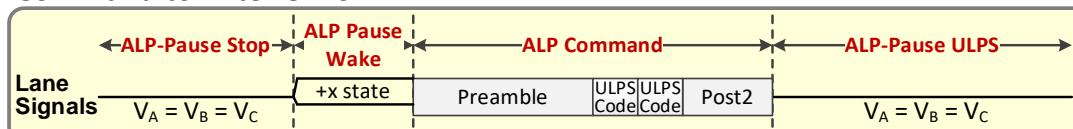
5042 For interop sake, only these three types of sequences are required to support CSI-2. Note that all bursts begin
5043 in the same manner with the assertion of ALP-Pause Wake followed by a Preamble. The words that follow
5044 the Preamble determine the type of burst that is being transmitted. All bursts end in the same manner with
5045 multiple Stop Codes followed by the Post2 field, or multiple ULPs Codes followed by the Post2 field. The
5046 Post 1 and Post2 fields are the same as Post (4444444), described in the C-PHY specification for burst
5047 transmission using legacy LP mode. The only difference is that the Post1 and Post2 fields are transmitted as
5048 a result of signaling over the PPI from the CSI-2 Tx to the C-PHY Tx.

5049 The last ALP code sent in the burst determines whether the system enters the ALP-Pause Stop or the ALP-
5050 Pause ULPs state.

High-Speed Data Burst



Command to Enter ULPS



Command to Exit from ULPS

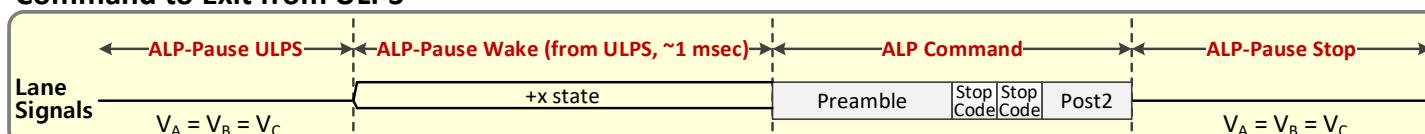
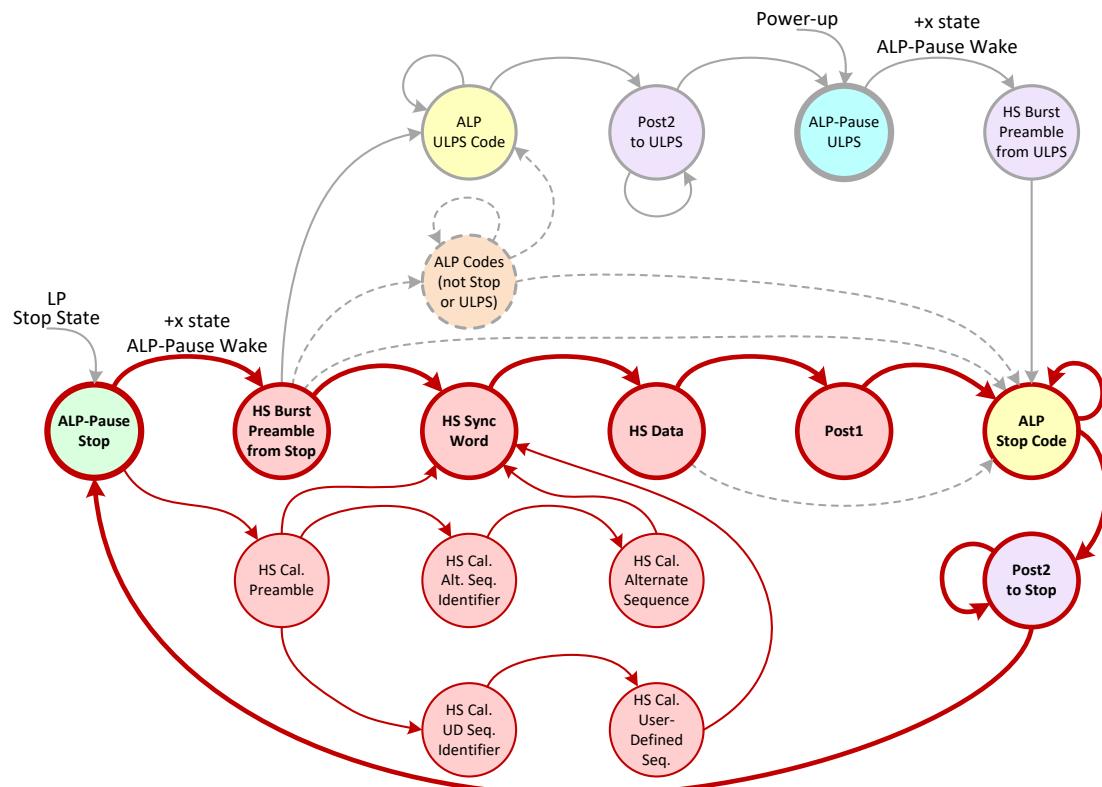


Figure 230 Examples of Bursts to Send High-Speed Data and ALP Commands

5052
5053
5054
5055 **Figure 231** shows the ALP state machine transitions (highlighted in red) necessary to transmit a High-Speed data burst in ALP mode. States and state transitions that are not used by CSI-2 for any type of burst are shown using dashed lines. The red highlighted states and transitions indicate the path required to transmit and receive the High-Speed Data Burst example in **Figure 230**.

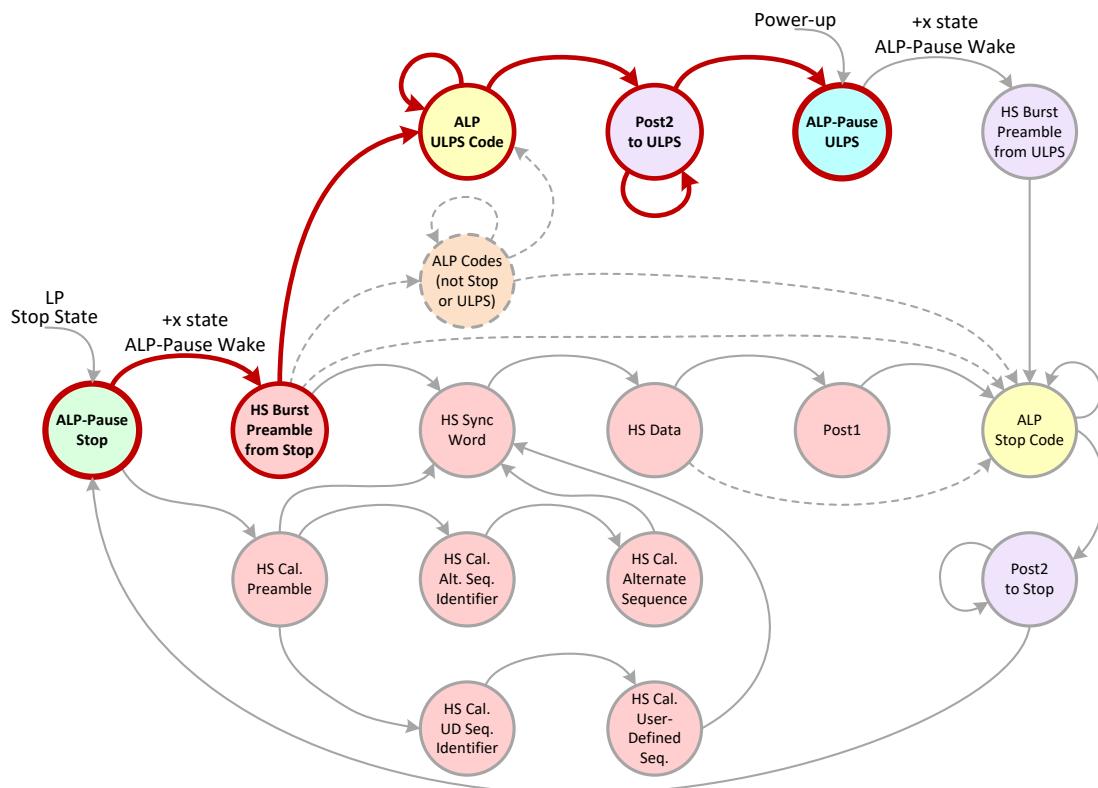


5056

Figure 231 State Transitions for an HS Data Burst

5057
5058

Figure 232 shows the ALP state machine transitions (highlighted in red) necessary to enter the ALP-Pause ULPS state.

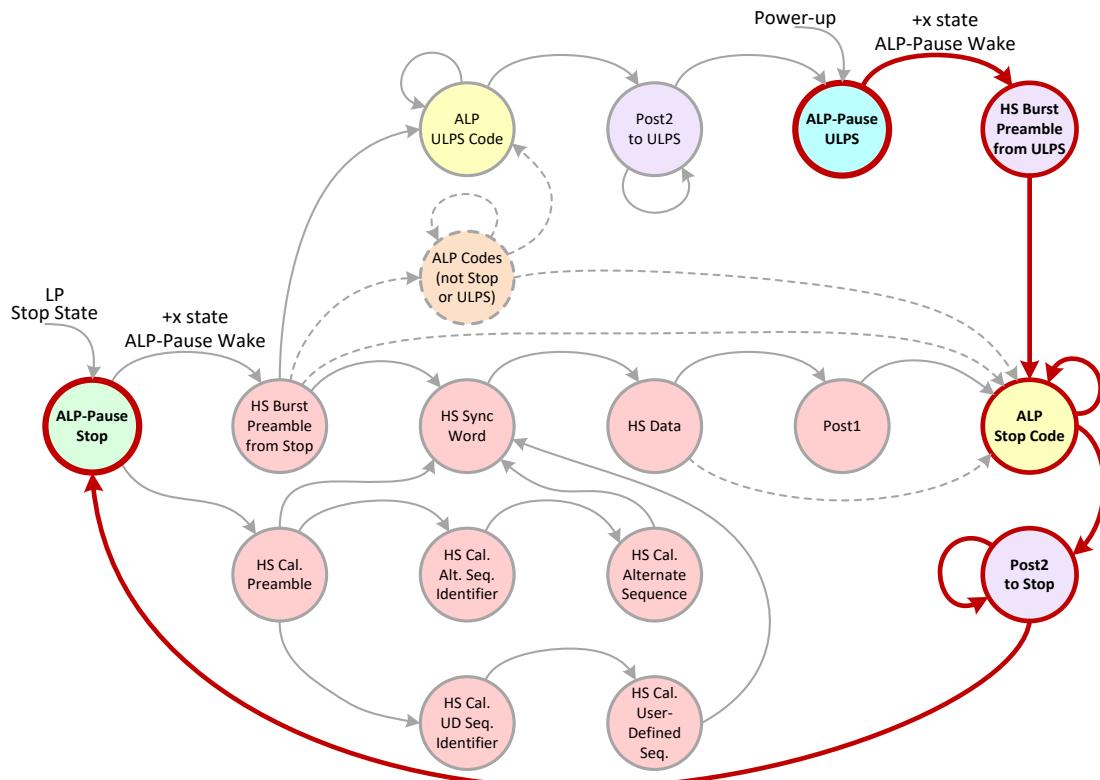


5059

Figure 232 State Transitions to Enter the ULPS State

5060
5061

Figure 233 shows the ALP state machine transitions (highlighted in red) necessary to enter the ALP-Pause Stop state.



5062

Figure 233 State Transitions to Exit from the ULPS State

5063
5064

Table 70 describes the 7-symbol codes transmitted in ALP mode. The corresponding LP mode or Escape mode function is described, where applicable.

5065

Table 70 ALP Code Definitions used by CSI-2

ALP Code	Symbol Sequence	PPI ALP Code	Corresponding LP State or Escape Mode Sequence
Stop Code	0244440	0b0000	LP-111 (End of Transmission, or EoT)
ULPS Code	0244441	0b0001	Escape Mode Entry + Ultra-Low Power State (ULPS)
Post1	4444444	0b1011	No equivalent legacy LP mode sequence exists. The CSI-2 TX can cause the Post sequence to be transmitted by sending this code.
Post2			
Turnaround Code (TAC)	2144441	0b1100	No equivalent legacy LP mode sequence exists, although TAC triggers a Fast Lane Turnaround that is functionally similar to Control Mode Turnaround.

H.1.3 Transmission and Reception of ALP Commands Through the PPI

In ALP mode there are three types of code words transmitted by the PHY:

- **Data:** Data words received from the CSI-2 Tx are mapped through the C-PHY mapper, encoded, and transmitted over the Lane.
- **Sync Words:** The CSI-2 Tx can cause the C-PHY Tx to transmit a Sync Word in place of a data word created by the C-PHY mapper. Sync Words can have one of five different values which are defined as Sync Types.
- **ALP Codes:** The CSI-2 Tx can cause the C-PHY Tx to transmit a specific ALP code which is one of the 7-symbol sequences defined in *Table 70*.

These three different types of code words comprise a high-speed burst while in ALP mode. *Figure 234* highlights the control signals that facilitate the transmission of each of these three different types of code words.

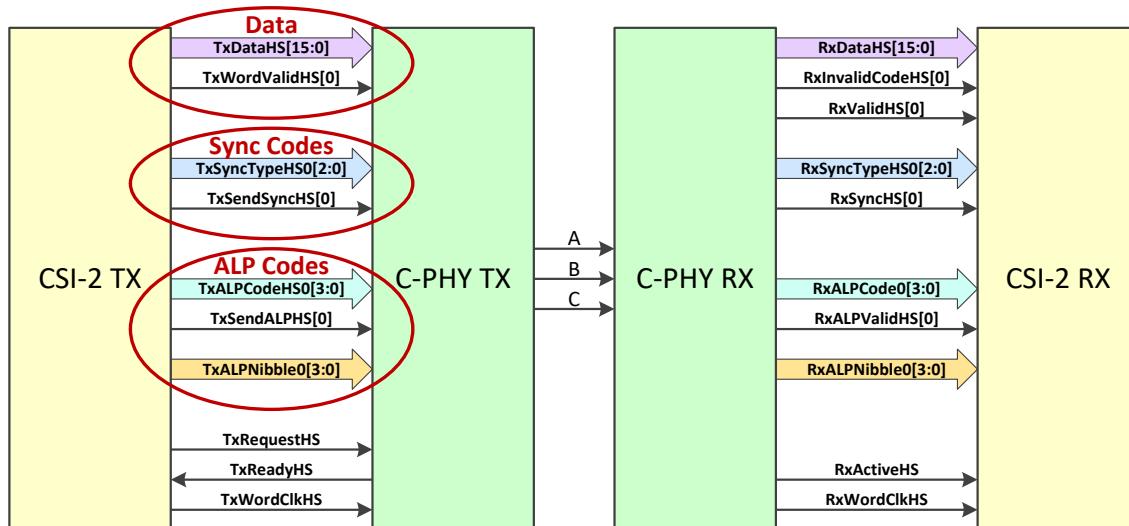


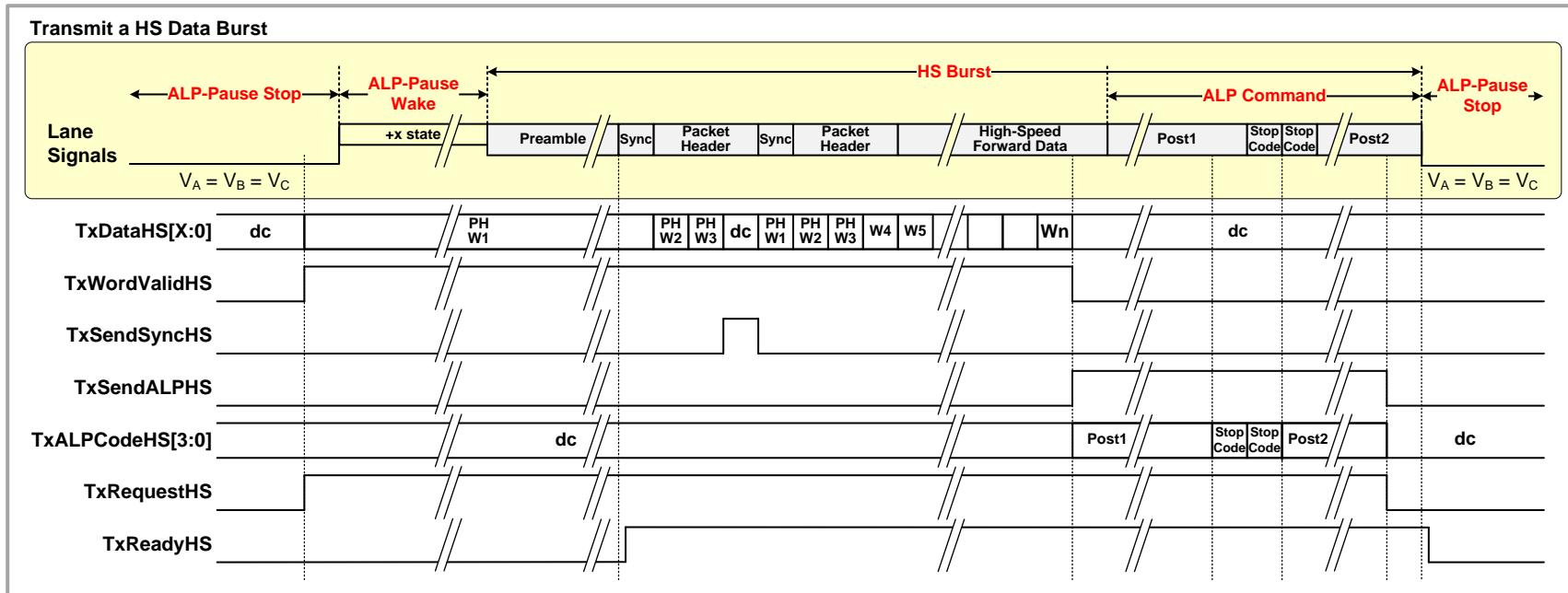
Figure 234 PPI Example: HS Signals for Transmission of Data, Sync and ALP Commands

Figure 235 and *Figure 236* show examples of PPI signals and the corresponding PHY data for transmission and reception of high-speed data in ALP mode. These figures show additional detail of the High-Speed Data Burst waveform in *Figure 230*.

The signal TxRequestHS is asserted simultaneously with TxWordValidHS to request that a high-speed burst be transmitted. The PHY will know to send a data burst because TxWordValidHS is asserted early in the burst timing. This will cause the C-PHY Tx to transmit the first Sync Word at the end of the Preamble. Note that the first Sync Word is transmitted autonomously by the C-PHY Tx, and has the default Sync Type value of 3. Subsequent Sync Words transmitted in a burst are sent as a result of asserting the TxSendSyncHS[0] signal, and the associated Sync Type is defined by the TxSyncTypeHS0[2:0] signals.

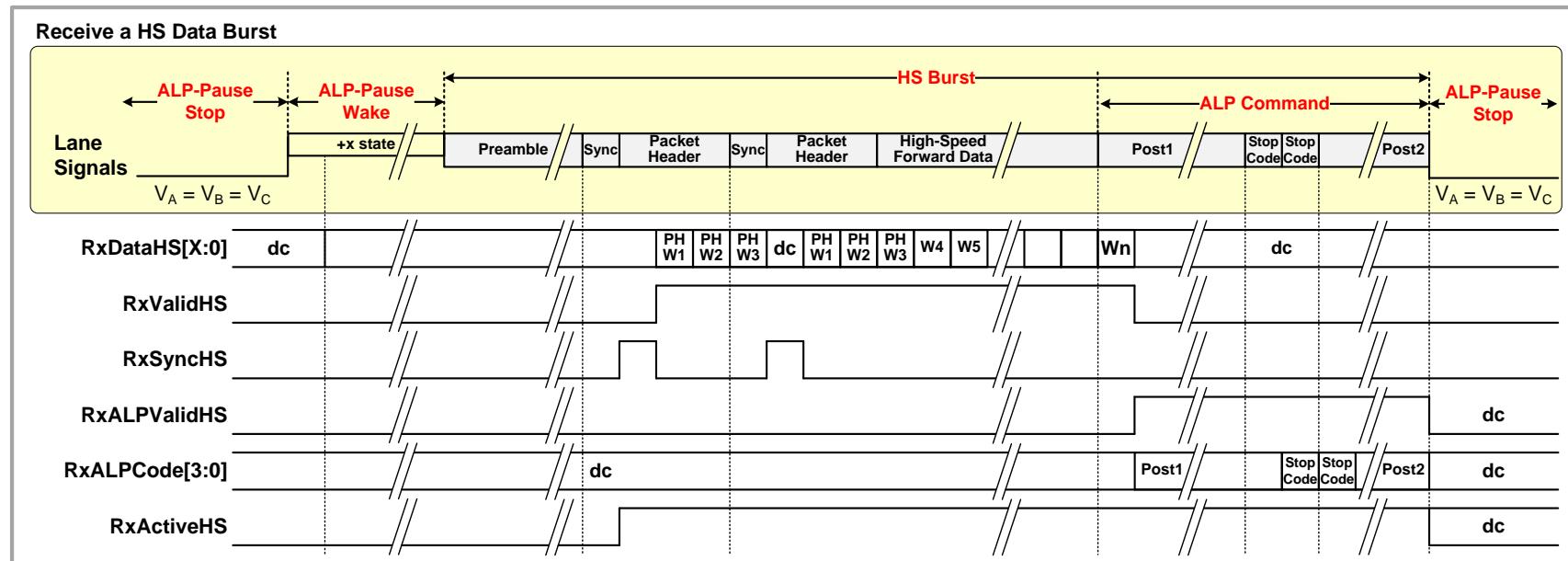
The end of burst in the Transmitter functions differently for ALP mode compared to the non-ALP high-speed mode. In the non-ALP high-speed mode, the end of burst is signaled to the PHY by pulling TxRequestHS low, as described in Annex A of the C-PHY specification. After TxRequestHS goes low, the C-PHY Tx will generate the Post sequence of length determined by a PHY configuration parameter that sets the length of Post.

In ALP mode, the protocol transmit unit generates all fields of the burst after the first sync word, including the packet headers, data burst, Stop Code, ULPS Code, Post1, and Post2. The burst is ended by pulling TxRequestHS low, and no additional data is transmitted on the Lane after this time.



5095

Figure 235 PPI Example Transmit Side Timing for an HS Data Burst

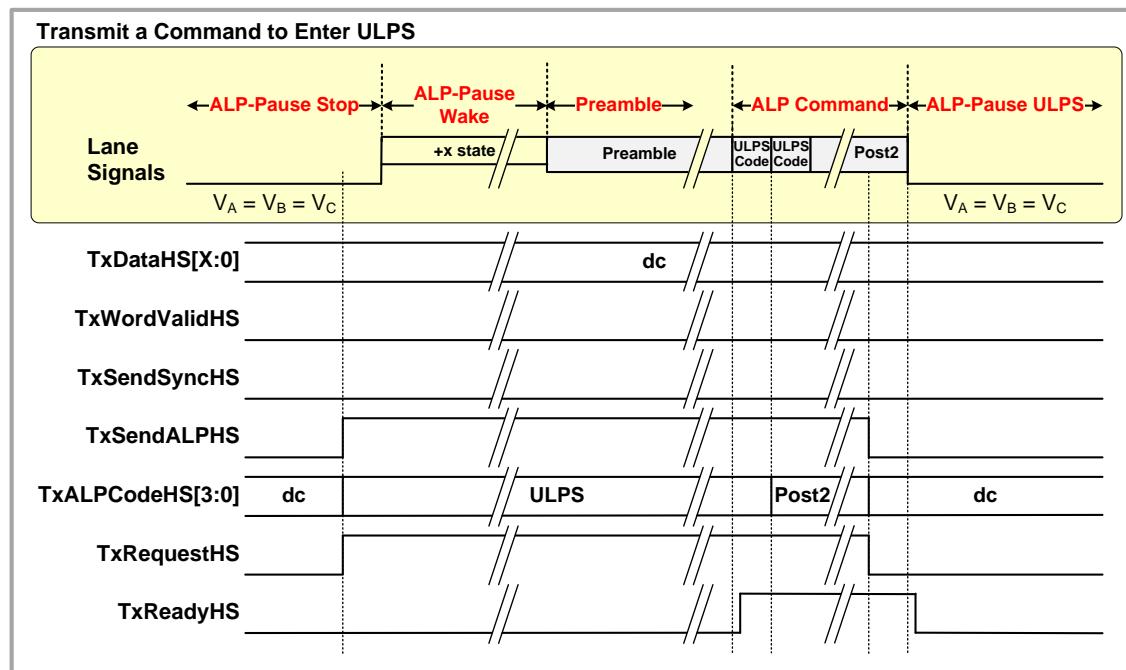


5096

Figure 236 PPI Example Receive Side Timing for an HS Data Burst

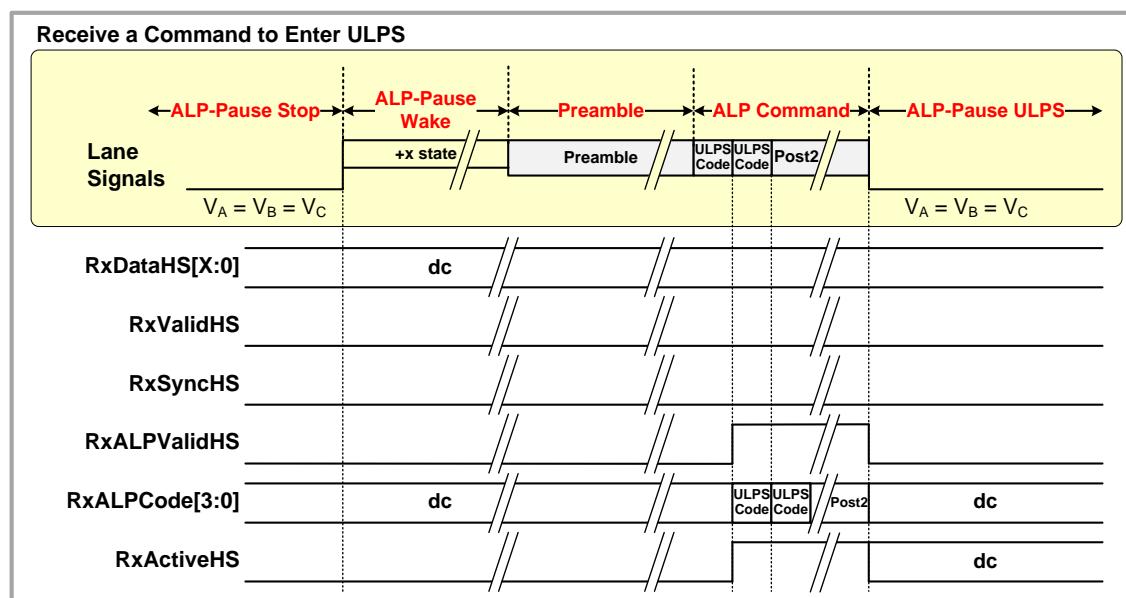
5097
5098
5099
5100 **Figure 237, Figure 238, Figure 239, and Figure 240** show examples of PPI signals and the corresponding PHY data for transmission and reception ALP Commands to enter into and exit from the ALP-Pause ULPS state in ALP mode. These figures show additional detail of the Command to Enter ULPS and the Command to Exit from ULPS waveforms in **Figure 230**.

5101 The signal TxRequestHS is asserted simultaneously with TxSendALPHS to request that a high-speed burst
5102 be transmitted. The PHY will know to send a ALP commands in the burst rather than the Sync Word because
5103 TxSendALPHS is asserted early in the burst timing, and TxWordValidHS is not asserted.



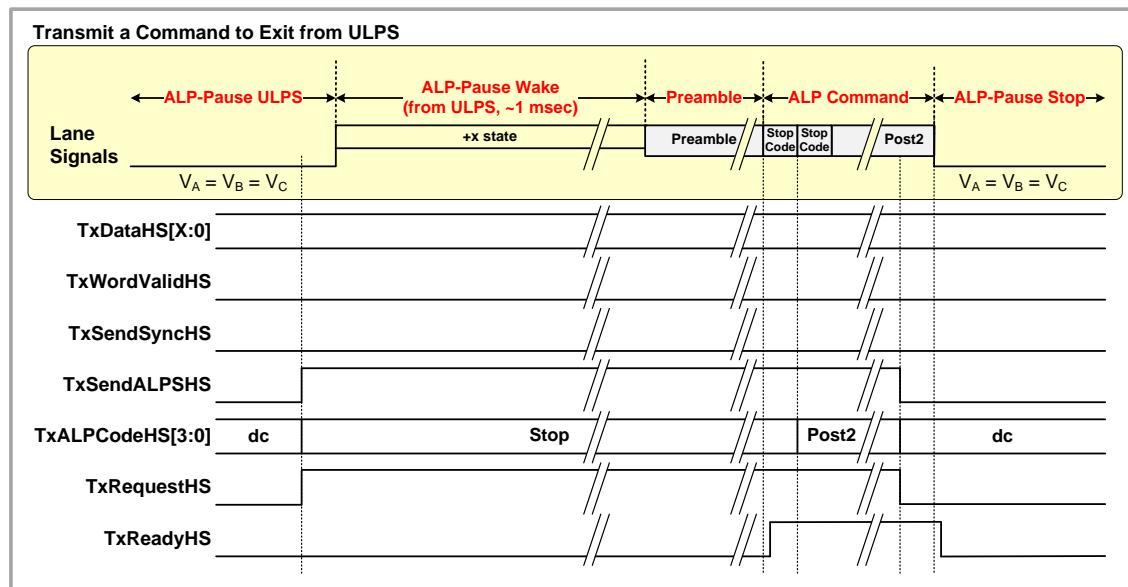
5104

Figure 237 PPI Example Transmit Side Timing to Enter the ULPS State

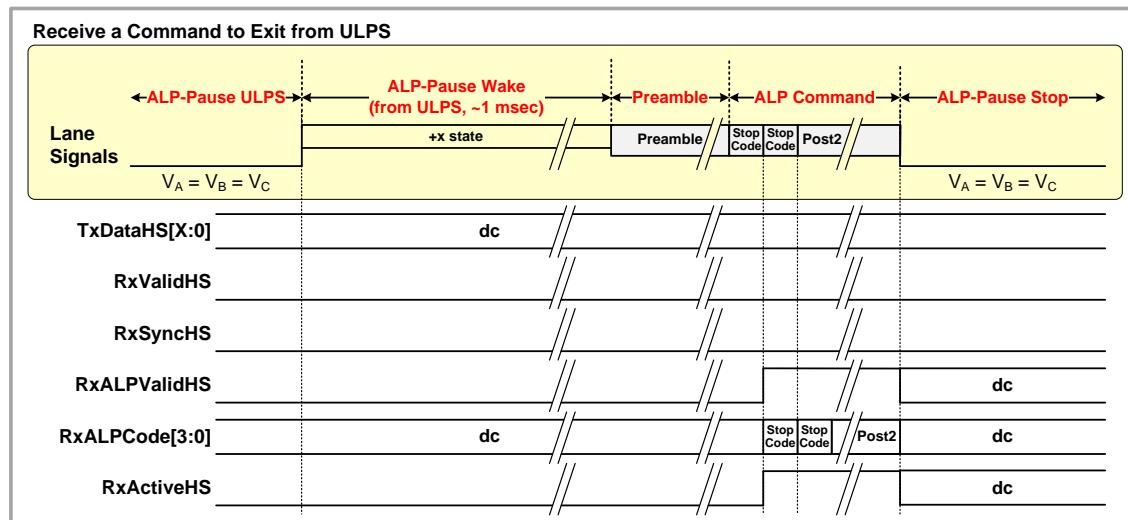


5105

Figure 238 PPI Example Receive Side Timing to Enter the ULPS State



5106

Figure 239 PPI Example Transmit Side Timing to Exit from the ULPS State

5107

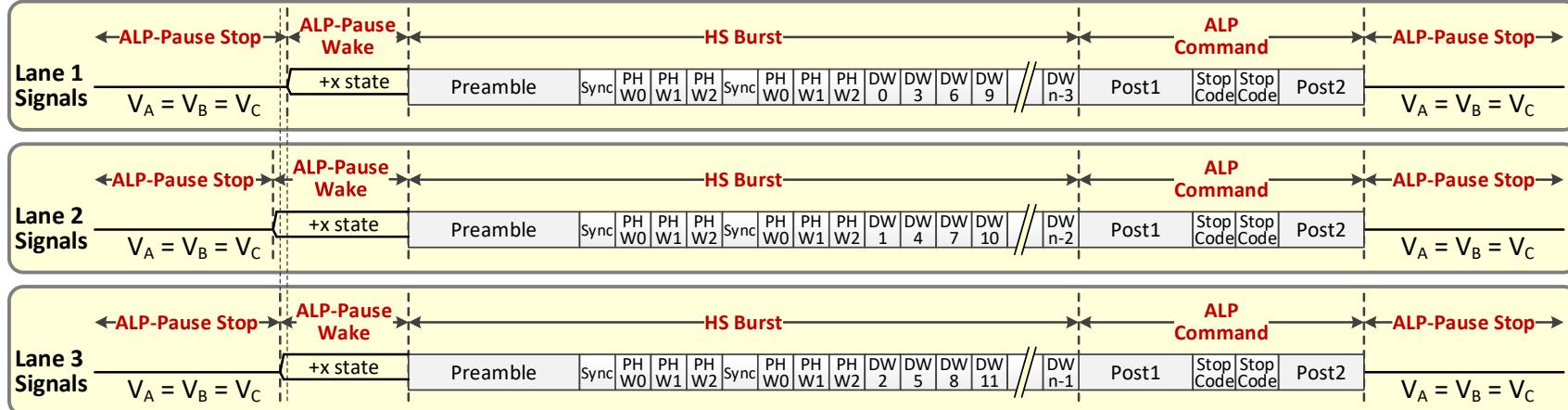
Figure 240 PPI Example Receive Side Timing to Exit from the ULPS State

H.1.4 Multi-Lane Operation Using ALP Mode

5108 *Figure 241* and *Figure 242* show examples of three Lanes operating together in a Link in ALP mode. The
5109 High-Speed data burst in *Figure 241* begins with identical packet headers (consisting of PH W0, PH W1,
5110 and PH W2) transmitted twice on each of the three Lanes. The Packet Headers are followed by packet data
5111 (consisting of DW 0 through DW n-1) striped across the three Lanes by the CSI-2 Lane Distribution Function.
5112 The burst starts and ends in the manner described in Section H.1.2 above. The example of *Figure 242*
5113 showing the command to enter ULPS has identical data on each of the three Lanes.

5114 The example also shows that the assertion of the +x state for ALP-Pause Wake can be staggered in time on
5115 each of the lanes. This is shown to highlight a particular implementation where the system designer might
5116 prefer to enable the high-speed drivers for each of the Lanes at a slightly different time.

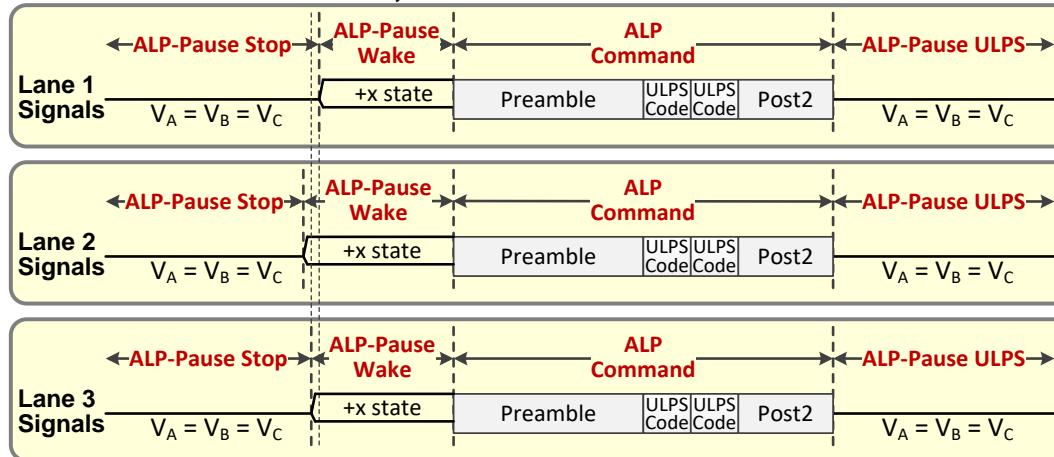
High-Speed Data Burst, Three Lanes



5117

Figure 241 Example Showing a Data Transmission Burst using Three Lanes

Command to Enter ULPS, Three Lanes



5118

Figure 242 Example Showing an ALP Command Burst using Three Lanes

H.1.5 LP and ALP Operation

Section 6.4.5.8 of [MIPI02] describes support of LP and ALP operation. The transmitter and receiver can be configured for LP-only operation, or ALP-only operation using the PPI signals TxALP-LPSelect and RxALP-LPSelect, respectively. Devices can use a variety of means such as a register, an I/O pin, or a non-volatile storage element to determine the initial state of these two PPI signals

H.1.6 Bi-Directional Lane Turnaround

The transmission direction of a Bi-Directional Lane can be swapped by performing a Lane Turnaround procedure. This procedure enables information transfer in the opposite direction of the current direction. The procedure is the same for either a change from Forward Direction to Reverse Direction, or a change from Reverse Direction to Forward Direction. Notice that the roles of Primary and Secondary are not changed as a result of performing the Turnaround procedure.

The Turnaround procedure can be performed in two ways: one is via a Control Mode Lane Turnaround that uses LP Mode signaling, and the other is via a Fast Lane Turnaround.

Control Mode Lane Turnaround occurs by going to the Exit state (Control Mode) where LP Mode signaling is used to perform the Control Mode Turnaround procedure, as shown in **Figure 243**.

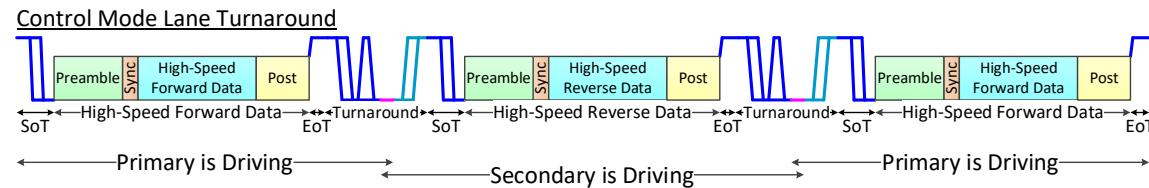


Figure 243 High-Level View of the Control Mode Lane Turnaround Procedure

A somewhat less likely configuration is to combine ALP mode and Control Mode Lane Turnaround if optional Dynamic LP and ALP operation is supported by the C-PHY, and if the electrical specifications can be met with the transmission channel being used in the system. An example is shown in **Figure 244**.

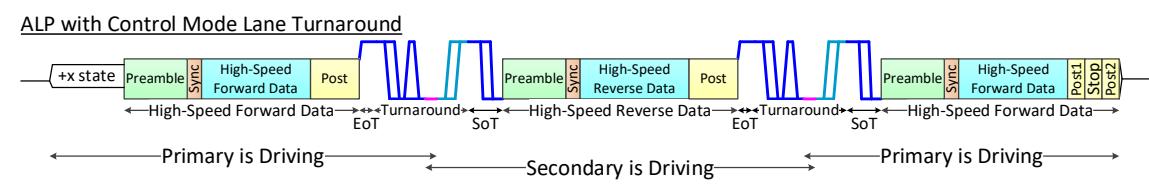


Figure 244 High-Level View of ALP Mode with the Control Mode Lane Turnaround Procedure

Fast Lane Turnaround is the most likely method to be used with the CSI-2 USL feature. Fast Lane Turnaround is performed without having to return to LP mode. This reduces the latency to change the transmission direction of a Bi-directional lane. Fast Lane Turnaround is handled completely in High-Speed mode. One or more Turnaround Codes (TAC) is transmitted near the end of the burst (between Post1 and Post 2) to inform the receiver that the Lane is about to change the transmission direction. A small Turnaround Gap (TGAP) exists between Post2 from the First Transmitting Device and the Preamble from the Second Transmitting Device to allow the Primary and Secondary to swap roles as transmitter and receiver. *Figure 245* shows a high-level view of the Fast Lane Turnaround Procedure with ALP Mode. It is anticipated that the Fast Lane Turnaround will most often be used with ALP Mode, and infrequently with Control Mode.

5146

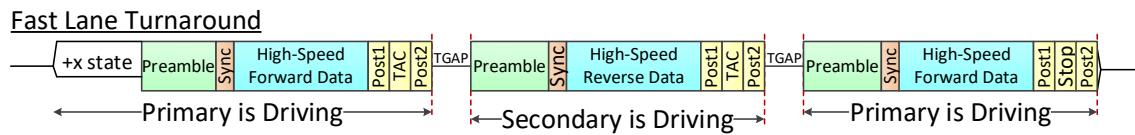
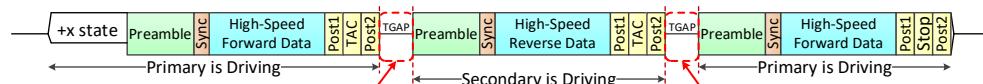


Figure 245 High-Level View of the Fast Lane Turnaround Procedure with ALP Mode

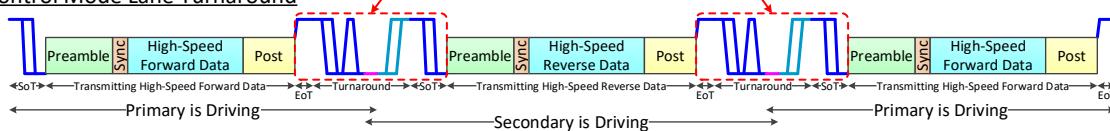
5147
 5148
 5149
 5150
 5151
 5152
 5153
 5154

Figure 246 is a comparison of events that occur in the Fast Lane Turnaround Procedure versus the Control Mode Lane Turnaround Procedure. Fields that are the same color are either the same field in the two waveforms, or have comparable durations. Post1 + TAC + Post2 in the Fast Lane Turnaround is like Post in the Control Mode Lane Turnaround. The Preambles are the same duration and Syncs are the same duration. Fields that cause the durations of the two Turnaround methods to be different are highlighted with “Time difference between Fast Lane Turnaround and Control Mode Lane Turnaround” in the figure. In this case, the TGAP (nominally 14 UI) in the Fast Lane Turnaround waveform is usually much shorter in duration compared to the LP signaling (EoT + Turnaround + SoT, in the Control Mode Lane Turnaround waveform).

Fast Lane Turnaround



Control Mode Lane Turnaround



5155

Figure 246 High-Level View, Comparing Lane Turnaround Procedures

5156
 5157

The Fast Lane Turnaround Procedure is triggered by the transmission of a sequence of ALP codes at the end of a burst.

5158 **Figure 247** shows the detailed sequence of events that occur during the Fast Lane Turnaround Procedure.
 5159 The transmitting C-PHY ends the data burst by sending Post1, followed by the TAC symbol sequence,
 5160 followed by Post2. TAC is the symbol sequence “2144441”, which is an unmapped sequence of seven
 5161 symbols. The least significant symbol of TAC (“1”) is transmitted first and the most significant symbol (“2”)
 5162 is transmitted last, which is the same convention used for transmitting any ALP code word. TAC may be
 5163 transmitted multiple times for improved system reliability, to increase the probability that TAC will be
 5164 detected by the receiver. The Post1 + TAC + Post2 is somewhat similar to the end of a burst in ALP mode,
 5165 except that the TAC Code is sent instead of the Stop Code. The protocol receiver can easily identify the end
 5166 of Packet Data when it detects Post1. The duration of the TAC and Post2 are programmable in the transmitter,
 5167 and this duration is determined by the protocol layer. The protocol layer enables the transmission of Post1,
 5168 TAC, and Post2 via the PPI signals TxSendALPHS[1:0], TxALPCodeHS0[3:0], and TxALPCodeHS1[3:0].

5169 The purpose of Post1 is to indicate the end of the burst and provide a sufficient number of word clock intervals
 5170 so the protocol layer can gracefully stop transmitting and receiving packet data that is sent prior to Post1.
 5171 Post2 exists so the C-PHY transmitter and receiver has a sufficient number of clock intervals following TAC
 5172 to be able to shut down transmitter and receiver circuitry and change the direction of transmission.

5173 A Turnaround Gap (TGAP) exists to allow one transmitter to be disabled before the other is enabled. This
 5174 avoids contention between the two drivers. The First Transmitting Device that sent the TAC disables its
 5175 output by placing the driver into a high-impedance state just after the last symbol of Post2. The C-PHY
 5176 ensures that the transmitter in the First Transmitting Device is completely disabled at the end of TGAP. The
 5177 Second Transmitting Device begins to enable its output after TGAP at the beginning of the Preamble.

5178 When the Second Transmitting Device receives TAC, it starts a timer that is used to determine when the end
 5179 of TGAP, and the beginning of Preamble, should occur. It is necessary for the Second Transmitting Device
 5180 to identify this interval using a timer, because there are no transmissions during TGAP to identify when the
 5181 Second Transmitting Device needs to begin transmitting the Preamble. The number of words of TAC and
 5182 duration of Post2 can be stored in registers in both the First Transmitting Device and Second Transmitting
 5183 Device, so the C-PHY can determine the proper $t_{3-TAC_TO_TX}$ time. This is so the Second Transmitting Device
 5184 starts transmitting Preamble at the proper time at the end of TGAP. The number of words of TAC and duration
 5185 of Post2 can be different when transmission changes from Primary to Secondary, versus from Secondary to
 5186 Primary. Therefore, it is necessary for the Primary and Secondary to have registers related to the TAC duration
 5187 and Post2 duration for both types of turnaround (Primary-to-Secondary and Secondary-to-Primary).

5188 An interval $t_{3-TA-SETTLE}$ exists to allow the driver transmitting the Preamble to have sufficient time to stabilize
 5189 before it is used by the receiver for symbol clock recovery. The $t_{3-TA-SETTLE}$ interval is a time during which
 5190 the HS receiver in the C-PHY will ignore any HS transitions on the Lane. This concept is very similar to the
 5191 $t_{3-SETTLE}$ time at the beginning of a HS burst from LP mode.

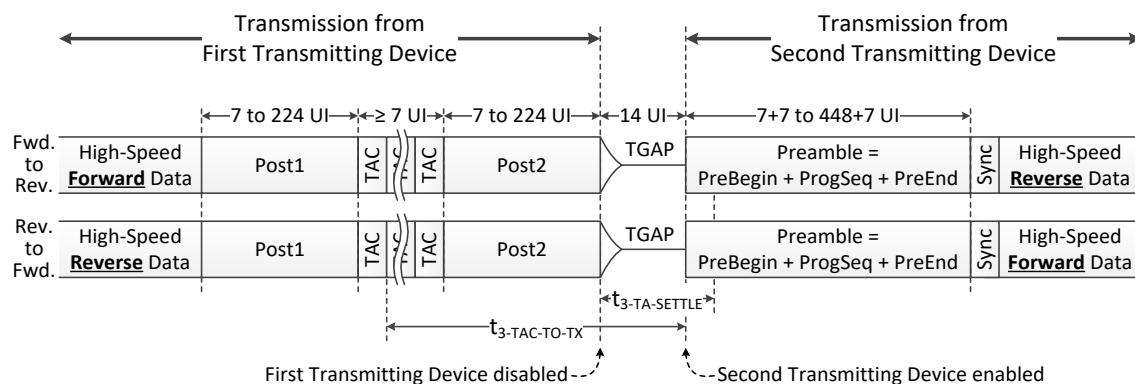


Figure 247 Detailed View of the Fast Lane Turnaround Procedure

5193 **Figure 248** shows an example of ALP state machine transitions (highlighted in red) that occur when a burst
 5194 begins in the conventional manner with an ALP-Pause Wake pulse and finishes by performing a Fast Lane
 5195 Turnaround Procedure. This is the sequence of events that occur during the first “Primary is Driving” interval
 5196 shown in **Figure 245**. The highlighted sequence begins from the ALP-Pause Stop state, and ends when the
 5197 turnaround event occurs during the HS-BTA Rx Wait state. This state diagram presents a high-level view of
 5198 events. It can be interpreted to illustrate states that occur in the transmitter as well as the receiver.

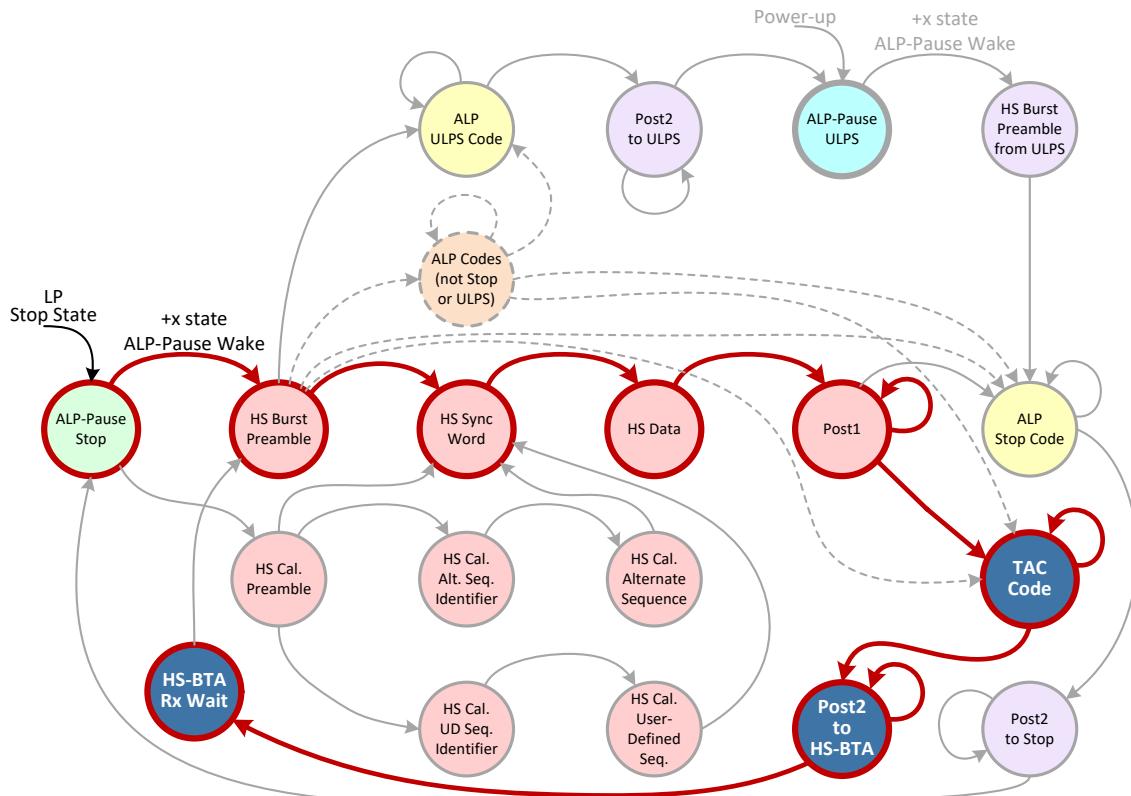
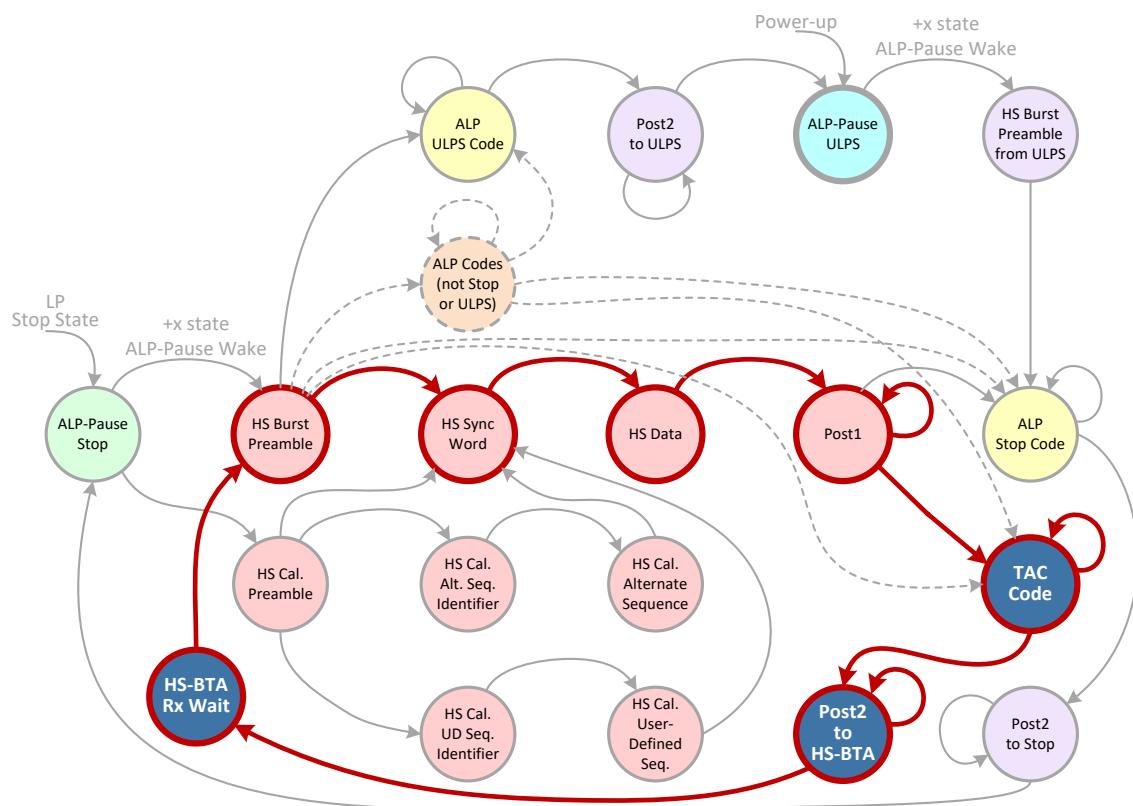


Figure 248 State Transitions from ALP-Pause Stop to Turnaround

5200
5201
5202

Figure 249 shows an example of ALP state machine transitions (highlighted in red) that occur between two Fast Lane Turnaround events. This is the sequence of events that occur during the “Secondary is Driving” interval shown in **Figure 245**.



5203

Figure 249 State Transitions from Turnaround to Turnaround

Figure 250 shows an example of ALP state machine transitions (highlighted in red) that occur, starting from a Fast Lane Turnaround Procedure (the HS-BTA Rx Wait state) and finishing when the burst ends and there is a transition to the ALP-Pause Stop state. This is the sequence of events that occur during the second “Primary is Driving” interval shown in **Figure 245**.

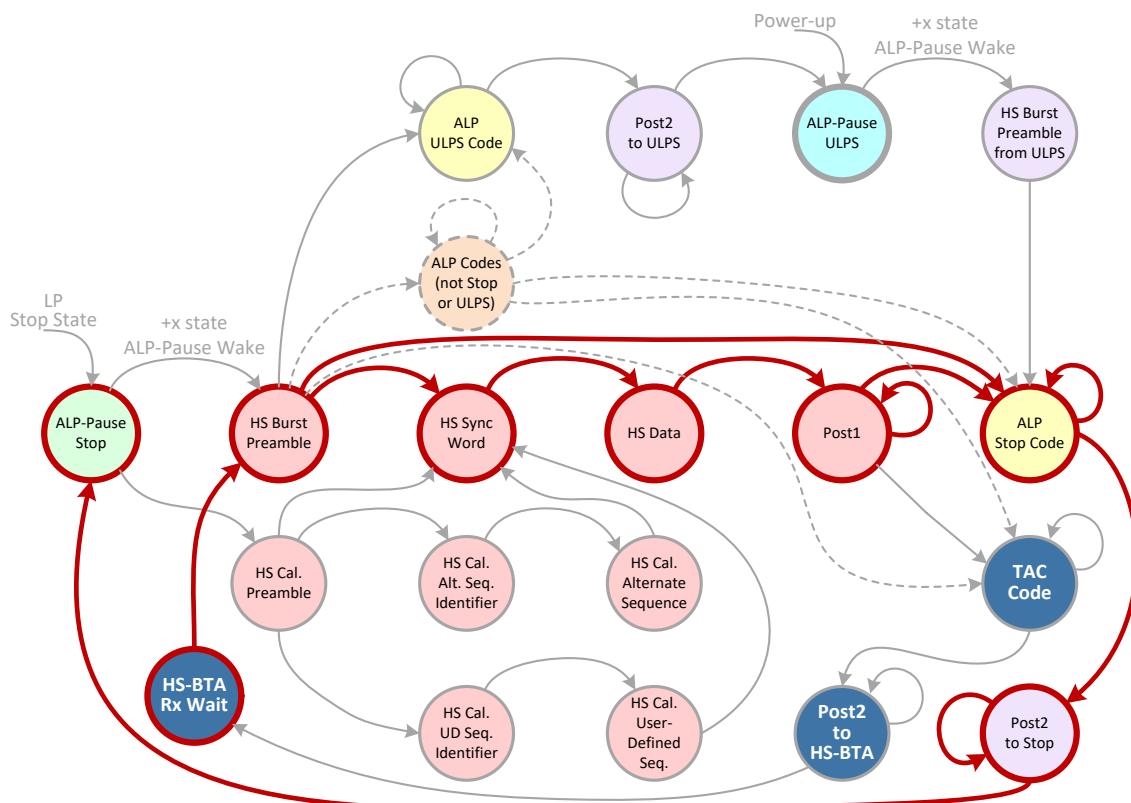


Figure 250 State Transitions from Turnaround to ALP-Pause Stop

The PPI signals that support ALP functions are used to perform a Fast Lane Turnaround. The first transmitting device stops transmission in much the same way as a transmitter signals the end of an ALP burst, except that a TAC code is sent instead of a Stop code. The first transmitting device becomes a receiver after it ceases transmission, so it can immediately switch to receive mode. In this case, the device that becomes a receiver does not need to be triggered by a long ALP Pause Wake pulse prior to the preamble. Instead, the trigger for assuming the role of a receiver is the transmission of the TAC code prior to the TGAP interval.

5215
5216 **Figure 251** shows an example of the transmitter and receiver PPI signaling in the first transmitting device when a Fast Lane Turnaround occurs.

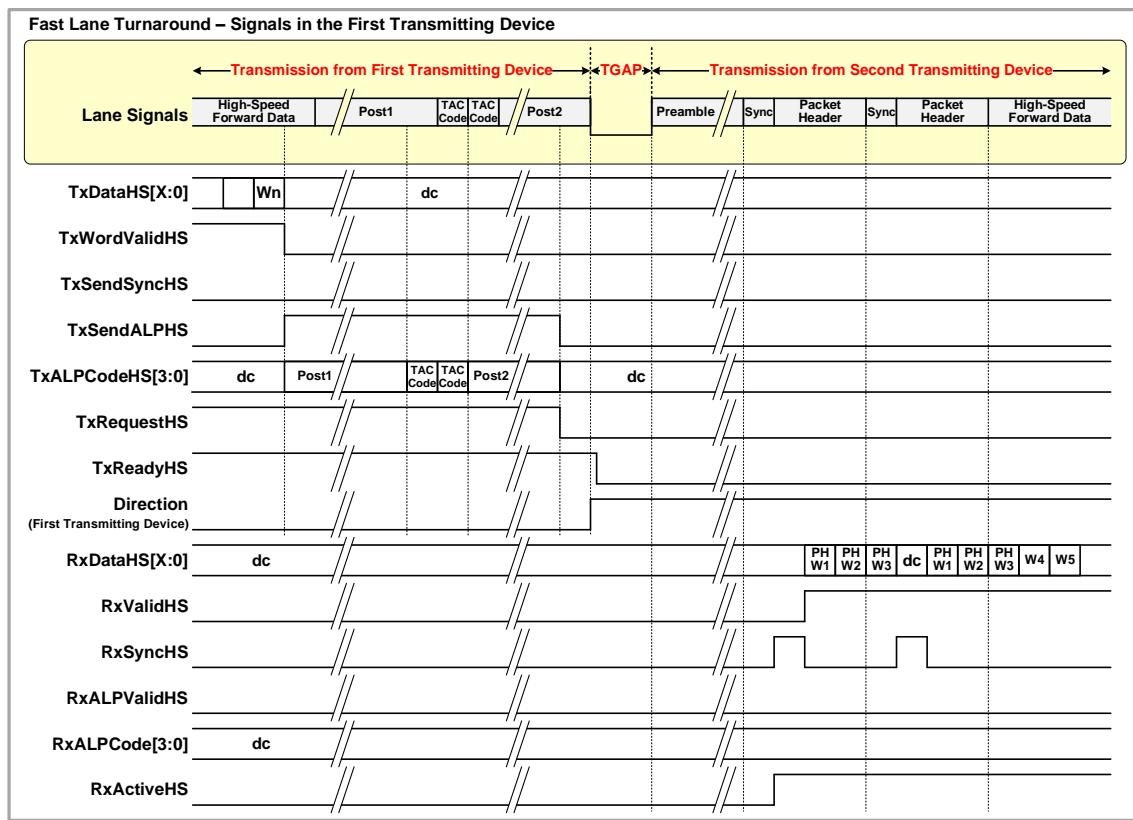
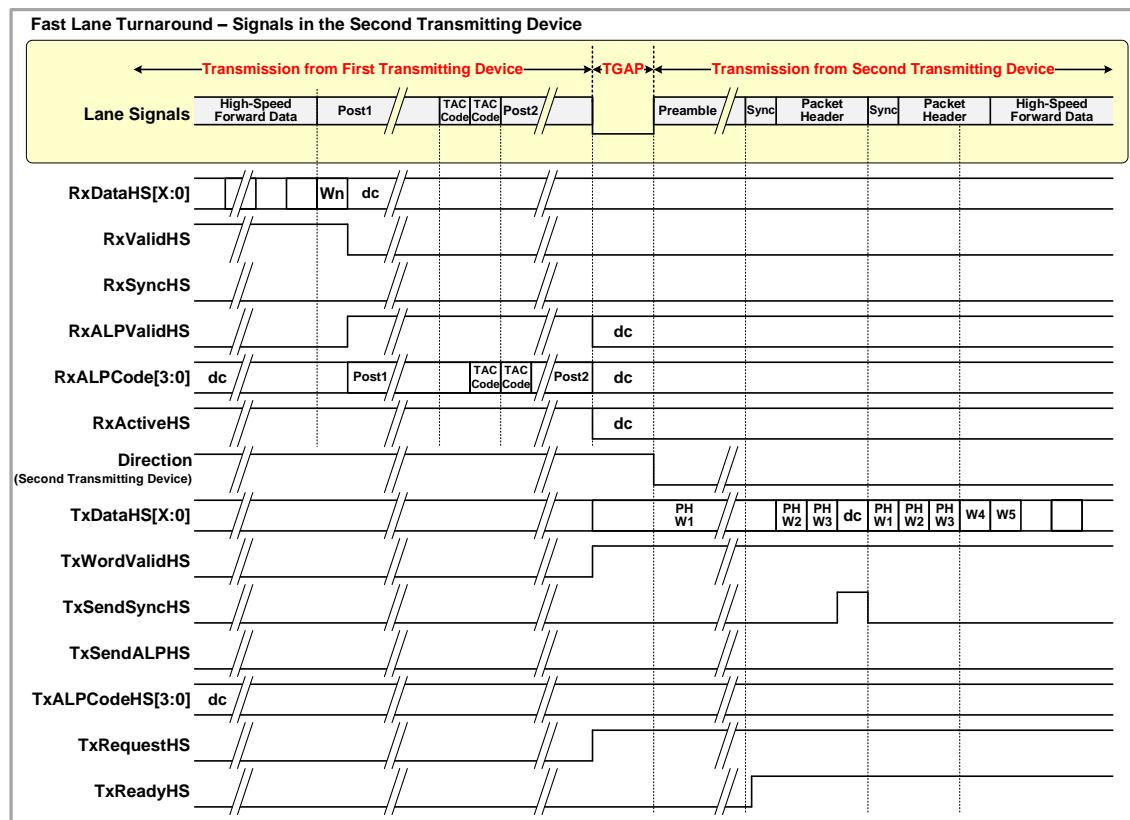


Figure 251 Example Fast Lane Turnaround at the First Transmitting Device

5218
5219 **Figure 252** shows an example of the receiver and transmitter PPI signaling in the second transmitting device when a Fast Lane Turnaround occurs.



5220 **Figure 252 Example Fast Lane Turnaround at the Second Transmitting Device**

Annex I Multi-Pixel Compression (MPC)

Multi-Pixel image sensors such as Tetra-Cell and Nona-Cell image sensors have been developed as image resolution has increased. As the amount of data being transmitted has increased, the importance of efficient transmission has grown. For improved efficiency, CSI-2 supports the MPC (Multi-Pixel Compression) data compression scheme for Multi-Pixel images.

MPC can compress not only Multi-Pixel images, but also the common Bayer pattern image. Compared with the common Bayer pattern image, a Multi-Pixel image has additional compression opportunities because neighboring pixels with the same color filter are highly correlated. MPC outperforms conventional compression by utilizing this redundancy.

MPC is specified for Bayer, Tetra-Cell, and Nona-Cell image sensors, as shown in *Figure 253*. MPC provides three compression schemes having 10:5, 10:6, and 10:7 compression ratios for Tetra-Cell, and Nona-Cell image sensors. Bayer image sensors have only a single compression scheme with a 10:5 compression ratio.

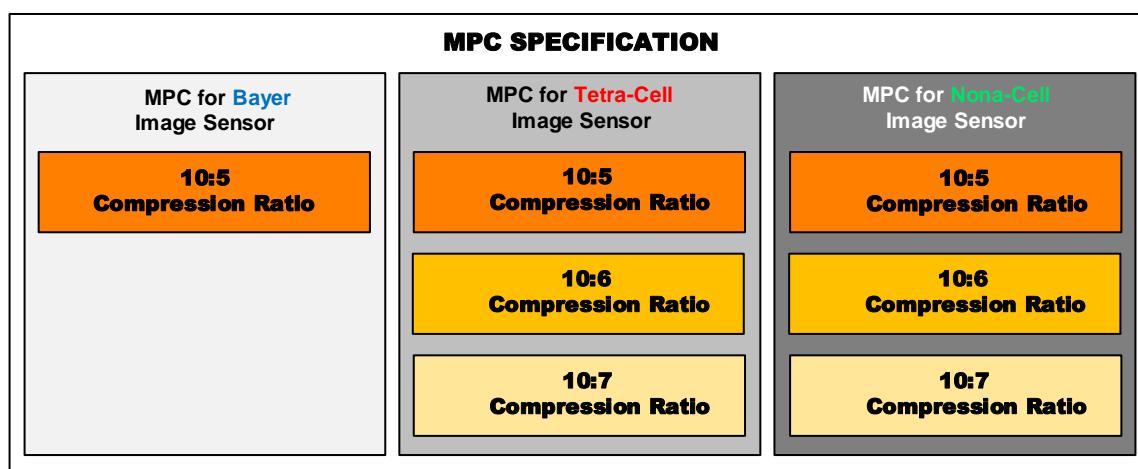


Figure 253 MPC Data Compression Schemes

To support high image quality even at high compression ratios, MPC features the following two Prediction Mode types:

- **Basic Prediction Mode** includes a simple algorithm to handle image details oriented in a specific direction, or basic cases of differences between neighboring pixels.
- **Advanced Prediction Mode** has a more extensive algorithm for better direction awareness.

MPC Packet data is identified using one of the User Defined data type codes in *Table 61*. Note that User Defined data type codes are not reserved for compressed data types. Therefore, it is recommended that the assignment of specific User Defined data type codes to compression schemes be communicated to the image sensor via the CCI. The protocol and CCI register address to map a data compression scheme to a Data Type code are beyond the scope of this specification.

The number of bits in an MPC Packet shall be a multiple of eight. Therefore, implementations with data compression schemes that result in each pixel having other than eight encoded bits per pixel shall transfer the encoded data using a packed pixel format. For example, the 10–5–10 data compression scheme cannot use the RAW6, RAW7, or RAW8 packed pixel formats described in *Section 11.4*. For this example, the RAW 10 format described in *Section 11.4.2* is appropriate, except that the Data Type value in the Packet Header is a User Defined data type code. MPC with a Tetra-Cell sensor uses the RAW10, RAW12, or RAW14 formats depending on the data compression ratio. MPC with a Nona-Cell image sensor uses the RAW10, RAW12, or RAW14 formats by concatenating two MPC Blocks of encoded data.

The MPC encoded data stream shall consist of a series of Blocks, where each Block represents one compression processing unit. Each Block shall consist of a 4-bit Header and a Payload whose length shall depend upon the image sensor type (i.e., Bayer, Tetra-Cell, or Nona-Cell) and the data compression ratio as defined by **Table 71**. The number of pixels encoded per Block shall be 4 (1x4) for Bayer image sensors, 4 (2x2) for Tetra-Cell image sensors, and 9 (3x3) for Nona-Cell image sensors.

Example: The MPC Packet for Tetra-Cell image sensors always has 4 compressed pixels per Block and a 4-bit Header. The Payload size is either 16 bits for compression ratio 10:5, or 20 bits for 10:6 compression, or 24 bits for 10:7 compression.

Table 71 MPC Bits Per Block and CSI-2 Image Data Format

Image Sensor Type	Pixels Per Block	Bits Per Block			CSI-2 RAW Image Data Format		
		Header Size (bits)	Payload Size (bits)				
			10:5	10:6	10:7		
Bayer	1x4	4	16	N/A	N/A	RAW10 User Defined	N/A
Tetra-Cell	2x2	4	16	20	24	RAW10 User Defined	RAW12 User Defined
Nona-Cell	3x3	4	41	50	59	RAW10 (2 Blocks) User Defined	RAW12 (2 Blocks) User Defined
						RAW14 (2 Blocks) User Defined	RAW14 (2 Blocks) User Defined

Figure 254 illustrates the RAW10 transmission scheme for MPC with Bayer image sensors. The MPC encoded data stream from the Bayer pixel array includes the Header and the Payload, which can be divided into STR n and STR n+1 to fit into the RAW10 transmission format: STR n includes the Header and some of the Payload, and STR n+1 includes the rest of the Payload.

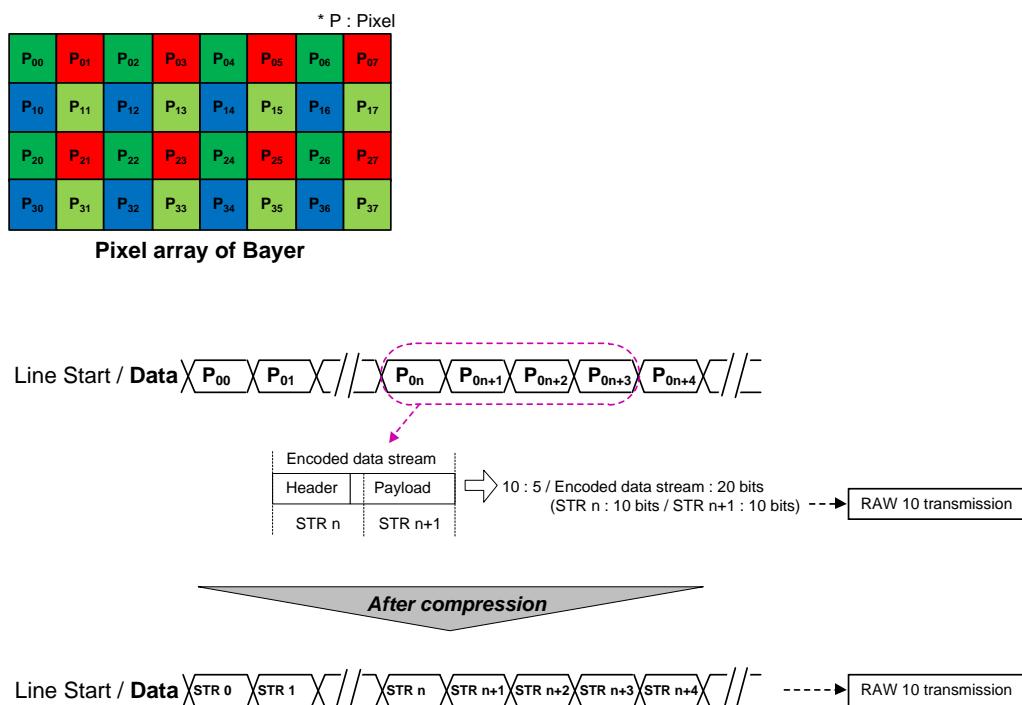
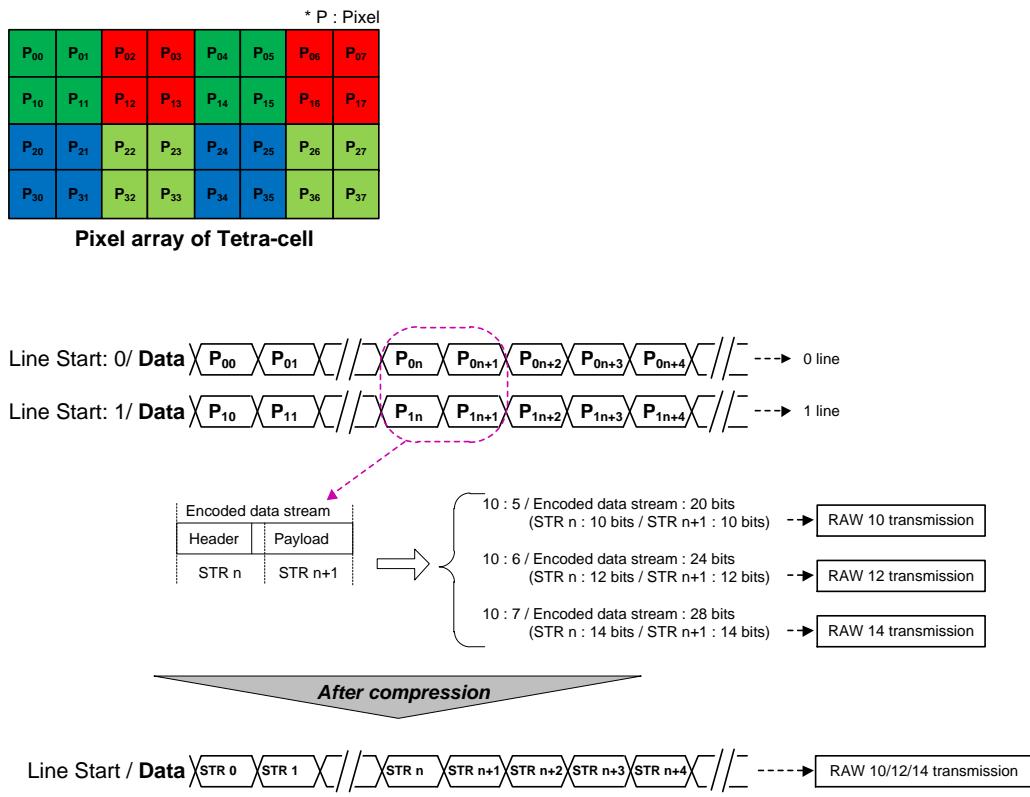


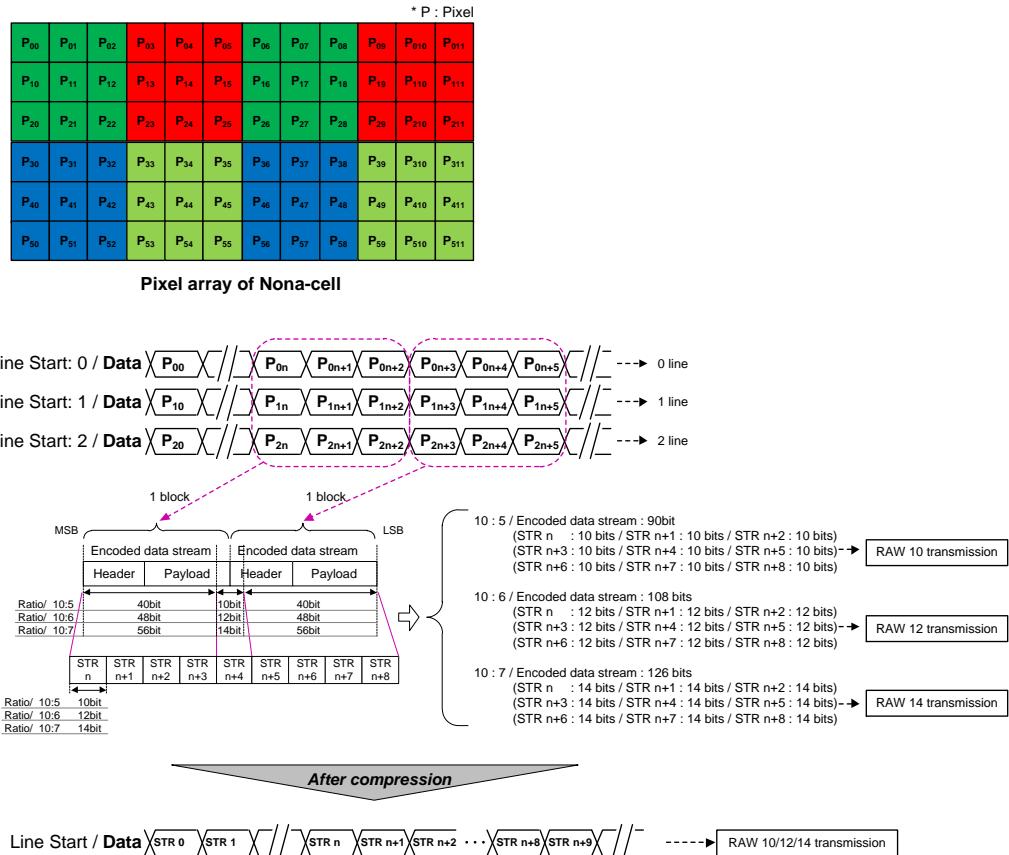
Figure 254 RAW10 Transmission for MPC for Bayer Image Sensors

5265
5266
5267
5268 **Figure 255** illustrates the RAW10/12/14 transmission scheme for MPC for Tetra-Cell image sensors. The MPC encoded data stream from the Tetra-Cell pixel array includes the Header and the Payload, which can be divided into STR n and STR $n+1$ to fit into the RAW10/12/14 transmission format. STR n includes the Header and some of Payload, and STR $n+1$ includes the rest of the Payload.



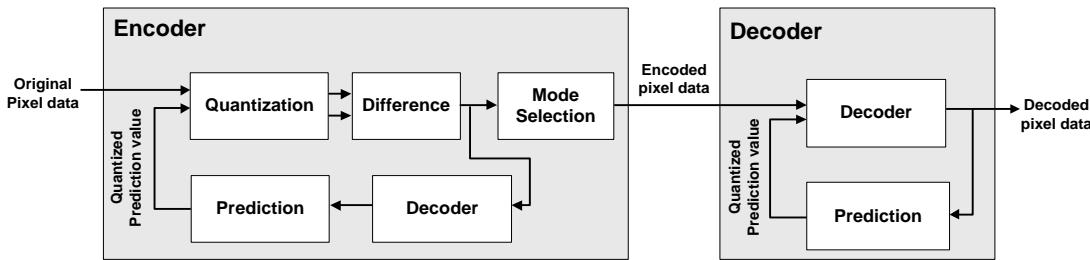
5269 **Figure 255 RAW10/12/14 Transmission for MPC for Tetra-Cell Image Sensors**

5270 **Figure 256** illustrates the RAW10/12/14 transmission scheme for MPC for Nona-Cell image sensors. The
 5271 MPC encoded data stream from the Nona-Cell pixel array includes the Header and the Payload, which can
 5272 be divided into the series STR n , STR $n+1$, ..., STR $n+8$ to fit into the RAW10/12/14 transmission format.
 5273 With this division method, STR n , STR $n+4$, and STR $n+5$ each include a Header and some of the Payload,
 5274 and STR $n+1$, STR $n+2$, STR $n+3$, STR $n+6$, STR $n+7$, and STR $n+8$ include the rest of the Payload.



5275 **Figure 256 RAW10/12/14 Transmission for MPC for Nona-Cell Image Sensors**

5276
5277 The MPC data compression system consists of an MPC Encoder and an MPC Decoder as shown in **Figure 257**.



5278 **Figure 257 MPC Data Compression System**

5279 To maintain constant bandwidth, MPC encodes data differences using a lossy scheme. The MPC Encoder
 5280 uses a variety of defined Compression Modes to improve image quality for object edge patterns. In the MPC
 5281 Encoder, each Compression Mode encodes the original pixel data based on direction awareness. The MPC
 5282 Encoder uses direction awareness to automatically select the Compression Mode that is the best choice for
 5283 minimizing data loss. Pixel values at the beginning of the first image line are encoded using simple prediction.
 5284 These first few pixels can use User Control Parameters for boundary fill (see **Section I.1.1.1**, **Section I.2.1.1**,
 5285 and **Section I.3.1.1**), so prediction does not need to be initialized for them. The remaining pixel values except
 5286 for the first image line are then encoded using normal prediction methods.

5287 The first step in MPC decoding is to calculate the prediction value, based on the Compression Mode that the
 5288 MPC Encoder used. In the MPC Packet, this is indicated via Header fields **PredictionIDX** and **ModeIDX**.
 5289 When performing prediction, previously decoded pixels are used as reference pixels and the image is
 5290 reconstructed using difference values relative to those reference pixels. After the prediction the reference
 5291 pixel is quantized, again based on the Compression Mode that the MPC Encoder used. The value after
 5292 quantization is added to the difference value (i.e., the incoming compressed data for that pixel). The MPC
 5293 Decoder's final output is the reconstructed image, i.e., the decoded image after de-quantization.

5294 The following sections define the MPC Compression Modes for each image sensor type (i.e., Bayer, Tetra-
 5295 Cell, and Nona-Cell), and describe the decoding process for each Compression Mode.

5296 **Note:**

5297 *Example code implementing MPC decompression is available from the MIPI Members site [[MIPI07](#)].*

I.1 MPC for Bayer Image Sensor

MPC for Bayer image sensors supports the seven Compression Modes that are defined in this Section and summarized in **Table 72**. If MPC is supported, then all seven MPC decoder Data Compression Modes shall be supported. Each Compression Mode uses either Basic Prediction Mode or Advanced Prediction Mode. **Figure 258** summarizes the MPC Packet Structures used for Bayer Compression Modes.

Table 72 MPC Compression Modes for Bayer Image Sensors

Compression Mode Name	Mode Description		Defined in Section
Basic Prediction Mode:			
PD	Pixel-based <u>D</u> irectional <u>D</u> ifferential Mode		<i>I.1.2.1</i>
DGD	<u>D</u> ia <u>G</u> onal <u>D</u> irection-based Differential Mode		<i>I.1.2.1</i>
FNR	<u>F</u> ixed <u>Q</u> uantization and <u>N</u> o- <u>R</u> eference Mode		<i>I.1.2.3</i>
OUT	<u>O</u> T <u>L</u> ier compensation Mode		<i>I.1.2.3</i>
Advanced Prediction Mode:			
eDGD	<u>e</u> xtended <u>D</u> ia <u>G</u> onal <u>D</u> irection-based Differential Mode		<i>I.1.3.1</i>
ePD	<u>e</u> xtended <u>P</u> ixel-based <u>D</u> irectional <u>D</u> ifferential Mode		<i>I.1.3.2</i>
eSHV	<u>e</u> xtended <u>S</u> lanted <u>H</u> orizontal or <u>V</u> ertical <u>D</u> irection-based Differential Mode		<i>I.1.3.3</i>

Mode Summary for Bayer Image Sensor							
Compression Mode	Header			Payload			
	Prediction IDX	Mode IDX					
<i>Basic Prediction Mode</i>							
PD	0	0	Qstep[1:0]	Diff0[3:0]	Diff1[3:0]	Diff2[3:0]	Diff3[3:0]
DGD	0	1	0	Qstep	Diff0[3:0]	Diff1[3:0]	Diff2[3:0]
FNR	0	1	1	0	Pixel0[3:0]	Pixel1[3:0]	Pixel2[3:0]
OUT	0	1	1	1	Pos[1:0] Qstep[1:0]	Diff1[3:0]	Diff2[3:0]
<i>Advanced Prediction Mode</i>							
eDGD	1	0	Qstep[1:0]	Diff0[3:0]	Diff1[3:0]	Diff2[3:0]	Diff3[3:0]
ePD	1	1	0	Qstep	Diff0[3:0]	Diff1[3:0]	Diff2[3:0]
eSHV	1	1	1	Qstep	Diff0[3:0]	Diff1[3:0]	Diff2[3:0]

5303

Figure 258 MPC Packet Structures for Bayer Compression Modes

I.1.1 Reference Pixels for Prediction (Bayer)

MPC compresses each pixel by referring to neighboring pixels. The reference pixels are selected according to the Compression Mode and the color of the current pixel.

In a Bayer image, a given pixel is expressed as $p(i, j)$, where:

- i : Horizontal index of the pixel
- j : Vertical index of the pixel

For a current pixel $p(h, v)$, coordinates relative to a reference pixel are expressed as:

- The Horizontal relative index from the current pixel to the reference pixel
 - Reference pixels to the left of the current pixel have negative h indexes
 - Reference pixels directly above or below the current pixel have zero h indexes
 - Reference pixels to the right of the current pixel have positive h indexes

and

- The Vertical relative index from the current pixel to the reference pixel
 - Reference pixels located in image lines above the current pixel have negative v indexes
 - Reference pixels in the same image line as the current pixel have zero v indexes

Examples:

$p(h, v-1)$ Pixel immediately above the current pixel

$p(h-1, v)$ Pixel immediately to the left of the current pixel

$p(h+1, v-1)$ Pixel immediately above and immediately to the right of the current pixel

$p(h-1, v-1)$ Pixel immediately above and immediately to the left of the current pixel

In the following sub-sections, all reference pixel coordinates are expressed using this notation.

5324
5325

Using this notation, the relative coordinates of reference pixels are as illustrated in **Figure 259**, where the color of each square indicates the color filter of the corresponding pixel.

Green Pixel First

	$p(h-2v-2)$	$p(h-1v-2)$	$p(h\ v - 2)$	$p(h+1v-2)$	$p(h+2v-2)$	$p(h+3v-2)$	$p(h+4v-2)$	$p(h+5v-2)$	
	$p(h-2v-1)$	$p(h-1v-1)$	$p(h\ v - 1)$	$p(h+1v-1)$	$p(h+2v-1)$	$p(h+3v-1)$	$p(h+4v-1)$	$p(h+5v-1)$	
	$p(h-2, v)$	$p(h-1, v)$	$p(h, v)$	$p(h + 1, v)$	$p(h + 2, v)$	$p(h + 3, v)$			

Pixels to be Encoded

Blue Pixel First

	$p(h-2v-2)$	$p(h-1v-2)$	$p(h\ v - 2)$	$p(h+1v-2)$	$p(h+2v-2)$	$p(h+3v-2)$	$p(h+4v-2)$	$p(h+5v-2)$	
	$p(h-2v-1)$	$p(h-1v-1)$	$p(h\ v - 1)$	$p(h+1v-1)$	$p(h+2v-1)$	$p(h+3v-1)$	$p(h+4v-1)$	$p(h+5v-1)$	
	$p(h-2, v)$	$p(h-1, v)$	$p(h, v)$	$p(h + 1, v)$	$p(h + 2, v)$	$p(h + 3, v)$			

Pixels to be Encoded

Not Shown: Red Pixel First

5326
5327
5328
5329

Figure 259 Reference Pixel Relative Coordinate Indexes (Bayer)

For a Bayer image sensor, the four highlighted pixels in the bottom row will be encoded. This corresponds to pixel coordinates $p(h, v)$ (i.e., the current pixel), $p(h+1, v)$, $p(h+2, v)$, and $p(h+3, v)$. The other colored squares represent the previously decoded pixels, which are available to be used as reference data.

I.1.1.1 User Control Parameters for Boundary Fill

At the boundary of the image frame, no neighboring reference pixels exist. As a result, difference calculations for pixels at the image boundary cannot depend upon the values of neighbor pixels. To handle this case, MPC allows the boundary to be filled with user control parameters. This feature makes prediction possible at the image boundary despite the absence of actual prediction pixels. The user control parameter should be calculated before the start of a new image frame by using sensor information such as the luminance value of the image, or the gain value of the sensor. For example, the user control parameter can be set to the value 64 in low-light conditions or the value 128 for normal conditions.

I.1.2 Basic Prediction Mode (Bayer)

For Bayer image sensors, Basic Prediction Mode:

- Compresses the image using a simple algorithm to handle image details oriented in a specific direction, or basic cases of differences between neighboring pixels.
- Supports Compression Modes PD, DGD, FNR, and OUT as defined below.

I.1.2.1 Standard Mode 0: PD (Pixel-based Directional Differential Mode)

Figure 260 defines the packet structure for Compression Mode PD. For this Compression Mode, Header field **Prediction IDX** shall contain 0, and the high bit of Header field **Mode IDX** shall contain 0.

The Payload shall contain compressed data for four pixels:

- Field **Qstep** shall indicate the quantization step size.
- Fields **Diff0** through **Diff3** shall contain the encoded data, i.e., for each pixel, the difference between the predicted pixel and each original pixel.

Table 73 shows pseudo-code for decoding an MPC Packet for Compression Mode PD.

Mode	Header			Payload			
	Prediction IDX	Mode IDX					
PD	0	0	Qstep[1:0]	Diff0[3:0]	Diff1[3:0]	Diff2[3:0]	Diff3[3:0]

Figure 260 MPC Packet Structure for Compression Mode PD

5349

Note:

5350

5351 This pseudo-code assumes that the Green pixel appears first in the image. If the first pixel in the
 5352 image is Red or Blue then the processing sequence must be changed, but compression is performed
 in the same way for each color.

5353

Table 73 Decoder for Compression Mode PD

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 0 : (Qstep==1) ? 1 : (Qstep==2) ? 2 : 3
3: // For decoding the pixel p(h,v)
4: if (Line count < 2) // Vertical line counter, Boundary condition
5:   q(h,v) = p(h-2,v)
6: else if (unit count == 0) // unit count is for processing unit block, boundary condition
7:   q(h,v) = p(h,v-2)
8: else //Prediction value q of the preview pixel
9:   if p(h-2,v-2) >= max{p(h,v-2), p(h-2,v)}
10:    q(h,v) = min{p(h,v-2), p(h-2,v)}
11:   else if p(h-2,v-2) <= min{p(h,v-2), p(h-2,v)}
12:    q(h,v) = max{p(h,v-2), p(h-2,v)}
13:   else
14:     q(h,v) = p(h-2,v) + p(h,v-2) - p(h-2,v-2)
15:   end if
16: end if
17: p(h,v) = ((q(h,v) >> shift0 + Diff0) << shift0) + (1 << (shift0 -1))
18: // For decoding the pixel p(h+1,v)
19: if (Line count < 2) // Vertical line counter, Boundary condition
20:   q(h+1,v) = p(h-1,v)
21: else if (unit count == 0) // unit count is for processing unit block, boundary condition
22:   q(h+1,v) = p(h+1,v-2)
23: else //Prediction value q of the preview pixel
24:   if p(h-1,v-2) >= max{p(h+1,v-2), p(h-1,v)}
25:     q(h+1,v) = min{p(h+1,v-2), p(h-1,v)}
26:   else if p(h-1,v-2) <= min{p(h+1,v-2), p(h-1,v)}
27:     q(h+1,v) = max{p(h+1,v-2), p(h-1,v)}
28:   else
29:     q(h+1,v) = p(h-1,v) + p(h+1,v-2) - p(h-1,v-2)
30:   end if
31: end if
32: p(h+1,v) = ((q(h+1,v) >> shift0 + Diff1) << shift0) + (1 << (shift0 -1))
33: // For decoding the pixel p(h+2,v)
34: if (Line count < 4) // Vertical line counter, Boundary condition
35:   q(h+2,v) = p(h,v)
36: else if (unit count == 0) // unit count is for processing unit block, boundary condition
37:   q(h+2,v) = p(h+2,v-2)
38: else

```

```

39:     if  $p(h,v-2) \geq \max\{p(h+2,v-2), p(h,v)\}$            //Prediction value  $q$  of the preview pixel
40:          $q(h+2,v) = \min\{p(h+2,v-2), p(h,v)\}$ 
41:     else if  $p(h,v-2) \leq \min\{p(h+2,v-2), p(h,v)\}$ 
42:          $q(h+2,v) = \max\{p(h+2,v-2), p(h,v)\}$ 
43:     else
44:          $q(h+2,v) = p(h,v) + p(h+2,v-2) - p(h,v-2)$ 
45:     end if
46: end if
47:  $p(h+2,v) = ((q(h+2,v) >> shift0 + Diff2) << shift0) + (1 << (shift0 - 1))$ 
48: // For decoding the pixel  $p(h+3,v)$ 
49: if (Line count < 4)           // Vertical line counter, Boundary condition
50:      $q(h+3,v) = p(h+1,v)$ 
51: else if (unit count == 0)    // unit count is for processing unit block, boundary condition
52:      $q(h+3,v) = p(h+3,v-2)$ 
53: else
54:     if  $p(h+1,v-2) < \min\{p(h+3,v-2), p(h+1,v)\}$            //Prediction value  $q$  of the preview pixel
55:          $q(h+3,v) = \max\{p(h+3,v-2), p(h+1,v)\}$ 
56:     else if  $p(h+1,v-2) > \max\{p(h+3,v-2), p(h+1,v)\}$ 
57:          $q(h+3,v) = \min\{p(h+3,v-2), p(h+1,v)\}$ 
58:     else
59:          $q(h+3,v) = p(h+3,v-2) + p(h+1,v) - p(h+1,v-2)$ 
60:     end if
61: end if
62:  $p(h+3,v) = ((q(h+3,v) >> shift0 + Diff3) << shift0) + (1 << (shift0 - 1))$ 

```

I.1.2.2 Standard Mode 1: DGD (Diagonal Direction-based Differential Mode)

Compression Mode DGD references pixels that are located along diagonal directions. It gets the diagonal direction information using only Green pixels.

Figure 261 defines the packet structure for Compression Mode DGD (Bayer). For this Compression Mode, Header field **Prediction IDX** shall contain 0, and the high two bits of Header field **Mode IDX** shall contain 2'b10.

The Payload shall contain compressed data for four pixels:

- Field **Qstep** shall indicate the quantization step size.
- Fields **Diff0** through **Diff3** shall contain the encoded data, i.e., for each pixel, the difference between the predicted pixel and each original pixel.

Table 74 shows pseudo-code for decoding an MPC Packet for Compression Mode DGD (Bayer).

Mode	Header				Payload			
	Prediction IDX		Mode IDX		Diff0[3:0]	Diff1[3:0]	Diff2[3:0]	Diff3[3:0]
DGD	0	1	0	Qstep				

Figure 261 MPC Packet Structure for Compression Mode DGD (Bayer)

5365

Note:

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color. The Red- or Blue-first case is described in comments.

5369

Table 74 Decoder for Compression Mode DGD (Bayer)

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 0 : 1
3: // direction for p(h,v)
4: ldif0 = abs(p(h-2,v-2) - p(h-1,v-1)) // color pixel first: ldif0 = abs(p(h-1,v-2) - p(h,v-1))
5: ldif1 = abs(p(h,v-2) - p(h+1,v-1)) // color pixel first: ldif1 = abs(p(h+1,v-2) - p(h+2,v-1))
6: ldif2 = abs(p(h+2,v-2) - p(h+3,v-1)) // color pixel first: ldif2 = abs(p(h+3,v-2) - p(h+4,v-1))
7: rdif0 = abs(p(h,v-2) - p(h-1,v-1)) // color pixel first: rdif0 = abs(p(h-1,v-2) - p(h-2,v-1))
8: rdif1 = abs(p(h+2,v-2) - p(h+1,v-1)) // color pixel first: rdif1 = abs(p(h+1,v-2) - p(h,v-1))
9: rdif2 = abs(p(h+4,v-2) - p(h+3,v-1)) // color pixel first: rdif2 = abs(p(h+3,v-2) - p(h+2,v-1))
10: diff0 = rdif0 + rdif1
11: diff1 = ldif0 + ldif1
12: diff2 = abs(diff0 - diff1)
13: rup_diff = (diff2 <= 10) ? (diff0 + rdif2) : diff0
14: lup_diff = (diff2 <= 10) ? (diff1 + ldif2) : diff1
15: q(h,v) = (rup_diff < lup_diff) ? p(h+1,v-1) : p(h-1,v-1) // Prediction pixel
16: q(h+1,v) = (rup_diff < lup_diff) ? p(h+3,v-2) : p(h-1,v-2) // Prediction pixel
17: // color pixel first: q(h,v) = (rup_diff < lup_diff) ? p(h+2,v-2) : p(h-2,v-2)
18: // color pixel first: q(h+1,v) = (rup_diff < lup_diff) ? p(h+2,v-1) : p(h,v-1)
19: // direction for p(h+1,v)
20: ldif0 = abs(p(h,v-2) - p(h+1,v-1)) // color pixel first: ldif0 = abs(p(h-1,v-2) - p(h,v-1))
21: ldif1 = abs(p(h+2,v-2) - p(h+3,v-1)) // color pixel first: ldif1 = abs(p(h+1,v-2) - p(h+2,v-1))
22: ldif2 = abs(p(h+4,v-2) - p(h+5,v-1)) // color pixel first: ldif2 = abs(p(h+3,v-2) - p(h+4,v-1))
23: rdif0 = abs(p(h+2,v-2) - p(h+1,v-1)) // color pixel first: rdif0 = abs(p(h+3,v-2) - p(h+2,v-1))
24: rdif1 = abs(p(h+4,v-2) - p(h+3,v-1)) // color pixel first: rdif1 = abs(p(h+5,v-2) - p(h+4,v-1))
25: rdif2 = abs(p(h,v-2) - p(h-1,v-1)) // color pixel first: rdif2 = abs(p(h+1,v-2) - p(h,v-1))
26: diff0 = rdif0 + rdif1
27: diff1 = ldif0 + ldif1
28: diff2 = abs(diff0 - diff1)
29: rup_diff = (diff2 <= 10) ? (diff0 + rdif2) : diff0
30: lup_diff = (diff2 <= 10) ? (diff1 + ldif2) : diff1
31: q(h+2,v) = (rup_diff < lup_diff) ? p(h+3,v-1) : p(h+1,v-1) // Prediction pixel
32: q(h+3,v) = (rup_diff < lup_diff) ? p(h+5,v-2) : p(h+1,v-2) // Prediction pixel
33: // color pixel first: q(h+2,v) = (rup_diff < lup_diff) ? p(h+4,v-2) : p(h,v-2)
34: // color pixel first: q(h+3,v) = (rup_diff < lup_diff) ? p(h+4,v-1) : p(h+2,v-1)
35: // For decoding the pixel p(h+i,v), i={0,1,2,3}
36: p(h,v) = ((q(h,v) >> shift0 + Diff0) << shift0) + (1 << (shift0 -1))
37: p(h+1,v) = ((q(h+1,v) >> shift0 + Diff1) << shift0) + (1 << (shift0 -1))
38: p(h+2,v) = ((q(h+2,v) >> shift0 + Diff2) << shift0) + (1 << (shift0 -1))
39: p(h+3,v) = ((q(h+3,v) >> shift0 + Diff3) << shift0) + (1 << (shift0 -1))

```

I.1.2.3 Standard Mode 2: FNR (Fixed Quantization and No-Reference Mode)

If the difference between the current pixel and the reference pixel is large, then the pixel should be compressed by fixed quantization. In Compression Mode FNR, the data is compressed using just quantization; no difference operation is involved.

Figure 262 defines the packet structure for Compression Mode FNR (Bayer). For this Compression Mode, Header field **Prediction IDX** shall contain 0, and Header field **Mode IDX** shall contain 3'b110.

The Payload shall contain compressed data for four pixels:

- Fields **Pixel0** through **Pixel3** shall contain the quantization result for each pixel.

Table 75 shows pseudo-code for decoding an MPC Packet for Compression Mode FNR (Bayer).

Mode	Header				Payload			
	Prediction IDX		Mode IDX		Pixel0[3:0]	Pixel1[3:0]	Pixel2[3:0]	Pixel3[3:0]
FNR	0	1	1	0	Pixel0[3:0]	Pixel1[3:0]	Pixel2[3:0]	Pixel3[3:0]

Figure 262 MPC Packet Structure for Compression Mode FNR (Bayer)

Note:

In this pseudo-code, any color pixel can appear first.

Table 75 Decoder for Compression Mode FNR (Bayer)

```

1: // Quantization step
2: shift0 = 6
3: // For decoding the pixel p(h+i,v), i={0,1,2,3}
4: p(h,v) = (Pixel0<< shift0) + (1 << (shift0 -1))
5: p(h+1,v) = (Pixel1<< shift0) + (1 << (shift0 -1))
6: p(h+2,v) = (Pixel2<< shift0) + (1 << (shift0 -1))
7: p(h+3,v) = (Pixel3<< shift0) + (1 << (shift0 -1))

```

I.1.2.4 Standard Mode 3: OUT (OUTlier compensation Mode)

Compression Mode OUT performs compression when there is one outlier pixel in the current pixel $p(h+i, v)$. It can only handle a maximum of one outlier pixel out of the four consecutive pixels. The OUT packet informs the MPC Decoder which pixel (out of the four) is the outlier pixel.

Decoded pixels are usually used as reference pixels. However, the decoded pixels and reference pixels are different from each other for the outlier pixels in OUT mode (see **Figure 263**). Meanwhile, for the other three pixels, decoded pixels are the same as reference pixels. The MPC Decoder replaces the outlier pixel with the average of the candidates for prediction pixel (i.e., k_0) which is added to the threshold value to get the decoded pixel. The decoded pixel is expressed as $p(h+i, v)$, and the reference pixel is expressed as $p_q(h+i, v)$.

- k_0 shall contain the average of the candidates for prediction pixel which are not outlier pixels. It shall be a reference pixel of the outlier pixel, and enables the MPC Decoder to ignore the outlier pixels when the prediction pixels are received.

p_0, p_1, p_2 , and p_3 in **Figure 263** correspond to $p(h, v)$, $p(h+1, v)$, $p(h+2, v)$, and $p(h+3, v)$ (i.e., the “Pixels to be Encoded”) in the “Green Pixel First” or “Blue Pixel First” portion of **Figure 259**.

If p_0 or p_2 is the outlier pixel in **Figure 263**, then:

k_0 shall be the average of $p(h-2, v-2)$, $p(h, v-2)$, $p(h+2, v-2)$, and $p(h+4, v-2)$

Otherwise, if p_1 or p_3 is the outlier pixel, then:

k_0 shall be the average of $p(h-1, v-2)$, $p(h+1, v-2)$, $p(h+3, v-2)$, and $p(h+5, v-2)$

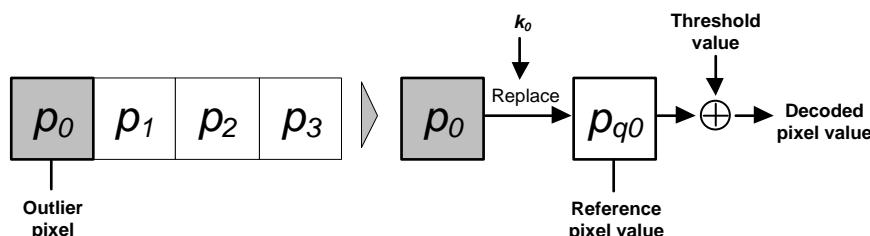


Figure 263 Decoded and Reference Pixel of Outlier Pixel (Bayer)

Figure 264 defines the packet structure for Compression Mode OUT (Bayer). For this Compression Mode, Header field **Prediction IDX** shall contain 0, and Header field **Mode IDX** shall contain 3'b111.

The Payload shall contain:

- The location of the outlier pixel in field **Pos**.
- The quantization step size in field **Qstep**.
- Encoded data for the three good pixels in fields **Diff1** through **Diff3** (i.e., for each good pixel, the difference between the predicted pixel and the original pixel).

Table 76 shows pseudo-code for decoding an MPC Packet for Compression Mode OUT (Bayer).

Mode	Header				Payload					
	Prediction IDX		Mode IDX		Pos[1:0]		Qstep[1:0]	Diff1[3:0]	Diff2[3:0]	Diff3[3:0]
OUT	0	1	1	1						

Figure 264 MPC Packet Structure for Compression Mode OUT (Bayer)

5409

Note:

5410

5411 This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the
 5412 processing sequence must be changed, but compression is performed in the same way for each
 5413 color.

Table 76 Decoder for Compression Mode OUT (Bayer)

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 1 : (Qstep==1) ? 2 : (Qstep==2) ? 3 : 4
3: // Prediction value
4: if (Pos == 0)
5:   q0 = p(h+1,v-2)           // color pixel first: q0 = (p(h,v-1) + p(h+2,v-1)) >> 1
6:   q1 = (p(h+1,v-1) + p(h+3,v-1)) >> 1    // color pixel first: q1 = p(h+2,v-2)
7:   q2 = p(h+3,v-2)           // color pixel first: q2 = (p(h+4,v-1) + p(h+2,v-1)) >> 1
8: else if (Pos == 1)
9:   q0 = (p(h-1,v-1) + p(h+1,v-1)) >> 1    // color pixel first: q0 = p(h,v-2)
10:  q1 = (p(h+1,v-1) + p(h+3,v-1)) >> 1    // color pixel first: q1 = p(h+2,v-2)
11:  q2 = p(h+3,v-2)           // color pixel first: q2 = (p(h+4,v-1) + p(h+2,v-1)) >> 1
12: else if (Pos == 2)
13:   q0 = (p(h-1,v-1) + p(h+1,v-1)) >> 1    // color pixel first: q0 = p(h,v-2)
14:   q1 = p(h+1,v-2)           // color pixel first: q1 = (p(h,v-1) + p(h+2,v-1)) >> 1
15:   q2 = p(h+3,v-2)           // color pixel first: q2 = (p(h+4,v-1) + p(h+2,v-1)) >> 1
16: else
17:   q0 = (p(h-1,v-1) + p(h+1,v-1)) >> 1    // color pixel first: q0 = p(h,v-2)
18:   q1 = p(h+1,v-2)           // color pixel first: q1 = (p(h,v-1) + p(h+2,v-1)) >> 1
19:   q2 = (p(h+1,v-1) + p(h+3,v-1)) >> 1    // color pixel first: q1 = p(h+2,v-2)
20: end if
21: k0 = (p(h-2,v-2) + p(h,v-2) + p(h+2,v-2) + p(h+4,v-2)) >> 2
22: // color pixel first: k0 = (p(h-1,v-2) + p(h+1,v-2) + p(h+3,v-2) + p(h+5,v-2)) >> 2
23: k1 = ((q0>> shift0 + Diff1) << shift0) + (1 << (shift0 - 1))
24: k2 = ((q1>> shift0 + Diff2) << shift0) + (1 << (shift0 - 1))
25: k3 = ((q2>> shift0 + Diff3) << shift0) + (1 << (shift0 - 1))
26: k`0 = (k0 + outlier_thresh) > 1023 ? 1023 : (k0 + outlier_thresh)
27: // outlier_thresh is user control parameter to detect outlier pixel with threshold value
28: // According to outlier pixel position Pos, decoded data mapping
29: // pq(h+i,v) is not output data but prediction pixel for using as reference pixel
30: pq(h,v) = (Pos == 0) ? k0 : k1
31: pq(h+1,v) = (Pos == 1) ? k0 : (Pos < 1) ? k1 : k2
32: pq(h+2,v) = (Pos == 2) ? k0 : (Pos < 2) ? k2 : k3
33: pq(h+3,v) = (Pos == 3) ? k0 : k3
34: // p(h+i,v) is output data, p and pq are different (basically these are same in other mode)
35: p(h,v) = (Pos == 0) ? k`0 : k1
36: p(h+1,v) = (Pos == 1) ? k`0 : (Pos < 1) ? k1 : k2
37: p(h+2,v) = (Pos == 2) ? k`0 : (Pos < 2) ? k2 : k3
38: p(h+3,v) = (Pos == 3) ? k`0 : k3

```

I.1.3 Advanced Prediction Mode (Bayer)

For Bayer image sensors, Advanced Prediction Mode:

- Compresses the image using more extensive direction-aware algorithms to minimize image quality loss.
- Supports Compression Modes ePD, eDGD, and eSHV as defined below.

I.1.3.1 Extended Mode 0: eDGD (extended DiaGonal Direction-based differential mode)

Compression Mode eDGD extends Compression Mode DGD by adding more quantization levels.

Figure 265 defines the packet structure for Compression Mode eDGD. For this Compression Mode, Header field **Prediction IDX** shall contain 1, and the high bit of Header field **Mode IDX** shall contain 0.

The Payload shall contain compressed data for four pixels:

- Field **Qstep** shall indicate the quantization step size.
- Fields **Diff0** through **Diff3** shall contain the encoded data (i.e., for each pixel, the difference between the predicted pixel and each original pixel).

Table 77 shows pseudo-code for decoding an MPC Packet for Compression Mode PD.

Mode	Header			Payload			
	Prediction IDX	Mode IDX		Diff0[3:0]	Diff1[3:0]	Diff2[3:0]	Diff3[3:0]
eDGD	1	0	Qstep[1:0]	Diff0[3:0]	Diff1[3:0]	Diff2[3:0]	Diff3[3:0]

Figure 265 MPC Packet Structure for Compression Mode eDGD

5427

Note:

5428

5429 This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the
 5430 processing sequence must be changed, but compression is performed in the same way for each
 color. The Red- or Blue-first case is described in comments.

Table 77 Decoder for Compression Mode eDGD

1:	// Quantization step
2:	shift0 = (<i>Qstep</i> ==0) ? 2 : (<i>Qstep</i> ==1) ? 3 : (<i>Qstep</i> ==2) ? 4 : 5
3:	// direction for <i>p(h,v)</i>
4:	<i>ldiff0</i> = abs(<i>p(h-2,v-2)</i> - <i>p(h-1,v-1)</i>) // color pixel first: <i>ldiff0</i> = abs(<i>p(h-1,v-2)</i> - <i>p(h,v-1)</i>)
5:	<i>ldiff1</i> = abs(<i>p(h,v-2)</i> - <i>p(h+1,v-1)</i>) // color pixel first: <i>ldiff1</i> = abs(<i>p(h+1,v-2)</i> - <i>p(h+2,v-1)</i>)
6:	<i>ldiff2</i> = abs(<i>p(h+2,v-2)</i> - <i>p(h+3,v-1)</i>) // color pixel first: <i>ldiff2</i> = abs(<i>p(h+3,v-2)</i> - <i>p(h+4,v-1)</i>)
7:	<i>rdiff0</i> = abs(<i>p(h,v-2)</i> - <i>p(h-1,v-1)</i>) // color pixel first: <i>rdiff0</i> = abs(<i>p(h-1,v-2)</i> - <i>p(h-2,v-1)</i>)
8:	<i>rdiff1</i> = abs(<i>p(h+2,v-2)</i> - <i>p(h+1,v-1)</i>) // color pixel first: <i>rdiff1</i> = abs(<i>p(h+1,v-2)</i> - <i>p(h,v-1)</i>)
9:	<i>rdiff2</i> = abs(<i>p(h+4,v-2)</i> - <i>p(h+3,v-1)</i>) // color pixel first: <i>rdiff2</i> = abs(<i>p(h+3,v-2)</i> - <i>p(h+2,v-1)</i>)
10:	<i>diff0</i> = <i>rdiff0</i> + <i>rdiff1</i>
11:	<i>diff1</i> = <i>ldiff0</i> + <i>ldiff1</i>
12:	<i>diff2</i> = abs(<i>diff0</i> - <i>diff1</i>)
13:	<i>rup_diff</i> = (<i>diff2</i> <= 10) ? (<i>diff0</i> + <i>rdiff2</i>) : <i>diff0</i>
14:	<i>lup_diff</i> = (<i>diff2</i> <= 10) ? (<i>diff1</i> + <i>ldiff2</i>) : <i>diff1</i>
15:	<i>q(h,v)</i> = (<i>rup_diff</i> < <i>lup_diff</i>) ? <i>p(h+1,v-1)</i> : <i>p(h-1,v-1)</i> // Prediction pixel
16:	<i>q(h+1,v)</i> = (<i>rup_diff</i> < <i>lup_diff</i>) ? <i>p(h+3,v-2)</i> : <i>p(h-1,v-2)</i> // Prediction pixel
17:	// color pixel first: <i>q(h,v)</i> = (<i>rup_diff</i> < <i>lup_diff</i>) ? <i>p(h+2,v-2)</i> : <i>p(h-2,v-2)</i>
18:	// color pixel first: <i>q(h+1,v)</i> = (<i>rup_diff</i> < <i>lup_diff</i>) ? <i>p(h+2,v-1)</i> : <i>p(h,v-1)</i>
19:	// direction for <i>p(h+1,v)</i>
20:	<i>ldiff0</i> = abs(<i>p(h,v-2)</i> - <i>p(h+1,v-1)</i>) // color pixel first: <i>ldiff0</i> = abs(<i>p(h-1,v-2)</i> - <i>p(h,v-1)</i>)
21:	<i>ldiff1</i> = abs(<i>p(h+2,v-2)</i> - <i>p(h+3,v-1)</i>) // color pixel first: <i>ldiff1</i> = abs(<i>p(h+1,v-2)</i> - <i>p(h+2,v-1)</i>)
22:	<i>ldiff2</i> = abs(<i>p(h+4,v-2)</i> - <i>p(h+5,v-1)</i>) // color pixel first: <i>ldiff2</i> = abs(<i>p(h+3,v-2)</i> - <i>p(h+4,v-1)</i>)
23:	<i>rdiff0</i> = abs(<i>p(h+2,v-2)</i> - <i>p(h+1,v-1)</i>) // color pixel first: <i>rdiff0</i> = abs(<i>p(h+3,v-2)</i> - <i>p(h+2,v-1)</i>)
24:	<i>rdiff1</i> = abs(<i>p(h+4,v-2)</i> - <i>p(h+3,v-1)</i>) // color pixel first: <i>rdiff1</i> = abs(<i>p(h+5,v-2)</i> - <i>p(h+4,v-1)</i>)
25:	<i>rdiff2</i> = abs(<i>p(h,v-2)</i> - <i>p(h-1,v-1)</i>) // color pixel first: <i>rdiff2</i> = abs(<i>p(h+1,v-2)</i> - <i>p(h,v-1)</i>)
26:	<i>diff0</i> = <i>rdiff0</i> + <i>rdiff1</i>
27:	<i>diff1</i> = <i>ldiff0</i> + <i>ldiff1</i>
28:	<i>diff2</i> = abs(<i>diff0</i> - <i>diff1</i>)
29:	<i>rup_diff</i> = (<i>diff2</i> <= 10) ? (<i>diff0</i> + <i>rdiff2</i>) : <i>diff0</i>
30:	<i>lup_diff</i> = (<i>diff2</i> <= 10) ? (<i>diff1</i> + <i>ldiff2</i>) : <i>diff1</i>
31:	<i>q(h+2,v)</i> = (<i>rup_diff</i> < <i>lup_diff</i>) ? <i>p(h+3,v-1)</i> : <i>p(h+1,v-1)</i> // Prediction pixel
32:	<i>q(h+3,v)</i> = (<i>rup_diff</i> < <i>lup_diff</i>) ? <i>p(h+5,v-2)</i> : <i>p(h+1,v-2)</i> // Prediction pixel
33:	// color pixel first: <i>q(h+2,v)</i> = (<i>rup_diff</i> < <i>lup_diff</i>) ? <i>p(h+4,v-2)</i> : <i>p(h,v-2)</i>
34:	// color pixel first: <i>q(h+3,v)</i> = (<i>rup_diff</i> < <i>lup_diff</i>) ? <i>p(h+4,v-1)</i> : <i>p(h+2,v-1)</i>
35:	// For decoding the pixel <i>p(h+i,v)</i> , <i>i</i> ={0,1,2,3}
36:	<i>p(h,v)</i> = ((<i>q(h,v)</i> >> <i>shift0</i> + <i>Diff0</i>) << <i>shift0</i>) + (1 << (<i>shift0</i> -1))
37:	<i>p(h+1,v)</i> = ((<i>q(h+1,v)</i> >> <i>shift0</i> + <i>Diff1</i>) << <i>shift0</i>) + (1 << (<i>shift0</i> -1))
38:	<i>p(h+2,v)</i> = ((<i>q(h+2,v)</i> >> <i>shift0</i> + <i>Diff2</i>) << <i>shift0</i>) + (1 << (<i>shift0</i> -1))
39:	<i>p(h+3,v)</i> = ((<i>q(h+3,v)</i> >> <i>shift0</i> + <i>Diff3</i>) << <i>shift0</i>) + (1 << (<i>shift0</i> -1))

I.1.3.2 Extended Mode 1: ePD (extended Pixel-based Directional Differential Mode)

Compression Mode ePD extends Compression Mode PD by adding more quantization levels.

Figure 266 defines the packet structure for Compression Mode ePD. For this Compression Mode, Header field **Prediction IDX** shall contain 1, and the two high bits of Header field **Mode IDX** shall contain 2'b10.

The Payload shall contain compressed data for four pixels:

- Field **Qstep** shall indicate the quantization step size.
- Fields **Diff0** through **Diff3** shall contain the encoded data (i.e., for each pixel, the difference between the predicted pixel and each original pixel).

Table 78 shows pseudo-code for decoding an MPC Packet for Compression Mode ePD.

Mode	Header			Payload			
	Prediction IDX	Mode IDX		Diff0[3:0]	Diff1[3:0]	Diff2[3:0]	Diff3[3:0]
ePD	1	1	0	Qstep	Diff0[3:0]	Diff1[3:0]	Diff2[3:0]

Figure 266 MPC Packet Structure for Compression Mode ePD

5440

Note:

5441

5442 This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the
 5443 processing sequence must be changed, but compression is performed in the same way for each
 color.

5444

Table 78 Decoder for Compression Mode ePD

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 4 : 5
3: // For decoding the pixel p(h,v)
4: if (Line count < 2)           // Vertical line counter, Boundary condition
5:     q(h,v) = p(h-2,v)
6: else if (unit count == 0)    // unit count is for processing unit block, boundary condition
7:     q(h,v) = p(h,v-2)
8: else                         //Prediction value q of the preview pixel
9:     if p(h-2,v-2) >= max{p(h,v-2), p(h-2,v)}
10:        q(h,v) = min{p(h,v-2), p(h-2,v)}
11:    else if p(h-2,v-2) <= min{p(h,v-2), p(h-2,v)}
12:        q(h,v) = max{p(h,v-2), p(h-2,v)}
13:    else
14:        q(h,v) = p(h-2,v) + p(h,v-2) - p(h-2,v-2)
15:    end if
16: end if
17: p(h,v) = ((q(h,v) >> shift0 + Diff0) << shift0) + (1 << (shift0 -1))
18: // For decoding the pixel p(h+1,v)
19: if (Line count < 2)           // Vertical line counter, Boundary condition
20:     q(h+1,v) = p(h-1,v)
21: else if (unit count == 0)    // unit count is for processing unit block, boundary condition
22:     q(h+1,v) = p(h+1,v-2)
23: else                         //Prediction value q of the preview pixel
24:     if p(h-1,v-2) >= max{p(h+1,v-2), p(h-1,v)}
25:         q(h+1,v) = min{p(h+1,v-2), p(h-1,v)}
26:     else if p(h-1,v-2) <= min{p(h+1,v-2), p(h-1,v)}
27:         q(h+1,v) = max{p(h+1,v-2), p(h-1,v)}
28:     else
29:         q(h+1,v) = p(h-1,v) + p(h+1,v-2) - p(h-1,v-2)
30:     end if
31: end if
32: p(h+1,v) = ((q(h+1,v) >> shift0 + Diff1) << shift0) + (1 << (shift0 -1))
33: // For decoding the pixel p(h+2,v)
34: if (Line count < 4)           // Vertical line counter, Boundary condition
35:     q(h+2,v) = p(h,v)
36: else if (unit count == 0)    // unit count is for processing unit block, boundary condition
37:     q(h+2,v) = p(h+2,v-2)
38: else

```

```

39:     if  $p(h,v-2) \geq \max\{p(h+2,v-2), p(h,v)\}$            //Prediction value  $q$  of the preview pixel
40:          $q(h+2,v) = \min\{p(h+2,v-2), p(h,v)\}$ 
41:     else if  $p(h,v-2) \leq \min\{p(h+2,v-2), p(h,v)\}$ 
42:          $q(h+2,v) = \max\{p(h+2,v-2), p(h,v)\}$ 
43:     else
44:          $q(h+2,v) = p(h,v) + p(h+2,v-2) - p(h,v-2)$ 
45:     end if
46: end if
47:  $p(h+2,v) = ((q(h+2,v) >> shift0 + Diff2) << shift0) + (1 << (shift0 - 1))$ 
48: // For decoding the pixel  $p(h+3,v)$ 
49: if (Line count < 4)           // Vertical line counter, Boundary condition
50:      $q(h+3,v) = p(h+1,v)$ 
51: else if (unit count == 0)    // unit count is for processing unit block, boundary condition
52:      $q(h+3,v) = p(h+3,v-2)$ 
53: else
54:     if  $p(h+1,v-2) < \min\{p(h+3,v-2), p(h+1,v)\}$            //Prediction value  $q$  of the preview pixel
55:          $q(h+3,v) = \max\{p(h+3,v-2), p(h+1,v)\}$ 
56:     else if  $p(h+1,v-2) > \max\{p(h+3,v-2), p(h+1,v)\}$ 
57:          $q(h+3,v) = \min\{p(h+3,v-2), p(h+1,v)\}$ 
58:     else
59:          $q(h+3,v) = p(h+3,v-2) + p(h+1,v) - p(h+1,v-2)$ 
60:     end if
61: end if
62:  $p(h+3,v) = ((q(h+3,v) >> shift0 + Diff3) << shift0) + (1 << (shift0 - 1))$ 

```

I.1.3.3 Extended Mode 2: eSHV (extended Slanted Horizontal or Vertical Direction-based Differential Mode)

5445 Compression Mode eSHV references pixels that are located along the slanted horizontal or vertical direction.

5446 **Figure 267** defines the packet structure for Compression Mode eSHV. For this Compression Mode, Header
5447 field **Prediction IDX** shall contain 1, and the high two bits of Header field **Mode IDX** shall contain 2'b11.

5448 The Payload shall contain compressed data for four pixels:

- 5449 • Field **Qstep** shall indicate the quantization step size.
- 5450 • Fields **Diff0** through **Diff3** shall contain the encoded data (i.e., for each pixel, the difference
5451 between the predicted pixel and each original pixel).

5452 **Table 79** shows pseudo-code for decoding an MPC Packet for Compression Mode eSHV.

Mode	Header				Payload			
	Prediction IDX		Mode IDX		Diff0[3:0]		Diff1[3:0]	Diff2[3:0]
eSHV	1	1	1	Qstep				

5453 **Figure 267 MPC Packet Structure for Compression Mode eSHV (Bayer)**

5454

Note:

5455

5456 This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the
 5457 processing sequence must be changed, but compression is performed in the same way for each
 color. The Red- or Blue-first case is described in comments.

5458

Table 79 Decoder for Compression Mode eSHV (Bayer)

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 2 : 3
3: // Prediction value q(h,v) from DGD mode for p(h+i,v), i={0,1}
4: if (q(h,v) == p(h-1,v-1)) // color pixel first: if (q(h+1,v) == p(h,v-1))
5:     left_en(h,v) = 1; left_en(h+1,v) = 1
6: else
7:     left_en(h,v) = 0; left_en(h+1,v) = 0
8: end if
9: offset(h,v) = left_en(h,v) ? 0 : (p(h,v-2) > p(h+1,v-1)) ? (p(h+1,v-1) - 40) : (p(h+1,v-1) + 40)
10: /* color pixel first: offset(h,v) = left_en(h,v) ? 0 :
11:    (p(h+1,v-2) > p(h+2,v-1)) ? (p(h+2,v-2) - 40) : (p(h+2,v-2) + 40) */
12: offset(h+1,v) = left_en(h+1,v) ? 0 : (p(h,v-2) > p(h+1,v-1)) ? (p(h+3,v-2) - 40) : (p(h+3,v-2) + 40)
13: /* color pixel first: offset(h+1,v) = left_en(h+1,v) ? 0 :
14:    (p(h+1,v-2) > p(h+2,v-1)) ? (p(h+2,v-1) - 40) : (p(h+2,v-1) + 40) */
15: offset(h,v) = (offset(h,v) > 1023) ? 0 : (offset(h,v) < 0) ? 0 : offset(h,v)
16: offset(h+1,v) = (offset(h+1,v) > 1023) ? 0 : (offset(h+1,v) < 0) ? 0 : offset(h+1,v)
17: // Prediction for high
18: q_h(h,v) = left_en(h,v) ? (p(h-1,v-1) + p(h,v-2)) >> 1 : (p(h,v-2) + p(h+1,v-1)) >> 1
19: // color pixel first: q_h(h,v) = left_en(h,v) ? (p(h-2,v-2) + p(h,v-2)) >> 1 : (p(h,v-2) + p(h+2,v-2)) >> 1
20: q_h(h+1,v) = left_en(h,v) ? (p(h-1,v-1) + p(h+1,v-2)) >> 1 : (p(h+1,v-2) + p(h+3,v-2)) >> 1
21: /* color pixel first: q_h(h+1,v) = left_en(h+1,v) ? (p(h,v-1) + p(h+1,v-2)) >> 1 :
22:    (p(h+1,v-2) + p(h+2,v-1)) >> 1 */
23: // Prediction for low
24: q_l(h,v) = left_en(h,v) ? (p(h-1,v-1) + p(h-2,v)) >> 1 : offset(h,v)
25: // color pixel first: q_l(h,v) = left_en(h,v) ? (p(h-2,v-2) + p(h-2,v)) >> 1 : offset(h,v)
26: q_l(h+1,v) = left_en(h,v) ? (p(h-1,v-2) + p(h-1,v)) >> 1 : offset(h+1,v)
27: // color pixel first: q_l(h+1,v) = left_en(h,v) ? (p(h-1,v-2) + p(h-1,v)) >> 1 : offset(h+1,v)
28: // Recon
29: low0 = left_en(h,v) ? p(h-2,v) : offset(h,v)
30: // color pixel first: low0 = left_en(h,v) ? p(h-1,v) : offset(h+1,v)
31: cent0 = left_en(h,v) ? p(h-1,v-1) : p(h+1,v-1)
32: // color pixel first: cent0 = left_en(h,v) ? p(h,v-1) : offset(h+2,v-1)
33: high0 = p(h,v-2) // color pixel first: high0 = p(h+1,v-2)
34: ldif0 = abs(cent0 - low0)
35: hdif0 = abs(cent0 - high0)
36: q(h,v) = (ldif0 < hdif0) ? q_l(h,v) : q_h(h,v)
37: q(h+1,v) = (ldif0 < hdif0) ? q_l(h+1,v) : q_h(h+1,v)
38: p(h,v) = ((q(h,v) >> shift0 + Diff0) << shift0) + (1 << (shift0 - 1))

```

```

39:     p(h+1,v) = ((q(h+1,v) >> shift0 + Diff1) << shift0) + (1 << (shift0 -1))
40:     // Prediction value q(h,v) from DGD mode for p(h+i,v), i={2,3}
41:     if (q(h+2,v) == p(h+1,v-1))           // color pixel first: if (q(h+3,v) == p(h+2,v-1))
42:         left_en(h+2,v) = 1; left_en(h+3,v) = 1
43:     else
44:         left_en(h+2,v) = 0; left_en(h+3,v) = 0
45:     end if
46:     offset(h+2,v) = left_en(h+2,v) ? 0 : (p(h+2,v-2) > p(h+3,v-1)) ? (p(h+3,v-1) - 40) : (p(h+3,v-1) + 40)
47:     /* color pixel first: offset(h+2,v) = left_en(h+2,v) ? 0 :
48:        (p(h+3,v-2) > p(h+4,v-1)) ? (p(h+4,v-2) - 40) : (p(h+4,v-2) + 40) */
49:     offset(h+3,v) = left_en(h+2,v) ? 0 : (p(h+2,v-2) > p(h+3,v-1)) ? (p(h+5,v-2) - 40) : (p(h+5,v-2) + 40)
50:     /* color pixel first: offset(h+3,v) = left_en(h+3,v) ? 0 :
51:        (p(h+3,v-2) > p(h+4,v-1)) ? (p(h+4,v-1) - 40) : (p(h+4,v-1) + 40) */
52:     offset(h+2,v) = (offset(h+2,v) > 1023) ? 0 : (offset(h+2,v) < 0) ? 0 : offset(h+2,v)
53:     offset(h+3,v) = (offset(h+3,v) > 1023) ? 0 : (offset(h+3,v) < 0) ? 0 : offset(h+3,v)
54:     // Prediction for high
55:     q_h(h+2,v) = left_en(h+2,v) ? (p(h+1,v-1) + p(h+2,v-2)) >> 1 : (p(h+2,v-2) + p(h+3,v-1)) >> 1
56:     /* color pixel first: q_h(h+2,v) = left_en(h+2,v) ? (p(h,v-2) + p(h+2,v-2)) >> 1 :
57:        (p(h+2,v-2) + p(h+4,v-2)) >> 1 */
58:     q_h(h+3,v) = left_en(h+2,v) ? (p(h+1,v-2) + p(h+3,v-2)) >> 1 : (p(h+3,v-2) + p(h+5,v-2)) >> 1
59:     /* color pixel first: q_h(h+3,v) = left_en(h+3,v) ? (p(h+2,v-1) + p(h+3,v-2)) >> 1 :
60:        (p(h+3,v-2) + p(h+4,v-1)) >> 1 */
61:     // Prediction for low
62:     q_l(h+2,v) = left_en(h+2,v) ? (p(h+1,v-1) + p(h,v)) >> 1 : offset(h+2,v)
63:     // color pixel first: q_l(h+2,v) = left_en(h+2,v) ? (p(h,v-2) + p(h,v)) >> 1 : offset(h+2,v)
64:     q_l(h+3,v) = left_en(h+2,v) ? (p(h+1,v-2) + p(h+1,v)) >> 1 : offset(h+3,v)
65:     // color pixel first: q_l(h+3,v) = left_en(h+3,v) ? (p(h+2,v-1) + p(h+1,v)) >> 1 : offset(h+3,v)
66:     // Recon
67:     low2 = left_en(h+2,v) ? p(h,v) : offset(h+2,v)
68:     // color pixel first: low2 = left_en(h+2,v) ? p(h+1,v) : offset(h+3,v)
69:     cent2 = left_en(h+2,v) ? p(h+1,v-1) : p(h+3,v-1)
70:     // color pixel first: cent2 = left_en(h+2,v) ? p(h+2,v-1) : p(h+4,v-1)
71:     high2 = p(h+2,v-2)                                // color pixel first: high2 = p(h+3,v-2)
72:     ldif2 = abs(cent2 - low2)
73:     hdif2 = abs(cent2 - high2)
74:     q(h+2,v) = (ldif2 < hdif2) ? q_l(h+2,v) : q_h(h+2,v)
75:     q(h+3,v) = (ldif2 < hdif2) ? q_l(h+3,v) : q_h(h+3,v)
76:     p(h+2,v) = ((q(h+2,v) >> shift0 + Diff2) << shift0) + (1 << (shift0 -1))
77:     p(h+3,v) = ((q(h+3,v) >> shift0 + Diff3) << shift0) + (1 << (shift0 -1))

```

I.2 MPC for Tetra-Cell Image Sensor

MPC for Tetra-Cell image sensors supports the eight Compression Modes that are defined in this Section and summarized in **Table 80**. If MPC is supported, then all eight MPC decoder Data Compression Modes shall be supported. Each Compression Mode uses either Basic Prediction Mode or Advanced Prediction Mode. **Figure 268** summarizes the MPC Packet Structures used for Tetra-Cell Compression Modes.

Table 80 MPC Compression Modes for Tetra-Cell Image Sensors

Compression Mode Name	Mode Description	Defined in Section
Basic Prediction Mode:		
AD	Average-based Directional Differential Mode	I.2.2.1
OD	Oblique Direction-based Differential Mode	I.2.2.1
FNR	Fixed Quantization and No-Reference Mode	I.2.2.1
OUT	OUTlier Compensation Mode	I.2.2.1
Advanced Prediction Mode:		
eMPD	extended Multi-Pixel-based Directional Differential Mode	I.2.3.1
eHVD	extended Horizontal or Vertical Direction-based Differential Mode	I.2.3.1
eHVA	extended Horizontal or Vertical Average-based Differential Mode	I.2.3.1
eOUT	extended OUTlier Compensation Mode	I.2.3.1

Mode Summary for Tetra-Cell Image Sensor								
Compression Mode	Header		Payload					
	Prediction IDX	Mode IDX						
<i>Basic Prediction Mode</i>								
AD	0	0	Qstep[1:0]	ByrDiff[3:0]	MltDiff0[2:0]	MltDiff1[2:0]	MltDiff2[2:0]	MltDiff3[2:0]
OD	0	1 0	Dir	Qstep[1:0]	ByrDiff[3:0]	MltDiff1[3:0]	MltDiff2[2:0]	MltDiff3[2:0]
FNR	0	1 1 0		ByrPixel[3:0]	MltPixel1[3:0]	MltPixel2[3:0]	MltPixel3[3:0]	
OUT	0	1 1 1	Pos[1:0]	Qstep[1:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[3:0]	
<i>Advanced Prediction Mode</i>								
eMPD	1	0	Qstep[1:0]	ByrDiff[3:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[3:0]	
eHVD	1	1 0	Dir	Qstep[1:0]	ByrDiff[3:0]	MltDiff1[3:0]	MltDiff2[2:0]	MltDiff3[2:0]
eHVA	1	1 1 0		ByrDiff[5:0]		MltDiff[5:0]	Dir	Slope Detail[1:0]
eOUT	1	1 1 1	Opt	ByrDiff[4:0]		MltDiff1[3:0]	MltDiff2[2:0]	MltDiff3[2:0]

Figure 268 MPC Packet Structures for Tetra-Cell Compression Modes

I.2.1 Reference Pixels for Prediction (Tetra-Cell)

MPC compresses each pixel by referring to neighboring pixels. The reference pixels are selected according to the Compression Mode and the color of the current pixel.

In a Tetra-Cell image, each cell consists of four Multi-Pixels in a 2x2 matrix.

A given pixel is expressed as $p_k(i, j)$, where:

- k : The Multi-Pixel index within the Tetra-Cell, i.e., 0, 1, 2, or 3 as shown in **Figure 269**:
- p_0 is the upper left Multi-Pixel
- p_1 is the upper right Multi-Pixel
- p_2 is the lower left Multi-Pixel
- p_3 is the lower right Multi-Pixel.

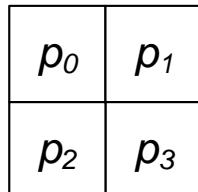


Figure 269 Tetra-Cell Multi-Pixel Indexing

- i : Horizontal index of the Tetra-Cell where the Multi-Pixel is located
- j : Vertical index of the Tetra-Cell where the Multi-Pixel is located

For a current Tetra-Cell $p(h, v)$, the coordinates are relative to a reference Tetra-Cell and are expressed as:

- The Horizontal relative index from the current Tetra-Cell to the reference Tetra-Cell,
 - Reference Tetra-Cells to the left of the current Tetra-Cell have negative h indexes
 - Reference Tetra-Cells directly above or below the current Tetra-Cell have zero h indexes
 - Reference Tetra-Cells to the right of the current Tetra-Cell have positive h indexes.

and

- The Vertical relative index from the current Tetra-Cell to the reference Tetra-Cell
 - Reference Tetra-Cells located above the current Tetra-Cell have negative v indexes
 - Reference Tetra-Cells in the same image line as the current cell have zero v indexes.

Examples:

- $p(h, v-1)$ Tetra-Cell immediately above the current Tetra-Cell
- $p(h-1, v)$ Tetra-Cell immediately to the left of the current Tetra-Cell
- $p(h+1, v-1)$ Tetra-Cell immediately above and immediately to the right of the current Tetra-Cell
- $p(h-1, v-1)$ Tetra-Cell immediately above and immediately to the left of the current Tetra-Cell

In the following sub-sections, all reference pixel coordinates are expressed using this notation.

5492
5493

Using this notation, the relative coordinates of reference pixels are as illustrated in **Figure 270**, where the color of each square indicates the color filter of the corresponding pixel.

Green Pixel First

	$p_0(h-2, v-2)$	$p_1(h-2, v-2)$	$p_0(h-1, v-2)$	$p_1(h-1, v-2)$	$p_0(h, v-2)$	$p_1(h, v-2)$	$p_0(h+1, v-2)$	$p_1(h+1, v-2)$	$p_0(h+2, v-2)$	$p_1(h+2, v-2)$	$p_0(h+3, v-2)$	$p_1(h+3, v-2)$			
	$p_2(h-2, v-2)$	$p_3(h-2, v-2)$	$p_2(h-1, v-2)$	$p_3(h-1, v-2)$	$p_2(h, v-2)$	$p_3(h, v-2)$	$p_2(h+1, v-2)$	$p_3(h+1, v-2)$	$p_2(h+2, v-2)$	$p_3(h+2, v-2)$	$p_2(h+3, v-2)$	$p_3(h+3, v-2)$			
	$p_0(h-2, v-1)$	$p_1(h-2, v-1)$	$p_0(h-1, v-1)$	$p_1(h-1, v-1)$	$p_0(h, v-1)$	$p_1(h, v-1)$	$p_0(h+1, v-1)$	$p_1(h+1, v-1)$	$p_0(h+2, v-1)$	$p_1(h+2, v-1)$	$p_0(h+3, v-1)$	$p_1(h+3, v-1)$			
	$p_2(h-2, v-1)$	$p_3(h-2, v-1)$	$p_2(h-1, v-1)$	$p_3(h-1, v-1)$	$p_2(h, v-1)$	$p_3(h, v-1)$	$p_2(h+1, v-1)$	$p_3(h+1, v-1)$	$p_2(h+2, v-1)$	$p_3(h+2, v-1)$	$p_2(h+3, v-1)$	$p_3(h+3, v-1)$			
	$p_0(h-2, v)$	$p_1(h-2, v)$	$p_0(h-1, v)$	$p_1(h-1, v)$	$p_0(h, v)$	$p_1(h, v)$	$p_0(h+1, v)$	$p_1(h+1, v)$							
	$p_2(h-2, v)$	$p_3(h-2, v)$	$p_2(h-1, v)$	$p_3(h-1, v)$	$p_2(h, v)$	$p_3(h, v)$	$p_2(h+1, v)$	$p_3(h+1, v)$							

Pixels to be Encoded

Blue Pixel First

	$p_0(h-2, v-2)$	$p_1(h-2, v-2)$	$p_0(h-1, v-2)$	$p_1(h-1, v-2)$	$p_0(h, v-2)$	$p_1(h, v-2)$	$p_0(h+1, v-2)$	$p_1(h+1, v-2)$	$p_0(h+2, v-2)$	$p_1(h+2, v-2)$	$p_0(h+3, v-2)$	$p_1(h+3, v-2)$			
	$p_2(h-2, v-2)$	$p_3(h-2, v-2)$	$p_2(h-1, v-2)$	$p_3(h-1, v-2)$	$p_2(h, v-2)$	$p_3(h, v-2)$	$p_2(h+1, v-2)$	$p_3(h+1, v-2)$	$p_2(h+2, v-2)$	$p_3(h+2, v-2)$	$p_2(h+3, v-2)$	$p_3(h+3, v-2)$			
	$p_0(h-2, v-1)$	$p_1(h-2, v-1)$	$p_0(h-1, v-1)$	$p_1(h-1, v-1)$	$p_0(h, v-1)$	$p_1(h, v-1)$	$p_0(h+1, v-1)$	$p_1(h+1, v-1)$	$p_0(h+2, v-1)$	$p_1(h+2, v-1)$	$p_0(h+3, v-1)$	$p_1(h+3, v-1)$			
	$p_2(h-2, v-1)$	$p_3(h-2, v-1)$	$p_2(h-1, v-1)$	$p_3(h-1, v-1)$	$p_2(h, v-1)$	$p_3(h, v-1)$	$p_2(h+1, v-1)$	$p_3(h+1, v-1)$	$p_2(h+2, v-1)$	$p_3(h+2, v-1)$	$p_2(h+3, v-1)$	$p_3(h+3, v-1)$			
	$p_0(h-2, v)$	$p_1(h-2, v)$	$p_0(h-1, v)$	$p_1(h-1, v)$	$p_0(h, v)$	$p_1(h, v)$	$p_0(h+1, v)$	$p_1(h+1, v)$							
	$p_2(h-2, v)$	$p_3(h-2, v)$	$p_2(h-1, v)$	$p_3(h-1, v)$	$p_2(h, v)$	$p_3(h, v)$	$p_2(h+1, v)$	$p_3(h+1, v)$							

Pixels to be Encoded

5494

Not Shown: Red Pixel First**Figure 270 Reference Pixel Relative Coordinate Indexes (Tetra-Cell)**5495
5496
5497

For a Tetra-Cell image sensor, the eight highlighted pixels in the bottom two rows will be encoded. This corresponds to Tetra-Cell coordinates $p(h, v)$ (i.e., the current Tetra-Cell) and $p(h+1, v)$. The other colored squares represent the previously decoded pixels, which are available to be used as reference data.

I.2.1.1 User Control Parameters for Boundary Fill

At the boundary of the image frame, no neighboring reference pixels exist. As a result, difference calculations for pixels at the image boundary cannot depend upon the values of neighbor pixels. To handle this case, MPC allows the boundary to be filled with user control parameters. This feature makes prediction possible at the image boundary despite the absence of actual prediction pixels. The user control parameter should be calculated before the start of a new image frame by using sensor information such as the luminance value of the image, or the gain value of the sensor. For example, the user control parameter can be set to the value 64 in low-light conditions or the value 128 for normal conditions.

I.2.2 Basic Prediction Mode (Tetra-Cell)

For Tetra-Cell image sensors, Basic Prediction Mode:

- Compresses the image using a simple algorithm to handle image details oriented in a specific direction, or basic cases of differences between neighboring pixels.
- Supports Compression Modes AD, OD, FNR, and OUT as defined below.

I.2.2.1 Standard Mode 0: AD (Average-based Directional Differential Mode)

Compression Mode AD is efficient for compressing pixels located on horizontal and vertical edges. An AD packet transmits one prediction pixel and three reference pixels. The prediction pixel is either the combination of the three reference pixels, or the reference pixel itself.

Figure 271 defines the packet structure for Compression Mode AD. For this Compression Mode, Header field **Prediction IDX** shall contain 0, and the high bit of Header field **Mode IDX** shall contain 0.

The Payload shall contain compressed data for the four pixels of a Tetra-Cell:

- Field **Qstep** shall indicate the quantization step size.
- Field **ByrDiff** shall contain the quantized difference between a prediction and the average of the four pixels.

Field **ByrDiff** is used both for decoding fields **MltDiff0** through **MltDiff3**, and for generating the Bayer image (see *Section I.3.4*).

- Fields **MltDiff0** through **MltDiff3** shall contain the encoded data, i.e., for each pixel, the encoded data between the averaged pixel and each Multi-Pixel.

Table 81 shows pseudo-code for decoding an MPC Packet for Compression Mode AD.

Mode	Header		Payload					
	Prediction IDX	Mode IDX						
AD	0	0	Qstep[1:0]	ByrDiff[3:0]	MltDiff0[2:0]	MltDiff1[2:0]	MltDiff2[2:0]	MltDiff3[2:0]

Figure 271 MPC Packet Structure for Compression Mode AD

5524

Note:

5525

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color.

5526

5527

5528

5529

Note also that this pseudo-code only describes 2x2 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

5530

Table 81 Decoder for Compression Mode AD

```

1:   // Quantization step
2:   shift0 = (Qstep==0) ? 1 : (Qstep==1) ? 2 : (Qstep==2) ? 3 : 5
3:   shift1 = (Qstep==0) ? 0 : (Qstep==1) ? 1 : (Qstep==2) ? 2 : 3
4:   // For decoding the pixel  $p_i(h, v)$ ,  $i=\{0,1,2,3\}$ 
5:   if (Line count < 4)           // Vertical line counter, Boundary condition
6:       q =  $p_1(h-2, v)$ 
7:   else if (unit count == 0)     // unit count is for processing unit block, boundary condition
8:       q =  $p_2(h, v-2)$ 
9:   else                         // Prediction value q of the preview pixel
10:      if  $p_3(h-2, v-2) \geq \max\{p_2(h, v-2), p_1(h-2, v)\}$ 
11:          q =  $\min\{p_2(h, v-2), p_1(h-2, v)\}$ 
12:      else if  $p_3(h-2, v-2) \leq \min\{p_2(h, v-2), p_1(h-2, v)\}$ 
13:          q =  $\max\{p_2(h, v-2), p_1(h-2, v)\}$ 
14:      else
15:          q =  $p_2(h, v-2) + p_1(h-2, v) - p_3(h-2, v-2)$ 
16:      end if
17:  end if
18:   $p_{byr}(h, v) = q >> shift0 + ByrDiff$            // Bayer pixel  $p_{byr}$ 
19:   $p_0(h, v) = ((p_{byr} >> shift1 + MltDiff0) << shift1) + (1 << (shift1-1))$ 
20:   $p_1(h, v) = ((p_{byr} >> shift1 + MltDiff1) << shift1) + (1 << (shift1-1))$ 
21:   $p_2(h, v) = ((p_{byr} >> shift1 + MltDiff2) << shift1) + (1 << (shift1-1))$ 
22:   $p_3(h, v) = ((p_{byr} >> shift1 + MltDiff3) << shift1) + (1 << (shift1-1))$ 

```

I.2.2.2 Standard Mode 1: OD (Oblique Direction-based Differential Mode)

5531 Compression Mode OD references pixels that are located along oblique directions.

5532 *Figure 272* defines the packet structure for Compression Mode OD. For this Compression Mode, Header
5533 field **Prediction IDX** shall contain 0, and the high two bits of Header field **Mode IDX** shall contain 2'b10.

5534 The Payload shall contain compressed data for the four pixels of a Tetra-Cell:

- 5535 • 1-bit field **Dir** shall indicate the reference direction:

5536 **If Dir contains 0** then the reference direction is the Slash Direction, i.e., rising to the right
5537 like a slash character: /

5538 **If Dir contains 1** then the reference direction is the Backslash Direction, i.e., rising to the
5539 left like a backslash character: \

- 5540 • Field **Qstep** shall indicate the quantization step size.
5541 • Fields **ByrDiff** and **MltDiff0** through **MltDiff3** shall contain the encoded data, i.e., for each pixel, the
5542 difference between the predicted pixel and each original pixel.

5543 Field **ByrDiff** can also be regarded as **MltDiff0**; it is used for generation of the Bayer
5544 image, see *Section I.3.4*.

5545 *Table 82* shows pseudo-code for decoding an MPC Packet for Compression Mode OD.

Mode	Header				Payload				
	Prediction IDX		Mode IDX		Qstep[1:0]	ByrDiff[3:0]	MltDiff1[3:0]	MltDiff2[2:0]	MltDiff3[2:0]
OD	0	1	0	Dir	Qstep[1:0]	ByrDiff[3:0]	MltDiff1[3:0]	MltDiff2[2:0]	MltDiff3[2:0]

5546 **Figure 272 MPC Packet Structure for Compression Mode OD**

5547

Note:

5548

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color. The Red- or Blue-first case is described in comments.

5549

Note also that this pseudo-code describes each case of 2x2 Green $p_k(h, v)$ and 2x2 Red/Blue $p_k(h+1, v)$ pixels.

5550

5551

5552

Table 82 Decoder for Compression Mode OD

```

1:   **** For decoding the first 2x2 pixel  $p_k(h, v)$ ,  $k=\{0,1,2,3\}$  ****
2:   // Quantization step
3:   shift0 = ( $Qstep == 0$ ) ? 2 : ( $Qstep == 1$ ) ? 3 : ( $Qstep == 2$ ) ? 4 : 5
4:   // Extracting prediction mode
5:   th = 50
6:   if Dir == 1    // Back-slash direction
7:     if abs(abs( $p_2(h-1, v-1)$ ) -  $p_3(h-1, v-1)$ ) - abs( $p_1(h-1, v-1)$  -  $p_3(h-1, v-1)$ ) >= th
8:       // color pixel first: abs(abs( $p_2(h-2, v-2)$ ) -  $p_3(h-2, v-2)$ ) - abs( $p_1(h-2, v-2)$  -  $p_3(h-2, v-2)$ ) >= th
9:         if abs( $p_2(h-1, v-1)$  -  $p_3(h-1, v-1)$ ) >= (abs( $p_1(h-1, v-1)$  -  $p_3(h-1, v-1)$ )*2)
10:        // color pixel first: abs( $p_2(h-2, v-2)$  -  $p_3(h-2, v-2)$ ) >= (abs( $p_1(h-2, v-2)$  -  $p_3(h-2, v-2)$ )*2)
11:        mode = 3
12:      else if abs( $p_1(h-1, v-1)$  -  $p_3(h-1, v-1)$ ) >= (abs( $p_2(h-1, v-1)$  -  $p_3(h-1, v-1)$ )*2)
13:        // color pixel first: abs( $p_1(h-2, v-2)$  -  $p_3(h-2, v-2)$ ) >= (abs( $p_2(h-2, v-2)$  -  $p_3(h-2, v-2)$ )*2)
14:        mode = 4
15:      else if abs( $p_2(h-1, v-1)$  -  $p_3(h-1, v-1)$ ) >= abs( $p_1(h-1, v-1)$  -  $p_3(h-1, v-1)$ )
16:        // color pixel first: abs( $p_2(h-2, v-2)$  -  $p_3(h-2, v-2)$ ) >= abs( $p_1(h-2, v-2)$  -  $p_3(h-2, v-2)$ )
17:        mode = 1
18:      else
19:        mode = 2
20:      end if
21:    else
22:      mode = 0
23:    end if
24:  else      // Slash direction
25:    if abs(abs( $p_2(h+1, v-1)$ ) -  $p_3(h+1, v-1)$ ) - abs( $p_0(h+1, v-1)$  -  $p_2(h+1, v-1)$ ) >= th
26:      // color pixel first: abs(abs( $p_2(h+2, v-2)$ ) -  $p_3(h+2, v-2)$ ) - abs( $p_0(h+2, v-2)$  -  $p_2(h+2, v-2)$ ) >= th
27:        if abs( $p_2(h+1, v-1)$  -  $p_3(h+1, v-1)$ ) >= (abs( $p_0(h+1, v-1)$  -  $p_2(h+1, v-1)$ )*2)
28:          // color pixel first: abs( $p_2(h+2, v-2)$  -  $p_3(h+2, v-2)$ ) >= (abs( $p_0(h+2, v-2)$  -  $p_2(h+2, v-2)$ )*2)
29:          mode = 8
30:        else if abs( $p_0(h+1, v-1)$  -  $p_2(h+1, v-1)$ ) >= (abs( $p_2(h+1, v-1)$  -  $p_3(h+1, v-1)$ )*2)
31:          // color pixel first: abs( $p_0(h+2, v-2)$  -  $p_2(h+2, v-2)$ ) >= (abs( $p_2(h+2, v-2)$  -  $p_3(h+2, v-2)$ )*2)
32:          mode = 9
33:        else if abs( $p_2(h+1, v-1)$  -  $p_3(h+1, v-1)$ ) >= abs( $p_0(h+1, v-1)$  -  $p_2(h+1, v-1)$ )
34:          // color pixel first: abs( $p_2(h+2, v-2)$  -  $p_3(h+2, v-2)$ ) >= abs( $p_0(h+2, v-2)$  -  $p_2(h+2, v-2)$ )
35:          mode = 6
36:        else

```

```

37:           mode = 7
38:       end if
39:   else
40:       mode = 5
41:   end if
42: end if
43: // Decoding process
44: if mode == 0
45:     q0 = p3(h-1, v-1)          // color pixel first: q0 = p3(h-2, v-2)
46:     q1 = p1(h-1, v-1)          // color pixel first: q1 = p1(h-2, v-2)
47:     q2 = p2(h-1, v-1)          // color pixel first: q2 = p2(h-2, v-2)
48:     p0(h,v) = ((q0>>shift0 + ByrDiff) << shift0) + (1 << (shift0-1))
49:     p1(h,v) = ((q1>>shift0 + MltDiff1) << shift0) + (1 << (shift0-1))
50:     p2(h,v) = ((q2>>shift0 + MltDiff2) << shift0) + (1 << (shift0-1))
51:     p3(h,v) = ((p0(h,v)>>shift0 + MltDiff3) << shift0) + (1 << (shift0-1))
52: else if mode == 1
53:     q0 = p3(h-1, v-1)          // color pixel first: q0 = p3(h-2, v-2)
54:     q1 = p1(h-1, v-1)          // color pixel first: q1 = p1(h-2, v-2)
55:     q2 = (p2(h-1, v-1) + p3(h-1, v-1)) >> 1    // color pixel first: q2 = (p2(h-2, v-2) + p3(h-2, v-2)) >> 1
56:     p0(h,v) = ((q0>>shift0 + ByrDiff) << shift0) + (1 << (shift0-1))
57:     p1(h,v) = ((q1>>shift0 + MltDiff1) << shift0) + (1 << (shift0-1))
58:     p2(h,v) = ((q2>>shift0 + MltDiff2) << shift0) + (1 << (shift0-1))
59:     p3(h,v) = ((p0(h,v)>>shift0 + MltDiff3) << shift0) + (1 << (shift0-1))
60: else if mode == 2
61:     q0 = p3(h-1, v-1)          // color pixel first: q0 = p3(h-2, v-2)
62:     q1 = (p1(h-1, v-1) + p3(h-1, v-1)) >> 1    // color pixel first: q1 = (p1(h-2, v-2) + p3(h-2, v-2)) >> 1
63:     q2 = p2(h-1, v-1)          // color pixel first: q2 = p2(h-2, v-2)
64:     p0(h,v) = ((q0>>shift0 + ByrDiff) << shift0) + (1 << (shift0-1))
65:     p1(h,v) = ((q1>>shift0 + MltDiff1) << shift0) + (1 << (shift0-1))
66:     p2(h,v) = ((q2>>shift0 + MltDiff3) << shift0) + (1 << (shift0-1))
67:     p3(h,v) = ((p0(h,v)>>shift0 + MltDiff2) << shift0) + (1 << (shift0-1))
68: else if mode == 3
69:     q0 = p1(h-1, v-1)          // color pixel first: q0 = p1(h-2, v-2)
70:     q1 = (p2(h, v-2) + p1(h-1, v-1)) >> 1    // color pixel first: q1 = p1(h-2, v-2)
71:     q2 = p0(h-1, v-1)          // color pixel first: q2 = p3(h-2, v-2)
72:     p0(h,v) = ((q0>>shift0 + ByrDiff) << shift0) + (1 << (shift0-1))
73:     p1(h,v) = ((q1>>shift0 + MltDiff1) << shift0) + (1 << (shift0-1))
74:     p2(h,v) = ((q2>>shift0 + MltDiff3) << shift0) + (1 << (shift0-1))
75:     p3(h,v) = ((p0(h,v)>>shift0 + MltDiff2) << shift0) + (1 << (shift0-1))
76: else if mode == 4
77:     q0 = p2(h-1, v-1)          // color pixel first: q0 = p2(h-2, v-2)
78:     q1 = p3(h-1, v-1)          // color pixel first: q1 = p3(h-2, v-2)
79:     q2 = p2(h-1, v-1)          // color pixel first: q2 = p2(h-2, v-2)

```

```

80:    $p_0(h,v) = ((q_0 >> shift0 + \text{ByrDiff}) << shift0) + (1 << (shift0-1))$ 
81:    $p_1(h,v) = ((q_1 >> shift0 + \text{MltDiff1}) << shift0) + (1 << (shift0-1))$ 
82:    $p_2(h,v) = ((q_2 >> shift0 + \text{MltDiff2}) << shift0) + (1 << (shift0-1))$ 
83:    $p_3(h,v) = ((p_0(h,v) >> shift0 + \text{MltDiff3}) << shift0) + (1 << (shift0-1))$ 
84: else if mode == 5
85:    $q_0 = p_0(h+1, v-1)$            // color pixel first:  $q_0 = p_0(h+2, v-2)$ 
86:    $q_1 = p_2(h+1, v-1)$            // color pixel first:  $q_1 = p_2(h+2, v-2)$ 
87:    $q_3 = p_3(h+1, v-1)$            // color pixel first:  $q_3 = p_3(h+2, v-2)$ 
88:    $p_0(h,v) = ((q_0 >> shift0 + \text{ByrDiff}) << shift0) + (1 << (shift0-1))$ 
89:    $p_1(h,v) = ((q_1 >> shift0 + \text{MltDiff1}) << shift0) + (1 << (shift0-1))$ 
90:    $p_3(h,v) = ((q_3 >> shift0 + \text{MltDiff3}) << shift0) + (1 << (shift0-1))$ 
91:    $p_2(h,v) = ((p_1(h,v) >> shift0 + \text{MltDiff2}) << shift0) + (1 << (shift0-1))$ 
92: else if mode == 6
93:    $q_0 = p_0(h+1, v-1)$            // color pixel first:  $q_0 = p_0(h+2, v-2)$ 
94:    $q_1 = p_2(h+1, v-1)$            // color pixel first:  $q_1 = p_2(h+2, v-2)$ 
95:    $q_3 = (p_2(h+1, v-1) + p_3(h+1, v-1)) >> 1$  // color pixel first:  $q_3 = (p_2(h+2, v-2) + p_3(h+2, v-2)) >> 1$ 
96:    $p_0(h,v) = ((q_0 >> shift0 + \text{ByrDiff}) << shift0) + (1 << (shift0-1))$ 
97:    $p_1(h,v) = ((q_1 >> shift0 + \text{MltDiff1}) << shift0) + (1 << (shift0-1))$ 
98:    $p_3(h,v) = ((q_3 >> shift0 + \text{MltDiff3}) << shift0) + (1 << (shift0-1))$ 
99:    $p_2(h,v) = ((p_1(h,v) >> shift0 + \text{MltDiff2}) << shift0) + (1 << (shift0-1))$ 
100: else if mode == 7
101:   $q_0 = (p_0(h+1, v-1) + p_2(h+1, v-1)) >> 1$  // color pixel first:  $q_0 = (p_0(h+2, v-2) + p_2(h+2, v-2)) >> 1$ 
102:   $q_1 = p_2(h+1, v-1)$            // color pixel first:  $q_1 = p_2(h+2, v-2)$ 
103:   $q_3 = p_3(h+1, v-1)$            // color pixel first:  $q_3 = p_3(h+2, v-2)$ 
104:   $p_0(h,v) = ((q_0 >> shift0 + \text{ByrDiff}) << shift0) + (1 << (shift0-1))$ 
105:   $p_1(h,v) = ((q_1 >> shift0 + \text{MltDiff1}) << shift0) + (1 << (shift0-1))$ 
106:   $p_3(h,v) = ((q_3 >> shift0 + \text{MltDiff3}) << shift0) + (1 << (shift0-1))$ 
107:   $p_2(h,v) = ((p_1(h,v) >> shift0 + \text{MltDiff2}) << shift0) + (1 << (shift0-1))$ 
108: else if mode == 8
109:   $q_0 = (p_3(h, v-2) + p_0(h+1, v-1)) >> 1$  // color pixel first:  $q_0 = p_0(h+2, v-2)$ 
110:   $q_1 = p_0(h+1, v-1)$            // color pixel first:  $q_1 = p_0(h+2, v-2)$ 
111:   $q_3 = p_1(h+1, v-1)$            // color pixel first:  $q_3 = p_2(h+2, v-2)$ 
112:   $p_0(h,v) = ((q_0 >> shift0 + \text{ByrDiff}) << shift0) + (1 << (shift0-1))$ 
113:   $p_1(h,v) = ((q_1 >> shift0 + \text{MltDiff1}) << shift0) + (1 << (shift0-1))$ 
114:   $p_3(h,v) = ((q_3 >> shift0 + \text{MltDiff3}) << shift0) + (1 << (shift0-1))$ 
115:   $p_2(h,v) = ((p_1(h,v) >> shift0 + \text{MltDiff2}) << shift0) + (1 << (shift0-1))$ 
116: else if mode == 9
117:   $q_0 = p_2(h+1, v-1)$            // color pixel first:  $q_0 = p_2(h+2, v-2)$ 
118:   $q_1 = p_3(h+1, v-1)$            // color pixel first:  $q_1 = p_3(h+2, v-2)$ 
119:   $q_3 = p_2(h+1, v-1)$            // color pixel first:  $q_3 = p_3(h+2, v-2)$ 
120:   $p_0(h,v) = ((q_0 >> shift0 + \text{ByrDiff}) << shift0) + (1 << (shift0-1))$ 
121:   $p_1(h,v) = ((q_1 >> shift0 + \text{MltDiff1}) << shift0) + (1 << (shift0-1))$ 
122:   $p_3(h,v) = ((q_3 >> shift0 + \text{MltDiff3}) << shift0) + (1 << (shift0-1))$ 

```

```

123:      $p_2(h,v) = ((p_1(h,v) >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
124:   end if
125:   ***** For decoding the second 2x2 pixel  $p_k(h+I,v)$ ,  $k=\{0,1,2,3\}$ *****
126:   // Quantization step
127:    $shift0 = (Qstep == 0) ? 2 : (Qstep == 1) ? 3 : (Qstep == 2) ? 4 : 5$ 
128:   // Extracting prediction mode
129:    $th = 50$ 
130:   if Dir == 1 // Back-slash direction
131:     if  $abs(abs(p_2(h-I,v-2) - p_3(h-I,v-2)) - abs(p_1(h-I,v-2) - p_3(h-I,v-2))) >= th$ 
132:       // color pixel first:  $abs(abs(p_2(h,v-1) - p_3(h,v-1)) - abs(p_1(h,v-1) - p_3(h,v-1))) >= th$ 
133:       if  $abs(p_2(h-I,v-2) - p_3(h-I,v-2)) >= (abs(p_1(h-I,v-2) - p_3(h-I,v-2))*2)$ 
134:         // color pixel first:  $abs(p_2(h,v-1) - p_3(h,v-1)) >= (abs(p_1(h,v-1) - p_3(h,v-1))*2)$ 
135:          $mode = 3$ 
136:       else if  $abs(p_1(h-I,v-2) - p_3(h-I,v-2)) >= (abs(p_2(h-I,v-2) - p_3(h-I,v-2))*2)$ 
137:         // color pixel first:  $abs(p_1(h,v-1) - p_3(h,v-1)) >= (abs(p_2(h,v-1) - p_3(h,v-1))*2)$ 
138:          $mode = 4$ 
139:       else if  $abs(p_2(h-I,v-2) - p_3(h-I,v-2)) >= abs(p_1(h-I,v-2) - p_3(h-I,v-2))$ 
140:         // color pixel first:  $abs(p_2(h,v-1) - p_3(h,v-1)) >= abs(p_1(h,v-1) - p_3(h,v-1))$ 
141:          $mode = 1$ 
142:       else
143:          $mode = 2$ 
144:       end if
145:     else
146:        $mode = 0$ 
147:     end if
148:   else // Slash direction
149:     if  $abs(abs(p_2(h+3,v-2) - p_3(h+3,v-2)) - abs(p_0(h+3,v-2) - p_2(h+3,v-2))) >= th$ 
150:       // color pixel first:  $abs(abs(p_2(h+2,v-1) - p_3(h+2,v-1)) - abs(p_0(h+2,v-1) - p_2(h+2,v-1))) >= th$ 
151:       if  $abs(p_2(h+3,v-2) - p_3(h+3,v-2)) >= (abs(p_0(h+3,v-2) - p_2(h+3,v-2))*2)$ 
152:         // color pixel first:  $abs(p_2(h+2,v-1) - p_3(h+2,v-1)) >= (abs(p_0(h+2,v-1) - p_2(h+2,v-1))*2)$ 
153:          $mode = 8$ 
154:       else if  $abs(p_0(h+3,v-2) - p_2(h+3,v-2)) >= (abs(p_2(h+3,v-2) - p_3(h+3,v-2))*2)$ 
155:         // color pixel first:  $abs(p_0(h+2,v-1) - p_2(h+2,v-1)) >= (abs(p_2(h+2,v-1) - p_3(h+2,v-1))*2)$ 
156:          $mode = 9$ 
157:       else if  $abs(p_2(h+3,v-2) - p_3(h+3,v-2)) >= abs(p_0(h+3,v-2) - p_2(h+3,v-2))$ 
158:         // color pixel first:  $abs(p_2(h+2,v-1) - p_3(h+2,v-1)) >= abs(p_0(h+2,v-1) - p_2(h+2,v-1))$ 
159:          $mode = 6$ 
160:       else
161:          $mode = 7$ 
162:       end if
163:     else
164:        $mode = 5$ 
165:     end if

```

```

166:    end if
167:    // Decoding process
168:    if mode == 0
169:         $q_0 = p_3(h-1, v-2)$           // color pixel first:  $q_0 = p_3(h, v-1)$ 
170:         $q_1 = p_1(h-1, v-2)$           // color pixel first:  $q_1 = p_1(h, v-1)$ 
171:         $q_2 = p_2(h-1, v-2)$           // color pixel first:  $q_2 = p_2(h, v-1)$ 
172:         $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
173:         $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
174:         $p_2(h+1, v) = ((q_2 >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
175:         $p_3(h+1, v) = ((p_0(h+1, v) >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
176:    else if mode == 1
177:         $q_0 = p_3(h-1, v-2)$           // color pixel first:  $q_0 = p_3(h, v-1)$ 
178:         $q_1 = p_1(h-1, v-2)$           // color pixel first:  $q_1 = p_1(h, v-1)$ 
179:         $q_2 = (p_2(h-1, v-2) + p_3(h-1, v-2)) >> 1$   // color pixel first:  $q_2 = (p_2(h, v-1) + p_3(h, v-1)) >> 1$ 
180:         $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
181:         $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
182:         $p_2(h+1, v) = ((q_2 >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
183:         $p_3(h+1, v) = ((p_0(h+1, v) >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
184:    else if mode == 2
185:         $q_0 = p_3(h-1, v-2)$           // color pixel first:  $q_0 = p_3(h, v-1)$ 
186:         $q_1 = (p_1(h-1, v-2) + p_3(h-1, v-2)) >> 1$   // color pixel first:  $q_1 = (p_1(h, v-1) + p_3(h, v-1)) >> 1$ 
187:         $q_2 = p_2(h-1, v-2)$           // color pixel first:  $q_2 = p_2(h, v-1)$ 
188:         $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
189:         $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
190:         $p_2(h+1, v) = ((q_2 >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
191:         $p_3(h+1, v) = ((p_0(h+1, v) >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
192:    else if mode == 3
193:         $q_0 = p_1(h-1, v-2)$           // color pixel first:  $q_0 = p_1(h, v-1)$ 
194:         $q_1 = p_1(h-1, v-2)$           // color pixel first:  $q_1 = (p_1(h, v-1) + p_2(h+1, v-2)) >> 1$ 
195:         $q_2 = p_3(h-1, v-2)$           // color pixel first:  $q_2 = p_0(h, v-1)$ 
196:         $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
197:         $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
198:         $p_2(h+1, v) = ((q_2 >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
199:         $p_3(h+1, v) = ((p_0(h+1, v) >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
200:    else if mode == 4
201:         $q_0 = p_2(h-1, v-2)$           // color pixel first:  $q_0 = p_2(h, v-1)$ 
202:         $q_1 = p_3(h-1, v-2)$           // color pixel first:  $q_1 = p_3(h, v-1)$ 
203:         $q_2 = p_2(h-1, v-2)$           // color pixel first:  $q_2 = p_2(h, v-1)$ 
204:         $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
205:         $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
206:         $p_2(h+1, v) = ((q_2 >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
207:         $p_3(h+1, v) = ((p_0(h+1, v) >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
208:    else if mode == 5

```

```

209:     $q_0 = p_0(h+3, v-2)$            // color pixel first:  $q_0 = p_0(h+2, v-1)$ 
210:     $q_1 = p_2(h+3, v-2)$            // color pixel first:  $q_1 = p_2(h+2, v-1)$ 
211:     $q_3 = p_3(h+3, v-2)$            // color pixel first:  $q_3 = p_3(h+2, v-1)$ 
212:     $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
213:     $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
214:     $p_3(h+1, v) = ((q_3 >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
215:     $p_2(h+1, v) = ((p_1(h+1, v) >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
216: else if mode == 6
217:     $q_0 = p_0(h+3, v-2)$            // color pixel first:  $q_0 = p_0(h+2, v-1)$ 
218:     $q_1 = p_2(h+3, v-2)$            // color pixel first:  $q_1 = p_2(h+2, v-1)$ 
219:     $q_3 = (p_2(h+3, v-2) + p_3(h+3, v-2)) >> 1$  // color pixel first:  $q_3 = (p_2(h+2, v-1) + p_3(h+2, v-1)) >> 1$ 
220:     $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
221:     $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
222:     $p_3(h+1, v) = ((q_3 >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
223:     $p_2(h+1, v) = ((p_1(h+1, v) >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
224: else if mode == 7
225:     $q_0 = (p_0(h+3, v-2) + p_2(h+3, v-2)) >> 1$  // color pixel first:  $q_0 = (p_0(h+2, v-1) + p_2(h+2, v-1)) >> 1$ 
226:     $q_1 = p_2(h+3, v-2)$            // color pixel first:  $q_1 = p_2(h+2, v-1)$ 
227:     $q_3 = p_3(h+3, v-2)$            // color pixel first:  $q_3 = p_3(h+2, v-1)$ 
228:     $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
229:     $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
230:     $p_3(h+1, v) = ((q_3 >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
231:     $p_2(h+1, v) = ((p_1(h+1, v) >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
232: else if mode == 8
233:     $q_0 = p_0(h+3, v-2)$            // color pixel first:  $q_0 = (p_3(h+1, v-2) + p_0(h+2, v-1)) >> 1$ 
234:     $q_1 = p_0(h+3, v-2)$            // color pixel first:  $q_1 = p_0(h+2, v-1)$ 
235:     $q_3 = p_2(h+3, v-2)$            // color pixel first:  $q_3 = p_1(h+2, v-1)$ 
236:     $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
237:     $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
238:     $p_3(h+1, v) = ((q_3 >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
239:     $p_2(h+1, v) = ((p_1(h+1, v) >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
240: else if mode == 9
241:     $q_0 = p_2(h+3, v-2)$            // color pixel first:  $q_0 = p_2(h+2, v-1)$ 
242:     $q_1 = p_3(h+3, v-2)$            // color pixel first:  $q_1 = p_3(h+2, v-1)$ 
243:     $q_3 = p_3(h+3, v-2)$            // color pixel first:  $q_3 = p_3(h+2, v-1)$ 
244:     $p_0(h+1, v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
245:     $p_1(h+1, v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
246:     $p_3(h+1, v) = ((q_3 >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
247:     $p_2(h+1, v) = ((p_1(h+1, v) >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
248: end if
249: //  $p_1(h, v), \dots, p_3(h, v)$  should be  $p_1(h, v), \dots, p_3(h, v) = p_0(h, v)$ , when you want to get the  $p_{byr}(h, v)$ 
250:  $p_{byr}(h, v) = p_0(h, v)$            // Bayer green pixel  $p_{byr}(h, v)$ 
251:  $p_{byr}(h+1, v) = p_0(h+1, v)$      // Bayer red/blue pixel  $p_{byr}(h+1, v)$ 

```

I.2.2.3 Standard Mode 2: FNR (Fixed Quantization and No-Reference Mode)

If the difference between the current pixel and the reference pixel is large, then the pixel should be compressed by fixed quantization. In Compression Mode FNR, the data is compressed using only quantization, with no difference operation.

Figure 272 defines the packet structure for Compression Mode FNR (Tetra-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 0, and Header field **Mode IDX** shall contain 3'b110.

The Payload shall contain compressed data for the four pixels of a Tetra-Cell:

- Field **Qstep** shall indicate the quantization step size.
- Fields **MltPixel1** through **MltPixel3** shall contain the quantization results for each Multi-Pixel.

Field **ByrDiff** can also be regarded as **MltDiff0**; it is used for generation of the Bayer image, for more information, see *Section I.3.4*.

Table 83 shows pseudo-code for decoding an MPC Packet for Compression Mode PD.

Mode	Header				Payload			
	Prediction IDX	Mode IDX			ByrPixel[3:0]	MltPixel1[3:0]	MltPixel2[3:0]	MltPixel3[3:0]
FNR	0	1	1	0				

Figure 273 MPC Packet Structure for Compression Mode FNR (Tetra-Cell)

Note:

In this pseudo-code, any color pixel can appear first.

Note also that this pseudo-code only describes 2x2 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

Table 83 Decoder for Compression Mode FNR (Tetra-Cell)

1:	// Quantization step
2:	shift0 = 6
3:	// For decoding the pixel p_k , $k=\{0,1,2,3\}$
4:	$p_0(h, v) = (\text{ByrPixel} << \text{shift0}) + (1 << (\text{shift0} - 1))$
5:	$p_1(h, v) = (\text{MltPixel1} << \text{shift0}) + (1 << (\text{shift0} - 1))$
6:	$p_2(h, v) = (\text{MltPixel2} << \text{shift0}) + (1 << (\text{shift0} - 1))$
7:	$p_3(h, v) = (\text{MltPixel3} << \text{shift0}) + (1 << (\text{shift0} - 1))$
8:	// $p_1(h, v), \dots, p_3(h, v)$ should be $p_1(h, v), \dots, p_3(h, v) = p_0(h, v)$, when you want to get the $p_{byr}(h, v)$
9:	$p_{byr}(h, v) = p_0(h, v)$ // Bayer pixel $p_{byr}(h, v)$

I.2.2.4 Standard Mode 3: OUT (OUTlier compensation Mode)

Compression Mode OUT performs compression when there is one outlier pixel in current pixel $p_k(h+i, v)$. It can only handle a maximum of one outlier pixel out of the four Multi-Pixels clustered within the same color filter. The OUT packet informs the MPC Decoder which pixel (out of the four) is the outlier pixel.

Decoded pixels are usually used as reference pixels. However, the decoded pixels and reference pixels are different from each other for the outlier pixels in OUT mode (see **Figure 274**). Meanwhile, for the other three pixels, decoded pixels are the same as reference pixels. The MPC Decoder replaces the outlier pixel with the average of the candidates for prediction pixel (i.e., k_0) which is added to the threshold value to get the decoded pixel. The decoded pixel is expressed as Tetra-cell $p(h, v)$, and the reference pixel is expressed as Tetra-cell $p_q(h, v)$.

- k_0 shall contain average of the candidates for prediction pixel which are not outlier pixels. It shall be a reference pixel of the outlier pixel and enables the MPC Decoder to ignore the outlier pixels when the prediction pixels are received.

If $p_0(h, v)$ through $p_3(h, v)$ in **Figure 270** are green or blue, and one of them is the outlier pixel, then:

k_0 shall be the average of $p_3(h-2, v-2)$, $p_2(h, v-2)$, $p_3(h, v-2)$, and $p_2(h+2, v-2)$

If $p_0(h+1, v)$ through $p_3(h+1, v)$ in **Figure 270** are red or green, and one of them is the outlier pixel, then:

k_0 shall be the average of $p_3(h-1, v-2)$, $p_2(h+1, v-2)$, $p_3(h+1, v-2)$, and $p_2(h+3, v-2)$

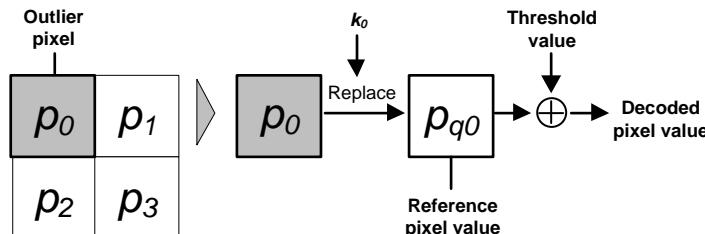


Figure 274 Decoded and Reference Pixel of Outlier Pixel (Tetra)

Figure 275 defines the packet structure for Compression Mode OUT (Tetra-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 0, and Header field **Mode IDX** shall contain 3'b111.

The Payload shall contain compressed data for the four pixels of a Tetra-Cell:

- The location of the outlier pixel in field **Pos**
- The quantization step size in field **Qstep**
- Encoded data for the three good pixels in fields **MltDiff1** through **MltDiff3** (i.e., for each good pixel, the difference between the predicted pixel and the original pixel).

For Compression Mode OUT, the Bayer image is generated by averaging neighboring pixels.

Table 84 shows pseudo-code for decoding an MPC Packet for Compression Mode OUT (Tetra-Cell).

Mode	Header				Payload				
	Prediction IDX	Mode IDX			Pos[1:0]	Qstep[1:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[3:0]
OUT	0	1	1	1					

Figure 275 MPC Packet Structure for Compression Mode OUT (Tetra-Cell)

5599

Note:

5600

5601 This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the
 5602 processing sequence must be changed, but compression is performed in the same way for each
 5603 color.

5604

Note also that this pseudo-code only describes 2x2 pixels $p_k(h, v)$ to be compressed. These should
 be Green or Red/Blue pixels.

Table 84 Decoder for Compression Mode OUT (Tetra-Cell)

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 1 : (Qstep==1) ? 2 : (Qstep==2) ? 3 : 4
3: // For decoding  $p_k(h, v)$ ,  $k=\{0,1,2,3\}$ 
4: if (Line count < 4) // Vertical line counter, Boundary condition
5:   q =  $p_1(h-2, v)$ 
6: else if (unit count == 0) // unit count is for processing unit block, boundary condition
7:   q =  $p_2(h, v-2)$ 
8: else //Prediction value q of the preview pixel
9:   if  $p_3(h-2, v-2) \geq \max\{p_2(h, v-2), p_1(h-2, v)\}$ 
10:    q =  $\min\{p_2(h, v-2), p_1(h-2, v)\}$ 
11:   else if  $p_3(h-2, v-2) \leq \min\{p_2(h, v-2), p_1(h-2, v)\}$ 
12:    q =  $\max\{p_2(h, v-2), p_1(h-2, v)\}$ 
13:   else
14:     q =  $p_2(h, v-2) + p_1(h-2, v) - p_3(h-2, v-2)$ 
15:   end if
16: end if
17:  $k_0 = (p_3(h-2, v-2) + p_2(h, v-2) + p_3(h, v-2) + p_2(h+2, v-2)) \gg 2$ 
18:  $k_1 = ((q \gg shift0 + MltDiff1) \ll shift0) + (1 \ll (shift0 - 1))$ 
19:  $k_2 = ((q \gg shift0 + MltDiff2) \ll shift0) + (1 \ll (shift0 - 1))$ 
20:  $k_3 = ((q \gg shift0 + MltDiff3) \ll shift0) + (1 \ll (shift0 - 1))$ 
21:  $k'_0 = (k_0 + outlier\_thresh) > 1023 ? 1023 : (k_0 + outlier\_thresh)$ 
22: // outlier_thresh is user control parameter to detect outlier pixel with threshold value
23: // According to outlier pixel position Pos, decoded data mapping
24: //  $p_{kq}(h, v)$  is not output data but prediction pixel for using as reference pixel
25:  $p_{0q}(h, v) = (Pos == 0) ? k_0 : k_1$ 
26:  $p_{1q}(h, v) = (Pos == 1) ? k_0 : (Pos < 1) ? k_1 : k_2$ 
27:  $p_{2q}(h, v) = (Pos == 2) ? k_0 : (Pos < 2) ? k_2 : k_3$ 
28:  $p_{3q}(h, v) = (Pos == 3) ? k_0 : k_3$ 
29: //  $p_k(h, v)$  is output data, p and  $p_q$  are different (basically these are same in other mode)
30:  $p_0(h, v) = (Pos == 0) ? k'_0 : k_1$ 
31:  $p_1(h, v) = (Pos == 1) ? k'_0 : (Pos < 1) ? k_1 : k_2$ 
32:  $p_2(h, v) = (Pos == 2) ? k'_0 : (Pos < 2) ? k_2 : k_3$ 
33:  $p_3(h, v) = (Pos == 3) ? k'_0 : k_3$ 
34: //  $p_1(h, v), \dots, p_3(h, v)$  should be  $p_1(h, v), \dots, p_3(h, v) = k'_0$ , when you want to get the  $p_{byr}(h, v)$ 
35:  $p_{byr}(h, v) = k'_0$  // Bayer pixel  $p_{byr}(h, v)$ 

```

I.2.3 Advanced Prediction Mode (Tetra-Cell)

For Tetra-Cell image sensors, Advanced Prediction Mode:

- Compresses the image using more extensive direction-aware algorithms to minimize image quality loss.
- Supports Compression Modes eMPD, eHVD, eHVA, and eOUT as defined below.

I.2.3.1 Extended Mode 0: eMPD (extended Multi-Pixel-based Directional Differential Mode)

Compared to Standard Compression Mode AD (which uses Basic Prediction Mode), Compression Mode eMPD references more pixels and allocates one more bit to each Multi-Pixel in the MPC Packet.

Figure 276 defines the packet structure for Compression Mode eMPD (Tetra-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 1, and the high bit of Header field **Mode IDX** shall contain 0.

The Payload shall contain compressed data for the four pixels of a Tetra-Cell:

- Field **Qstep** shall indicate the quantization step size.
 - Fields **ByrDiff** and **MltDiff1** through **MltDiff3** shall contain the quantized differences between the predicted value and each pixel.
- Field **ByrDiff** can also be regarded as **MltDiff0**; it is used for generation of the Bayer image, see *Section I.3.4*.

Table 85 shows pseudo-code for decoding an MPC Packet for Compression Mode eMPD (Tetra-Cell).

Mode	Header		Payload			
	Prediction IDX	Mode IDX				
eMPD	1	0	Qstep[1:0]	ByrDiff[3:0]	MltDiff1[3:0]	MltDiff2[3:0]

Figure 276 MPC Packet Structure for Compression Mode eMPD (Tetra-Cell)

5622

Note:

5623

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color. The Red- or Blue-first case is described in comments.

5624

Note also that this pseudo-code only describes 2x2 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

5625

Table 85 Decoder for Compression Mode eMPD (Tetra-Cell)

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 1 : (Qstep==1) ? 2 : (Qstep==2) ? 3 : 4
3: // Preprocessing reference data
4: threshold = green pixel ?10 : 20
5: href = abs(p3(h-2,v-2) - p2(h,v-2))
6: vref = abs(p3(h-2,v-2) - p1(h-2,v))
7: mref = href - vref
8: diag_ref = green pixel ? p3(h-1,v-1) : p3(h-1,v-2)
9: // color pixel first: green pixel ? p3(h,v-1) : p3(h-2,v-2)
10: // For decoding p0(h,v)
11: if (Line count < 4) // Vertical line counter, Boundary condition
12:     q0 = p1(h-2,v)
13: else if (unit count == 0) // unit count is for processing unit block, boundary condition
14:     q0 = p2(h,v-2)
15: else //Prediction value q
16:     if (mref < threshold)
17:         q0 = diag_ref
18:     else
19:         if p3(h-2,v-2) >= max{p2(h,v-2), p1(h-2,v)}
20:             q0 = min{p2(h,v-2), p1(h-2,v)}
21:         else if p3(h-2,v-2) <= min{p2(h,v-2), p1(h-2,v)}
22:             q0 = max{p2(h,v-2), p1(h-2,v)}
23:         else
24:             q0 = p2(h,v-2) + p1(h-2,v) - p3(h-2,v-2)
25:         end if
26:     end if
27: end if
28: p0(h,v) = ((q0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0 -1))
29: // For decoding p1(h,v)
30: if (Line count < 4) // Vertical line counter, Boundary condition
31:     q1 = p0(h,v)
32: else if (unit count == 0) // unit count is for processing unit block, boundary condition
33:     q1 = p3(h,v-2)
34: else //Prediction value q
35:     if p2(h,v-2) >= max{p3(h,v-2), p0(h,v)} // Prediction value q1 of p1(h,v)
36:         q1 = min{p3(h,v-2), p0(h,v)}
37:     else if p2(h,v-2) <= min{p3(h,v-2), p0(h,v)}

```

```

38:            $q_1 = \max\{p_3(h,v-2), p_0(h,v)\}$ 
39:           else
40:                $q_1 = p_3(h,v-2) + p_0(h,v) - p_2(h,v-2)$ 
41:           end if
42:       end if
43:        $p_1(h,v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0 - 1))$ 
44:       // For decoding  $p_2(h,v)$ 
45:       if  $q_0 == p_2(h,v-2)$ 
46:            $q_2 = p_0(h,v)$ 
47:       else
48:            $q_2 = (p_1(h-2,v) + p_3(h-2,v)) >> 1$ 
49:       end if
50:        $p_2(h,v) = ((q_2 >> shift0 + MltDiff2) << shift0) + (1 << (shift0 - 1))$ 
51:       // For decoding  $p_3(h,v)$ 
52:       if  $q_1 == p_0(h,v)$ 
53:            $q_3 = p_2(h,v)$ 
54:       else if  $q_1 == p_3(h,v-2)$ 
55:            $q_3 = p_1(h,v)$ 
56:       else
57:            $q_3 = p_0(h,v)$ 
58:       end if
59:        $p_3(h,v) = ((q_3 >> shift0 + MltDiff3) << shift0) + (1 << (shift0 - 1))$ 
60:       //  $p_1(h,v), \dots, p_3(h,v)$  should be  $p_1(h,v), \dots, p_3(h,v) = p_0(h,v)$ , when you want to get the  $p_{byr}(h,v)$ 
61:        $p_{byr}(h,v) = p_0(h,v)$       // Bayer pixel  $p_{byr}(h,v)$ 

```

I.2.3.2 Extended Mode 1: eHVD (extended Horizontal or Vertical Direction-based Differential Mode)

Compression Mode eHVD references pixels that are located along the horizontal or vertical direction.

Figure 277 defines the packet structure for Compression Mode eHVD (Tetra-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 1, and the high two bits of Header field **Mode IDX** shall contain 2'b10.

The Payload shall contain compressed data for the four pixels of a Tetra-Cell:

- 1-bit field **Dir** shall indicate the reference direction:

If **Dir** contains 0 then the reference direction is the vertical direction

If **Dir** contains 1 then the reference direction is the horizontal direction

- Field **Qstep** shall indicate the quantization step size.
- Fields **ByrDiff** and **MltDiff1** through **MltDiff3** shall contain the quantized differences between the predicted value and each pixel.

Field **ByrDiff** can also be regarded as **MltDiff0**; it is used for generation of the Bayer image, see **Section I.3.4**.

Table 86 shows pseudo-code for decoding an MPC Packet for Compression Mode eVHD (Tetra-Cell).

Mode	Header				Payload				
	Prediction IDX	Mode IDX			Qstep[1:0]	ByrDiff[3:0]	MltDiff1[3:0]	MltDiff2[2:0]	MltDiff3[2:0]
eHVD	1	1	0	Dir	Qstep[1:0]	ByrDiff[3:0]	MltDiff1[3:0]	MltDiff2[2:0]	MltDiff3[2:0]

Figure 277 MPC Packet Structure for Compression Mode eHVD (Tetra-Cell)

5644

Note:

5645

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color.

5646

Note also that this pseudo-code only describes 2x2 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

5647

5648

5649

Table 86 Decoder for Compression Mode eHVD (Tetra-Cell)

```

1: // Quantization step
2: shift0 = (Qstep == 0) ? 1 : (Qstep == 1) ? 2 : (Qstep == 2) ? 3 : 4
3: shift1 = (Qstep == 0) ? 2 : (Qstep == 1) ? 3 : (Qstep == 2) ? 4 : 5
4: if Dir == 1 // Horizontal direction
5:   q0 = p1(h-2,v)
6:   q2 = p3(h-2,v)
7:   p0(h,v) = ((q0>>shift0 + ByrDiff) << shift0) + (1 << (shift0-1))
8:   p2(h,v) = ((q2>>shift0 + MltDiff1) << shift0) + (1 << (shift0-1))
9:   q1 = p0(h,v)
10:  q3 = p2(h,v)
11:  p1(h,v) = ((q1>>shift1 + MltDiff2) << shift1) + (1 << (shift1-1))
12:  p3(h,v) = ((q3>>shift1 + MltDiff3) << shift1) + (1 << (shift1-1))
13: else // Vertical direction
14:   q0 = p2(h,v-2)
15:   q1 = p3(h,v-2)
16:   p0(h,v) = ((q0>>shift0 + ByrDiff) << shift0) + (1 << (shift0-1))
17:   p1(h,v) = ((q1>>shift0 + MltDiff1) << shift0) + (1 << (shift0-1))
18:   q2 = p0(h,v)
19:   q3 = p1(h,v)
20:   p2(h,v) = ((q2>>shift1 + MltDiff2) << shift1) + (1 << (shift1-1))
21:   p3(h,v) = ((q3>>shift1 + MltDiff3) << shift1) + (1 << (shift1-1))
22: end if
23: // p1(h,v),...,p3(h,v) should be p1(h,v),...,p3(h,v) = p0(h,v), when you want to get the pbyr(h,v)
24: pbyr(h,v) = p0(h,v) // Bayer pixel pbyr(h,v)

```

I.2.3.3 Extended Mode 2: eHVA (extended Horizontal or Vertical Average-based Differential Mode)

Compression Mode eHVA takes the horizontal or vertical averages of the four Multi-Pixels in the Tetra-Cell.

Figure 279 defines the packet structure for Compression Mode eHVA. For this Compression Mode, Header field **Prediction IDX** shall contain 1, and Header field **Mode IDX** shall contain 3'b110.

The Payload shall contain compressed data for the four pixels of a Tetra-Cell, organized as two 2-pixel couplets (see **Figure 278**):

- 1-bit field **Dir** shall indicate the compression orientation for this MPC Packet. The value of field **Dir** shall determine how all of the other fields are interpreted.

If Dir contains 0: The orientation shall be vertical, and each decoded pixel couplet shall be 1x2 pixels. The first pixel couplet shall be $p_0(h, v)$ and $p_2(h, v)$, and the second pixel couplet shall be $p_1(h, v)$ and $p_3(h, v)$.

If Dir contains 1: The orientation shall be horizontal, and each decoded pixel couplet shall be 2x1 pixels. The first pixel couplet shall be $p_0(h, v)$ and $p_1(h, v)$, and the second pixel couplet shall be $p_2(h, v)$ and $p_3(h, v)$.

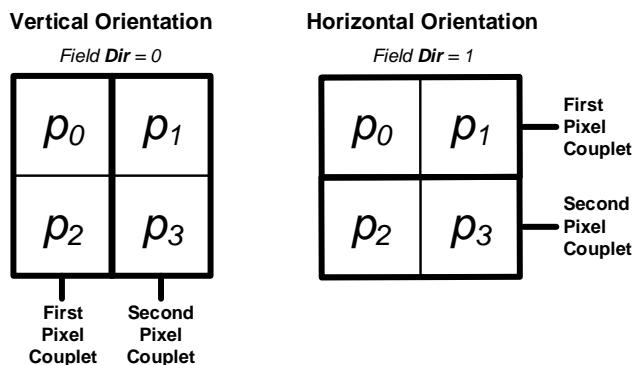


Figure 278 Pixel Couplet Definitions for eHVA Mode

- Field **ByrDiff** shall contain the quantized differences between the predicted value and the average of the two pixels in the first pixel couplet. Field **MltDiff** shall contain the quantized differences between the predicted value and the average of the two pixels in the second pixel couplet.

Note:

*Field **ByrDiff** can also be regarded as **MltDiff**; it is used for generation of the Bayer image, see **Section I.3.4**.*

That is,

If field Dir indicates vertical orientation (value 0), then:

- Field **ByrDiff** shall contain the average of $p_0(h, v)$ and $p_2(h, v)$
- Field **MltDiff** shall contain the average of $p_1(h, v)$ and $p_3(h, v)$

If field Dir indicates horizontal orientation (value 1), then:

- Field **ByrDiff** shall contain the average of $p_0(h, v)$ and $p_1(h, v)$
- Field **MltDiff** shall contain the average of $p_2(h, v)$ and $p_3(h, v)$

- 5677 • 1-bit field **Slope** shall indicate, for both pixel couplets, whether the slope of the gradient is
 5678 increasing or decreasing in the **Dir** direction.

5679 **If Slope contains 0**, then the gradient slope is decreasing and the MPC packet shall be decoded
 5680 as:

5681 First pixel couplet = (**ByrDiff** + *alpha*, **ByrDiff** – *alpha*)
 5682 Second pixel couplet = (**MltDiff** + *beta*, **MltDiff** – *beta*)

5683 **If Slope contains 1**, then the gradient slope is increasing and the MPC Packet shall be decoded as:

5684 First pixel couplet = (**ByrDiff** – *alpha*, **ByrDiff** + *alpha*)
 5685 Second pixel couplet = (**MltDiff** – *beta*, **MltDiff** + *beta*)

5686 Where *alpha* and *beta* are both defined as:

5687 4 for compression ratio 10:5, or
 5688 1 for compression ratio 10:6, or
 5689 0 for compression ratio 10:7

- 5690 • 1-bit fields **Detail[0]** and **Detail[1]** shall determine, for each pixel couplet, whether the pixel value
 5691 variations *alpha* and *beta* shown above shall be (value 1) or shall not be (value 0) replaced with
 5692 the difference from the neighboring pixels.

5693 **Example:** Assume that field **Slope** indicates decreasing slope (value 0). If field **Detail[0]**
 5694 contains 1, then the first pixel couplet shall be decoded not as (**MltDiff** + *alpha*, **MltDiff** –
 5695 *alpha*) as shown above, but rather as: (**MltDiff** + *delta*, **MltDiff** – *delta*) (where *delta* is the
 5696 difference from the neighboring pixels).

5697 **Table 87** shows pseudo-code for decoding an MPC Packet for Compression Mode eHVA.

Mode	Header				Payload			
	Prediction IDX		Mode IDX		ByrDiff[5:0]		MltDiff[5:0]	Dir Slope Detail[1:0]
eHVA	1	1	1	0				

5698 **Figure 279 MPC Packet Structure for Compression Mode eHVA**

5699 **Note:**

5700 This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the
 5701 processing sequence must be changed, but compression is performed in the same way for each
 5702 color.

5703 Note also that this pseudo-code only describes 2x2 pixels $p_k(h, v)$ to be compressed. These should
 5704 be Green or Red/Blue pixels.

Table 87 Decoder for Compression Mode eHVA

1:	// Quantization step
2:	shift0 = 4
3:	average0 = (ByrDiff << shift0) + (1 << (shift0-1))
4:	average1 = (MltDiff << shift0) + (1 << (shift0-1))
5:	if Dir == 0 // Vertical direction
6:	$q_0 = p_0(h, v-2)$
7:	$q_1 = p_2(h, v-2)$
8:	$q_2 = p_1(h, v-2)$
9:	$q_3 = p_3(h, v-2)$
10:	else // Horizontal direction, Dir == 1 case
11:	$q_0 = p_0(h-2, v)$
12:	$q_1 = p_2(h-2, v)$

```

13:       $q_2 = p_1(h-2, v)$ 
14:       $q_3 = p_3(h-2, v)$ 
15:  end if
16:  if Detail[1] == 0 // for Horizontal direction  $p_0(h, v), p_1(h, v)$  or Vertical direction  $p_0(h, v), p_2(h, v)$ 
17:     $diff_0 = \text{abs}(q_0 - q_1) >> 1$ 
18:  else
19:     $diff_0 = 4$ 
20:  end if
21:  if Detail[0] == 0 // for Horizontal direction  $p_2(h, v), p_3(h, v)$  or Vertical direction  $p_1(h, v), p_3(h, v)$ 
22:     $diff_1 = \text{abs}(q_2 - q_3) >> 1$ 
23:  else
24:     $diff_1 = 4$ 
25:  end if
26:  if Dir == 1 // Horizontal direction
27:    if Slope == 1
28:       $p_0(h, v) = average0 - diff_0$ 
29:       $p_1(h, v) = average0 + diff_0$ 
30:       $p_2(h, v) = average1 - diff_1$ 
31:       $p_3(h, v) = average1 + diff_1$ 
32:    else
33:       $p_0(h, v) = average0 + diff_0$ 
34:       $p_1(h, v) = average0 - diff_0$ 
35:       $p_2(h, v) = average1 + diff_1$ 
36:       $p_3(h, v) = average1 - diff_1$ 
37:    end if
38:  else // Vertical direction
39:    if Slope == 1
40:       $p_0(h, v) = average0 - diff_0$ 
41:       $p_1(h, v) = average1 - diff_1$ 
42:       $p_2(h, v) = average0 + diff_0$ 
43:       $p_3(h, v) = average1 + diff_1$ 
44:    else
45:       $p_0(h, v) = average0 + diff_0$ 
46:       $p_1(h, v) = average1 + diff_1$ 
47:       $p_2(h, v) = average0 - diff_0$ 
48:       $p_3(h, v) = average1 - diff_1$ 
49:    end if
50:     $p_0(h, v) = (p_0(h, v) > 1023) ? 1023 : (p_0(h, v) < 0) ? 0 : p_0(h, v)$ 
51:     $p_1(h, v) = (p_1(h, v) > 1023) ? 1023 : (p_1(h, v) < 0) ? 0 : p_1(h, v)$ 
52:     $p_2(h, v) = (p_2(h, v) > 1023) ? 1023 : (p_2(h, v) < 0) ? 0 : p_2(h, v)$ 
53:     $p_3(h, v) = (p_3(h, v) > 1023) ? 1023 : (p_3(h, v) < 0) ? 0 : p_3(h, v)$ 
54:    //  $p_1(h, v), \dots, p_3(h, v)$  should be  $p_1(h, v), \dots, p_3(h, v) = average0$ , when you want to get the  $p_{byr}(h, v)$ 
55:     $p_{byr}(h, v) = average0 // Bayer pixel  $p_{byr}(h, v)$$ 

```

I.2.3.4 Extended Mode 3: eOUT (extended OUTlier Compensation Mode)

Compression Mode eOUT is intended for compressing pixels that are located at the corners of object edge pattern.

Figure 280 defines the packet structure for Compression Mode eOUT. For this Compression Mode, Header field **Prediction IDX** shall contain 1, and Header field **Mode IDX** shall contain 3'b111.

The Payload shall contain compressed data for the four pixels of a Tetra-Cell:

- 1-bit field **Opt** shall indicate whether to invert the value of the prediction pixel:
 - **0: Normal Reference Value:** The actual value of the prediction pixel shall be used.
 - **1: Inverse Reference Value:** The inverse of the value of the prediction pixel shall be used. That is, the magnitude of the prediction pixel value shall be inverted, using the formula:
reference value = 1023 – prediction pixel value
- Fields **ByrDiff** and **MltDiff1** through **MltDiff3** shall contain the quantized differences between the predicted value and each pixel.
Field **ByrDiff** can also be regarded as **MltDiff0**; it is used for generation of the Bayer image, see *Section I.3.4*.

Table 88 shows pseudo-code for decoding an MPC Packet for Compression Mode eOUT.

Mode	Header				Payload			
	Prediction IDX		Mode IDX		Opt	ByrDiff[4:0]	MltDiff1[3:0]	MltDiff2[2:0]
eOUT	1	1	1	1				

Figure 280 MPC Packet Structure for Compression Mode eOUT

5721

Note:

5722

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color.

5723

5724

5725

5726

Note also that this pseudo-code describes each case of 2x2 Green $p_k(h, v)$ and 2x2 Red/Blue $p_k(h+1, v)$ pixels.

Table 88 Decoder for Compression Mode eOUT

```

1:   /***** For decoding the green pixel  $p_k(h,v)$ ,  $k=\{0,1,2,3\}$  *****/
2:   // Quantization step
3:   shift0 = 4
4:   shift1 = 5
5:   // Offset value definition
6:   offset1 = 64
7:   offset2 = 32
8:   offset3 = 64
9:   // Prediction pixel value definition
10:  q1 = (p2(h+1, v-1) + offset1)
11:  q1 = (q1 > 1023) ? 1023 : (q1 < 0) ? 0 : q1
12:  if Qpt == 1
13:    q_inv1 = 1023 - q1
14:    q1 = (q_inv1 == 0) ? q1 : q_inv1
15:  end if
16:  p1(h, v) = ((q1 >> shift1 + MltDiff2) << shift1) + (1 << (shift1-1))
17:  // Prediction pixel value definition
18:  q3 = (p3(h+1, v-1) + offset3)
19:  q3 = (q3 > 1023) ? 1023 : (q3 < 0) ? 0 : q3
20:  if Qpt == 1
21:    q_inv3 = 1023 - q3
22:    q3 = (q_inv3 == 0) ? q3 : q_inv3
23:  end if
24:  p3(h, v) = ((q3 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))
25:  // Prediction pixel value definition
26:  q0 = (p1(h, v) > p3(h, v)) ? p1(h, v) : p3(h, v)
27:  p0(h, v) = ((q0 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))
28:  // Prediction pixel value definition
29:  if (Line count < 4)           // Vertical line counter, boundary condition
30:    q2 = p2(h, v)
31:  else if (unit count == 0)     // unit count is for processing unit block, boundary condition
32:    q2 = p1(h, v)
33:  else                         //Prediction value q
34:    if p0(h, v) >= max{ p1(h, v), p2(h, v) }
35:      q2 = min{ p1(h, v), p2(h, v) }
36:    else if p0(h, v) <= min{ p1(h, v), p2(h, v) }
```

```

37:          $q_2 = \max\{p_1(h,v), p_2(h,v)\}$ 
38:     else
39:          $q_2 = p_1(h,v) + p_2(h,v) - p_0(h,v)$ 
40:     end if
41:   end if
42:    $q_2 = q_2 + offset_2$ 
43:    $q_2 = (q_2 > 1023) ? 1023 : (q_2 < 0) ? 0 : q_2$ 
44:    $p_2(h,v) = ((q_2 >> shift1 + MltDiff3) << shift1) + (1 << (shift1-1))$ 
45:   //  $p_1(h,v), \dots, p_3(h,v)$  should be  $p_1(h,v), \dots, p_3(h,v) = p_0(h,v)$ , when you want to get the  $p_{byr}(h,v)$ 
46:    $p_{byr}(h,v) = p_0(h,v)$       // Bayer pixel  $p_{byr}(h,v)$ 
47:   //***** For decoding the red/blue pixel  $p_i(h+1,v)$ ,
48:   // Quantization step
49:   shift0 = 4
50:   shift1 = 5
51:   // Offset value definition
52:   offset1 = 64
53:   offset2 = 32
54:   offset3 = 64
55:   // Prediction pixel value definition
56:    $q_1 = (p_3(h+1,v-2) + offset_1)$ 
57:    $q_1 = (q_1 > 1023) ? 1023 : (q_1 < 0) ? 0 : q_1$ 
58:   if Qpt == 1
59:        $q\_inv_1 = 1023 - q_1$ 
60:        $q_1 = (q\_inv_1 == 0) ? q_1 : q\_inv_1$ 
61:   end if
62:    $p_1(h+1,v) = ((q_1 >> shift1 + MltDiff2) << shift1) + (1 << (shift1-1))$ 
63:   // Prediction pixel value definition
64:    $q_3 = (p_3(h+1,v-2) + offset_3)$ 
65:    $q_3 = (q_3 > 1023) ? 1023 : (q_3 < 0) ? 0 : q_3$ 
66:   if Qpt == 1
67:        $q\_inv_3 = 1023 - q_3$ 
68:        $q_3 = (q\_inv_3 == 0) ? q_3 : q\_inv_3$ 
69:   end if
70:    $p_3(h+1,v) = ((q_3 >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
71:   // Prediction pixel value definition
72:    $q_0 = (p_1(h,v) > p_3(h,v)) ? p_1(h,v) : p_3(h,v)$ 
73:    $p_0(h+1,v) = ((q_0 >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
74:   // Prediction pixel value definition
75:   if  $p_0(h,v) < \min\{p_1(h,v), p_2(h,v)\}$ 
76:        $q_2 = \max\{p_1(h,v), p_2(h,v)\}$ 
77:   else if  $p_0(h,v) > \max\{p_1(h,v), p_2(h,v)\}$ 
78:        $q_2 = \min\{p_1(h,v), p_2(h,v)\}$ 
79:   else

```

```
74:      $q_2 = p_1(h,v) + p_2(h,v) - p_0(h,v)$ 
75:     end if
76:      $q_2 = q_2 + offset_2$ 
77:      $q_2 = (q_2 > 1023) ? 1023 : (q_2 < 0) ? 0 : q_2$ 
78:      $p_2(h+1,v) = ((q_2 >> shift1 + MltDiff3) << shift1) + (1 << (shift1-1))$ 
79:     //  $p_1(h+1,v), \dots, p_3(h+1,v)$  should be  $p_1(h+1,v), \dots, p_3(h+1,v) = p_0(h+1,v)$ ,
80:     // when you want to get the  $p_{byr}(h+1,v)$ 
81:      $p_{byr}(h+1,v) = p_0(h+1,v)$       // Bayer pixel  $p_{byr}(h+1,v)$ 
```

I.2.4 Low Resolution Image Generation (Tetra-Cell)

The MPC architecture for Tetra-Cell image sensors supports an additional low-resolution Bayer image. The Bayer image is reconstructed by p_{byr} which is described in the Compression Mode Decoder pseudo-code in **Section I.2.2.1** through **Section I.2.3.4**.

For all Tetra-Cell Compression Modes, the Bayer image can be generated using only the information that is present in the first half of the MPC Packet. This is shown in **Figure 281**.

- For Compression Mode OUT, the Bayer image is generated by averaging neighboring pixels (see **Section I.2.2.1**).
- For all other Compression Modes, the Bayer image is generated using field **ByrDiff** or **ByrPixel**.

Mode Summary for Tetra-Cell Image Sensor									
Compression Mode	Header		Payload						
	Prediction IDX	Mode IDX							
<i>Basic Prediction Mode</i>									
AD	0	0	Qstep[1:0]	ByrDiff[3:0]	M0Diff1[2:0]	M1Diff1[2:0]	M1Diff2[2:0]	M1Diff3[2:0]	
OD	0	1	0	Dir	Qstep[1:0]	ByrDiff[3:0]	M0Diff1[3:0]	M1Diff2[2:0]	M1Diff3[2:0]
FNR	0	1	1	0	ByrPixel[3:0]	M0Pixel1[3:0]	M0Pixel2[3:0]	M0Pixel3[3:0]	
OUT	0	1	1	1	Pos[1:0]	Qstep[1:0]	M0Diff1[3:0]	M0Diff2[3:0]	M0Diff3[3:0]
<i>Advanced Prediction Mode</i>									
eMPD	1	0	Qstep[1:0]	ByrDiff[3:0]	M0Diff1[3:0]	M0Diff2[3:0]	M0Diff3[3:0]		
eHVD	1	1	0	Dir	Qstep[1:0]	ByrDiff[3:0]	M0Diff1[3:0]	M0Diff2[2:0]	M0Diff3[2:0]
eHVA	1	1	1	0		ByrDiff[5:0]	M0Diff1[3:0]	Dir Slope Detail[1:0]	
eOUT	1	1	1	1	Opt	ByrDiff[4:0]	M0Diff1[3:0]	M0Diff2[2:0]	M0Diff3[2:0]

Figure 281 Low Resolution Image Generation by P_{byr} (Tetra-Cell)

5735

I.2.5 Adjustment of Compression Ratio (Tetra-Cell)

The MPC architecture for Tetra-Cell image sensors supports three compression ratios: 10:5, 10:6, and 10:7. Decoding uses the same algorithm for all Compression Ratios (with one exception: Compression Mode OUT with Compression Ratio 10:7 has a different decoding process, bit allocation, and the shift amount). Higher Compression Ratios are achieved by extending the bit allocation and the shift amount (i.e., field **Qstep**).

Table 89 shows pseudo-code for decoding an MPC Packet for Compression Mode OUT (Tetra-Cell) with the 10:7 Compression Ratio.

Table 89 Decoder for Compression Mode OUT (Tetra-Cell) with 10:7 Compression Ratio

```

1:   // Quantization step
2:   shift0 = 2, shift1 = 3
3:   // For decoding  $p_k(h,v)$ ,  $k=\{0,1,2,3\}$ 
11:   $k_0 = (p_3(h-2,v-2) + p_2(h,v-2) + p_3(h,v-2) + p_2(h+2,v-2)) >> 2$ 
12:   $k_1 = (\text{MltDiff1} << \text{shift0}) + (1 << (\text{shift0} - 1))$ 
13:   $k_2 = (\text{MltDiff2} << \text{shift1}) + (1 << (\text{shift1} - 1))$ 
14:   $k_3 = (\text{MltDiff3} << \text{shift1}) + (1 << (\text{shift1} - 1))$ 
15:   $k^* = (k_0 + \text{outlier\_thresh}) > 1023 ? 1023 : (k_0 + \text{outlier\_thresh})$ 
16:  // According to outlier pixel position Pos, decoded data mapping
17:  //  $p_{kq}(h,v)$  is not output data but prediction pixel for using as reference pixel
18:   $p_{0q}(h,v) = (\text{Pos} == 0) ? k_0 : k_1$ 
19:   $p_{1q}(h,v) = (\text{Pos} == 1) ? k_0 : (\text{Pos} < 1) ? k_1 : k_2$ 
20:   $p_{2q}(h,v) = (\text{Pos} == 2) ? k_0 : (\text{Pos} < 2) ? k_2 : k_3$ 
21:   $p_{3q}(h,v) = (\text{Pos} == 3) ? k_0 : k_3$ 
22:  //  $p_k(h,v)$  is output data,  $p$  and  $p_q$  are different (basically these are same in other mode)
23:   $p_0(h,v) = (\text{Pos} == 0) ? k^* : k_1$ 
24:   $p_1(h,v) = (\text{Pos} == 1) ? k^* : (\text{Pos} < 1) ? k_1 : k_2$ 
25:   $p_2(h,v) = (\text{Pos} == 2) ? k^* : (\text{Pos} < 2) ? k_2 : k_3$ 
26:   $p_3(h,v) = (\text{Pos} == 3) ? k^* : k_3$ 
27:   $p_{byr}(h,v) = k^* \quad // \text{Bayer pixel } p_{byr}(h,v)$ 

```

5742
5743 **Figure 282** and **Figure 283** define the Packet structures for Compression Ratios 10:6 and 10:7, respectively, for Tetra-Cell. Differences in the bit allocations are highlighted in red.

Mode Summary for Tetra-Cell Image Sensor								
Mode	Header			Payload				
	Profile IDX	Mode IDX						
<i>Basic Prediction Mode</i>								
AD	0	0	Qstep[1:0]	ByrDiff[3:0]	MltDiff0[3:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[3:0]
OD	0	1	0 Dir	Qstep[1:0]	ByrDiff[4:0]	MltDiff1[4:0]	MltDiff2[3:0]	MltDiff3[3:0]
FNR	0	1	1 0	ByrPixel[4:0]	MltPixel1[4:0]	MltPixel2[4:0]	MltPixel3[4:0]	
OUT	0	1	1 1	Pos[1:0] Qstep[1:0]	MltDiff1[5:0]	MltDiff2[4:0]	MltDiff3[4:0]	
<i>Advanced Prediction Mode</i>								
eMPD	1	0	Qstep[1:0]	ByrDiff[4:0]	MltDiff1[4:0]	MltDiff2[4:0]	MltDiff3[4:0]	
eHVD	1	1	0 Dir	Qstep[1:0]	ByrDiff[4:0]	MltDiff1[4:0]	MltDiff2[3:0]	MltDiff3[3:0]
eHVA	1	1	1 0	ByrDiff[6:0]	MltDiff[6:0]	Dir	Slope	Detail[1:0]
eOUT	1	1	1 1 Opt	ByrDiff[5:0]	MltDiff1[4:0]	MltDiff2[3:0]	MltDiff3[3:0]	

5744

Figure 282 MPC Packet Structures for Compression Ratio 10:6 (Tetra-Cell)

Mode Summary for Tetra-Cell Image Sensor								
Mode	Header			Payload				
	Profile IDX	Mode IDX						
<i>Basic Prediction Mode</i>								
AD	0	0	Qstep[1:0]	ByrDiff[3:0]	MltDiff0[4:0]	MltDiff1[4:0]	MltDiff2[4:0]	MltDiff3[4:0]
OD	0	1	0 Dir	Qstep[1:0]	ByrDiff[5:0]	MltDiff1[5:0]	MltDiff2[4:0]	MltDiff3[4:0]
FNR	0	1	1 0	ByrPixel[5:0]	MltPixel1[5:0]	MltPixel2[5:0]	MltPixel3[5:0]	
OUT	0	1	1 1	Pos[1:0]	MltDiff1[7:0]	MltDiff2[6:0]	MltDiff3[6:0]	
<i>Advanced Prediction Mode</i>								
eMPD	1	0	Qstep[1:0]	ByrDiff[5:0]	MltDiff1[5:0]	MltDiff2[5:0]	MltDiff3[5:0]	
eHVD	1	1	0 Dir	Qstep[1:0]	ByrDiff[5:0]	MltDiff1[5:0]	MltDiff2[4:0]	MltDiff3[4:0]
eHVA	1	1	1 0	ByrDiff[7:0]	MltDiff[7:0]	Dir	Slope	Detail[1:0]
eOUT	1	1	1 1 Opt	ByrDiff[6:0]	MltDiff1[5:0]	MltDiff2[4:0]	MltDiff3[4:0]	

5745

Figure 283 MPC Packet Structures for Compression Ratio 10:7 (Tetra-Cell)

5746
5747
5748

Figure 284 defines the values of *shift0* and *shift1* to use (i.e., in the respective Decoder pseudo-code per Compression Mode) for each combination of Tetra-Cell Compression Mode, value of field **Qstep**, and Compression Ratio.

Quantization Bits Map							
Basic Profile							
	Quant. IDX (Qstep)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)
AD	0	1	0	1	0	1	0
	1	2	1	2	1	2	1
	2	3	2	3	2	3	2
	3	5	3	5	3	5	3
OD	0	2	-	0	-	0	-
	1	3	-	1	-	1	-
	2	4	-	2	-	2	-
	3	5	-	4	-	3	-
FNR	-	6	-	5	-	4	-
OUT	0	1	-	0	-	-	-
	1	2	-	1	-	-	-
	2	3	-	2	-	-	-
	3	4	-	3	-	-	-
Advanced Profile							
	Quant. IDX (Qstep)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)
eMPD	0	1	-	0	-	0	-
	1	2	-	1	-	1	-
	2	3	-	3	-	2	-
	3	4	-	4	-	3	-
eHVD	0	1	2	0	1	0	1
	1	2	3	1	2	1	2
	2	3	4	2	3	2	3
	3	4	5	3	4	3	4
eHVA	-	-	-	-	-	-	-
eOUT	-	4	5	3	4	2	3

5749

Figure 284 Quantization Bits for Compression Ratios (Tetra-Cell)

I.3 MPC for Nona-Cell Image Sensor

MPC for Nona-Cell image sensors supports the eight Compression Modes that are defined in this Section and summarized in *Table 90*. If MPC is supported, then all eight MPC decoder Data Compression Modes shall be supported. Each Compression Mode uses either Basic Prediction Mode or Advanced Prediction Mode. *Figure 285* summarizes the MPC Packet structures used for Nona-Cell Compression Modes.

Table 90 MPC Compression Modes for Nona-Cell Image Sensors

Compression Mode Name	Mode Description	Defined in Section
Basic Prediction Mode:		
AD	Average-based Directional Differential Mode	<i>I.3.2.1</i>
DGD	Diagonal Direction-based Differential Mode	<i>I.3.2.1</i>
FNR	Fixed Quantization and No-Reference Mode	<i>I.3.2.1</i>
Advanced Prediction Mode:		
eAD	extended Average-based Directional Differential Mode	<i>I.3.3.1</i>
eHVD	extended Horizontal or Vertical Direction-based Differential Mode	<i>I.3.3.2</i>
eHVI	extended Horizontal or Vertical Average-based Intra-Cell Differential Mode	<i>I.3.3.2</i>
eSHV	extended Slanted Horizontal or Vertical Direction-based Differential Mode	<i>I.3.3.2</i>
eMPD	extended Multi-Pixel-based Directional Differential Mode	<i>I.3.3.5</i>

Mode Summary for Nona-Cell Image Sensor														
Compression Mode	Header			Payload										
	Prediction IDX	Mode IDX		Basic Profile										
AD	0	0	Qstep[1:0]	ByrDiff[4:0]	MltDiff0[3:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[3:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[3:0]	MltDiff7[3:0]	MltDiff8[3:0]	
DGD	0	1	0 0 0	Qstep[1:0]	ByrDiff[4:0]	MltDiff1[4:0]	MltDiff2[5:0]	MltDiff3[4:0]	MltDiff4[5:0]	MltDiff5[2:0]	MltDiff6[2:0]	MltDiff7[2:0]	MltDiff8[2:0]	
FNR	0	1	0 1	ByrPixel[4:0]	MltPixel1[3:0]	MltPixel2[4:0]	MltPixel3[3:0]	MltPixel4[4:0]	MltPixel5[3:0]	MltPixel6[4:0]	MltPixel7[3:0]	MltPixel8[4:0]		
-	0	1	1 0							RESERVED				
-	0	1	1 1							RESERVED				
<i>Advanced Profile</i>														
eAD	1	0	0	Qstep	ByrDiff[4:0]	MltDiff0[3:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[3:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[3:0]	MltDiff7[3:0]	MltDiff8[3:0]
eHVD	1	0	1	Dir	Qstep[1:0]	ByrDiff[4:0]	MltDiff1[4:0]	MltDiff2[4:0]	MltDiff3[3:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[3:0]	MltDiff7[3:0]	MltDiff8[3:0]
eHVI	1	1	0	Dir	Qstep[1:0]	ByrDiff[6:0]	MltDiff1[6:0]	MltDiff2[6:0]	MltDiff3[2:0]	MltDiff4[2:0]	MltDiff5[2:0]	MltDiff6[2:0]	MltDiff7[2:0]	MltDiff8[2:0]
eSHV	1	1	1	0	Qstep[1:0]	ByrDiff[3:0]	MltDiff1[4:0]	MltDiff2[4:0]	MltDiff3[4:0]	MltDiff4[4:0]	MltDiff5[2:0]	MltDiff6[3:0]	MltDiff7[3:0]	MltDiff8[3:0]
eMPD	1	1	1	1	Qstep[1:0]	ByrDiff[4:0]	MltDiff0[3:0]	MltDiff2[3:0]	MltDiff3[4:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[4:0]	MltDiff7[3:0]	MltDiff8[3:0]

Figure 285 MPC Packet Structures for Nona-Cell Compression Modes

I.3.1 Reference Pixels for Prediction (Nona-Cell)

MPC compresses each pixel by referring to neighboring pixels. The reference pixels are selected according to the Compression Mode and the color of the current pixel.

In a Nona-Cell image, each cell consists of nine Multi-Pixels in a 3x3 matrix.

A given pixel is expressed as $p_k(i, j)$, where:

- k : The Multi-Pixel index within the Nona-Cell, i.e., 0–8 as shown in **Figure 286**.

For example:

- p_0 is the upper left Multi-pixel
- p_4 is the center Multi-Pixel, and
- p_8 is the lower right Multi-Pixel.

p_0	p_1	p_2
p_3	p_4	p_5
p_6	p_7	p_8

Figure 286 Nona-Cell Multi-Pixel Indexing

- i : Horizontal index of the Nona-Cell where the Multi-Pixel is located.

- j : Vertical index of the Nona-Cell where the Multi-Pixel is located.

For a current Nona-Cell $p(h, v)$, the coordinates are relative to a reference Nona-Cell and are expressed as:

- The Horizontal relative index from the current Nona-Cell to the reference Nona-Cell,

Reference Nona-Cells to the left of the current Nona-Cell have negative h indexes,
reference Nona-Cells directly above or below the current Nona-Cell have zero h indexes,
and reference Nona-Cells to the right of the current Nona-Cell have positive h indexes.

and

- The Vertical relative index from the current Nona-Cell to the reference Nona-Cell

Reference Nona-Cell located above the current Nona-Cell have negative v indexes, and
reference Nona-Cell in the same image line as the current Nona-Cell have zero v indexes.

Examples:

$p(h, v-1)$ Nona-Cell immediately above the current Nona-Cell

$p(h-1, v)$ Nona-Cell immediately to the left of the current Nona-Cell

$p(h+1, v-1)$ Nona-Cell immediately above and immediately to the right of the current Nona-Cell

$p(h-1, v-1)$ Nona-Cell immediately above and immediately to the left of the current Nona-Cell

In the following sub-sections, all reference pixel coordinates are expressed using this notation.

5783
5784

Using this notation, the relative coordinates of reference pixels are as illustrated in **Figure 287**, where the color of each square indicates the color filter of the corresponding pixel.

Green Pixel First

p_0 ($h-2v-2$)	p_1 ($h-2v-2$)	p_2 ($h-2v-2$)	p_0 ($h-1v-2$)	p_1 ($h-1v-2$)	p_2 ($h-1v-2$)	p_0 ($h-v-2$)	p_1 ($h-v-2$)	p_2 ($h-v-2$)	p_0 ($h,v-2$)	p_1 ($h,v-2$)	p_2 ($h,v-2$)	p_0 ($h+1v-2$)	p_1 ($h+1v-2$)	p_2 ($h+1v-2$)	p_0 ($h+2v-2$)	p_1 ($h+2v-2$)	p_2 ($h+2v-2$)	p_0 ($h+3v-2$)	p_1 ($h+3v-2$)	p_2 ($h+3v-2$)
p_3 ($h-2v-2$)	p_4 ($h-2v-2$)	p_5 ($h-2v-2$)	p_3 ($h-1v-2$)	p_4 ($h-1v-2$)	p_5 ($h-1v-2$)	p_3 ($h,v-2$)	p_4 ($h,v-2$)	p_5 ($h,v-2$)	p_3 ($h,v-2$)	p_4 ($h,v-2$)	p_5 ($h,v-2$)	p_3 ($h+1v-2$)	p_4 ($h+1v-2$)	p_5 ($h+1v-2$)	p_3 ($h+2v-2$)	p_4 ($h+2v-2$)	p_5 ($h+2v-2$)	p_3 ($h+3v-2$)	p_4 ($h+3v-2$)	p_5 ($h+3v-2$)
p_6 ($h-2v-2$)	p_7 ($h-2v-2$)	p_8 ($h-2v-2$)	p_6 ($h-1v-2$)	p_7 ($h-1v-2$)	p_8 ($h-1v-2$)	p_6 ($h,v-2$)	p_7 ($h,v-2$)	p_8 ($h,v-2$)	p_6 ($h,v-2$)	p_7 ($h,v-2$)	p_8 ($h,v-2$)	p_6 ($h+1v-2$)	p_7 ($h+1v-2$)	p_8 ($h+1v-2$)	p_6 ($h+2v-2$)	p_7 ($h+2v-2$)	p_8 ($h+2v-2$)	p_6 ($h+3v-2$)	p_7 ($h+3v-2$)	p_8 ($h+3v-2$)
p_0 ($h-2v-1$)	p_1 ($h-2v-1$)	p_2 ($h-2v-1$)	p_0 ($h-1v-1$)	p_1 ($h-1v-1$)	p_2 ($h-1v-1$)	p_0 ($h-v-1$)	p_1 ($h-v-1$)	p_2 ($h-v-1$)	p_0 ($h,v-1$)	p_1 ($h,v-1$)	p_2 ($h,v-1$)	p_0 ($h+1v-1$)	p_1 ($h+1v-1$)	p_2 ($h+1v-1$)	p_0 ($h+2v-1$)	p_1 ($h+2v-1$)	p_2 ($h+2v-1$)	p_0 ($h+3v-1$)	p_1 ($h+3v-1$)	p_2 ($h+3v-1$)
p_3 ($h-2v-1$)	p_4 ($h-2v-1$)	p_5 ($h-2v-1$)	p_3 ($h-1v-1$)	p_4 ($h-1v-1$)	p_5 ($h-1v-1$)	p_3 ($h,v-1$)	p_4 ($h,v-1$)	p_5 ($h,v-1$)	p_3 ($h,v-1$)	p_4 ($h,v-1$)	p_5 ($h,v-1$)	p_3 ($h+1v-1$)	p_4 ($h+1v-1$)	p_5 ($h+1v-1$)	p_3 ($h+2v-1$)	p_4 ($h+2v-1$)	p_5 ($h+2v-1$)	p_3 ($h+3v-1$)	p_4 ($h+3v-1$)	p_5 ($h+3v-1$)
p_6 ($h-2v-1$)	p_7 ($h-2v-1$)	p_8 ($h-2v-1$)	p_6 ($h-1v-1$)	p_7 ($h-1v-1$)	p_8 ($h-1v-1$)	p_6 ($h,v-1$)	p_7 ($h,v-1$)	p_8 ($h,v-1$)	p_6 ($h,v-1$)	p_7 ($h,v-1$)	p_8 ($h,v-1$)	p_6 ($h+1v-1$)	p_7 ($h+1v-1$)	p_8 ($h+1v-1$)	p_6 ($h+2v-1$)	p_7 ($h+2v-1$)	p_8 ($h+2v-1$)	p_6 ($h+3v-1$)	p_7 ($h+3v-1$)	p_8 ($h+3v-1$)
p_0 ($h-2v$)	p_1 ($h-2v$)	p_2 ($h-2v$)	p_0 ($h-1v$)	p_1 ($h-1v$)	p_2 ($h-1v$)	p_0 (h,v)	p_1 (h,v)	p_2 (h,v)	p_0 (h,v)	p_1 (h,v)	p_2 (h,v)	p_0 ($h+1v$)	p_1 ($h+1v$)	p_2 ($h+1v$)	p_0 ($h+2v$)	p_1 ($h+2v$)	p_2 ($h+2v$)	p_0 ($h+3v$)	p_1 ($h+3v$)	p_2 ($h+3v$)
p_3 ($h-2v$)	p_4 ($h-2v$)	p_5 ($h-2v$)	p_3 ($h-1v$)	p_4 ($h-1v$)	p_5 ($h-1v$)	p_3 (h,v)	p_4 (h,v)	p_5 (h,v)	p_3 (h,v)	p_4 (h,v)	p_5 (h,v)	p_3 ($h+1v$)	p_4 ($h+1v$)	p_5 ($h+1v$)	p_3 ($h+2v$)	p_4 ($h+2v$)	p_5 ($h+2v$)	p_3 ($h+3v$)	p_4 ($h+3v$)	p_5 ($h+3v$)
p_6 ($h-2v$)	p_7 ($h-2v$)	p_8 ($h-2v$)	p_6 ($h-1v$)	p_7 ($h-1v$)	p_8 ($h-1v$)	p_6 (h,v)	p_7 (h,v)	p_8 (h,v)	p_6 (h,v)	p_7 (h,v)	p_8 (h,v)	p_6 ($h+1v$)	p_7 ($h+1v$)	p_8 ($h+1v$)	p_6 ($h+2v$)	p_7 ($h+2v$)	p_8 ($h+2v$)	p_6 ($h+3v$)	p_7 ($h+3v$)	p_8 ($h+3v$)

Pixels to be Encoded

Blue Pixel First

p_0 ($h-2v-2$)	p_1 ($h-2v-2$)	p_2 ($h-2v-2$)	p_0 ($h-1v-2$)	p_1 ($h-1v-2$)	p_2 ($h-1v-2$)	p_0 ($h,v-2$)	p_1 ($h,v-2$)	p_2 ($h,v-2$)	p_0 ($h+1v-2$)	p_1 ($h+1v-2$)	p_2 ($h+1v-2$)	p_0 ($h+2v-2$)	p_1 ($h+2v-2$)	p_2 ($h+2v-2$)	p_0 ($h+3v-2$)	p_1 ($h+3v-2$)	p_2 ($h+3v-2$)	p_0 ($h+3v-2$)	p_1 ($h+3v-2$)	p_2 ($h+3v-2$)
p_3 ($h-2v-2$)	p_4 ($h-2v-2$)	p_5 ($h-2v-2$)	p_3 ($h-1v-2$)	p_4 ($h-1v-2$)	p_5 ($h-1v-2$)	p_3 ($h,v-2$)	p_4 ($h,v-2$)	p_5 ($h,v-2$)	p_3 ($h,v-2$)	p_4 ($h,v-2$)	p_5 ($h,v-2$)	p_3 ($h+1v-2$)	p_4 ($h+1v-2$)	p_5 ($h+1v-2$)	p_3 ($h+2v-2$)	p_4 ($h+2v-2$)	p_5 ($h+2v-2$)	p_3 ($h+3v-2$)	p_4 ($h+3v-2$)	p_5 ($h+3v-2$)
p_6 ($h-2v-2$)	p_7 ($h-2v-2$)	p_8 ($h-2v-2$)	p_6 ($h-1v-2$)	p_7 ($h-1v-2$)	p_8 ($h-1v-2$)	p_6 ($h,v-2$)	p_7 ($h,v-2$)	p_8 ($h,v-2$)	p_6 ($h,v-2$)	p_7 ($h,v-2$)	p_8 ($h,v-2$)	p_6 ($h+1v-2$)	p_7 ($h+1v-2$)	p_8 ($h+1v-2$)	p_6 ($h+2v-2$)	p_7 ($h+2v-2$)	p_8 ($h+2v-2$)	p_6 ($h+3v-2$)	p_7 ($h+3v-2$)	p_8 ($h+3v-2$)
p_0 ($h-2v-1$)	p_1 ($h-2v-1$)	p_2 ($h-2v-1$)	p_0 ($h-1v-1$)	p_1 ($h-1v-1$)	p_2 ($h-1v-1$)	p_0 ($h,v-1$)	p_1 ($h,v-1$)	p_2 ($h,v-1$)	p_0 ($h,v-1$)	p_1 ($h,v-1$)	p_2 ($h,v-1$)	p_0 ($h+1v-1$)	p_1 ($h+1v-1$)	p_2 ($h+1v-1$)	p_0 ($h+2v-1$)	p_1 ($h+2v-1$)	p_2 ($h+2v-1$)	p_0 ($h+3v-1$)	p_1 ($h+3v-1$)	p_2 ($h+3v-1$)
p_3 ($h-2v-1$)	p_4 ($h-2v-1$)	p_5 ($h-2v-1$)	p_3 ($h-1v-1$)	p_4 ($h-1v-1$)	p_5 ($h-1v-1$)	p_3 ($h,v-1$)	p_4 ($h,v-1$)	p_5 ($h,v-1$)	p_3 ($h,v-1$)	p_4 ($h,v-1$)	p_5 ($h,v-1$)	p_3 ($h+1v-1$)	p_4 ($h+1v-1$)	p_5 ($h+1v-1$)	p_3 ($h+2v-1$)	p_4 ($h+2v-1$)	p_5 ($h+2v-1$)	p_3 ($h+3v-1$)	p_4 ($h+3v-1$)	p_5 ($h+3v-1$)
p_6 ($h-2v-1$)	p_7 ($h-2v-1$)	p_8 ($h-2v-1$)	p_6 ($h-1v-1$)	p_7 ($h-1v-1$)	p_8 ($h-1v-1$)	p_6 ($h,v-1$)	p_7 ($h,v-1$)	p_8 ($h,v-1$)	p_6 ($h,v-1$)	p_7 ($h,v-1$)	p_8 ($h,v-1$)	p_6 ($h+1v-1$)	p_7 ($h+1v-1$)	p_8 ($h+1v-1$)	p_6 ($h+2v-1$)	p_7 ($h+2v-1$)	p_8 ($h+2v-1$)	p_6 ($h+3v-1$)	p_7 ($h+3v-1$)	p_8 ($h+3v-1$)
p_0 ($h-2v$)	p_1 ($h-2v$)	p_2 ($h-2v$)	p_0 ($h-1v$)	p_1 ($h-1v$)	p_2 ($h-1v$)	p_0 (h,v)	p_1 (h,v)	p_2 (h,v)	p_0 (h,v)	p_1 (h,v)	p_2 (h,v)	p_0 ($h+1v$)	p_1 ($h+1v$)	p_2 ($h+1v$)	p_0 ($h+2v$)	p_1 ($h+2v$)	p_2 ($h+2v$)	p_0 ($h+3v$)	p_1 ($h+3v$)	p_2 ($h+3v$)
p_3 ($h-2v$)	p_4 ($h-2v$)	p_5 ($h-2v$)	p_3 ($h-1v$)	p_4 ($h-1v$)	p_5 ($h-1v$)	p_3 (h,v)	p_4 (h,v)	p_5 (h,v)	p_3 (h,v)	p_4 (h,v)	p_5 (h,v)	p_3 ($h+1v$)	p_4 ($h+1v$)	p_5 ($h+1v$)	p_3 ($h+2v$)	p_4 ($h+2v$)	p_5 ($h+2v$)	p_3 ($h+3v$)	p_4 ($h+3v$)	p_5 ($h+3v$)
p_6 ($h-2v$)	p_7 ($h-2v$)	p_8 ($h-2v$)	p_6 ($h-1v$)	p_7 ($h-1v$)	p_8 ($h-1v$)	p_6 (h,v)	p_7 (h,v)	p_8 (h,v)	p_6 (h,v)	p_7 (h,v)	p_8 (h,v)	p_6 ($h+1v$)	p_7 ($h+1v$)	p_8 ($h+1v$)	p_6 ($h+2v$)	p_7 ($h+2v$)	p_8 ($h+2v$)	p_6 ($h+3v$)	p_7 ($h+3v$)	p_8 ($h+3v$)

Pixels to be Encoded

Not Shown: Red Pixel First

Figure 287 Reference Pixel Relative Coordinate Indexes (Nona-Cell)

5786

For a Nona-Cell image sensor, the eighteen highlighted pixels in the bottom three rows will be encoded. This corresponds to Nona-Cell coordinates $p(h, v)$ (i.e., the current Nona-Cell) and $p(h+1, v)$. The other colored squares represent the previously decoded pixels, which are available to be used as reference data.

I.3.1.1 User Control Parameters for Boundary Fill

5787

At the boundary of the image frame no neighboring reference pixels exist. As a result, difference calculations for pixels at the image boundary cannot depend upon the values of neighbor pixels. To handle this case, MPC allows the boundary to be filled with user control parameters. This feature makes prediction

I.3.2 Basic Prediction Mode (Nona-Cell)

For Nona-Cell image sensors, Basic Prediction Mode:

- Compresses the image using a simple algorithm to handle image details oriented in a specific direction, or basic cases of differences between neighboring pixels.
- Supports Compression Modes AD, DGD, and FNR as defined below.

I.3.2.1 Standard Mode 0: AD (Average-based Directional Differential Mode)

Compression Mode AD is efficient for compressing pixels located on horizontal and vertical edges. An AD packet transmits one prediction pixel and three reference pixels. The prediction pixel is either the combination of the three reference pixels, or the reference pixel itself.

Figure 288 defines the packet structure for Compression Mode AD (Nona-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 0, and the high bit of Header field **Mode IDX** shall contain 0.

The Payload shall contain compressed data for the nine pixels of a Nona-Cell:

- Field **Qstep** shall indicate the quantization step size.
- Field **ByrDiff** shall contain the quantized difference between a prediction and the average of the nine pixels, and is also used for generation of the Bayer image (see *Section I.3.4*).
- Fields **MltDiff0** through **MltDiff8** shall contain the encoded data between the averaged pixel and each Multi-Pixel.

Table 91 shows pseudo-code for decoding an MPC Packet for Compression Mode AD (Nona-Cell).

Mode	Header		Payload										
	Prediction IDX	Mode IDX											
AD	0	0	Qstep[1:0]	ByrDiff[4:0]	MltDiff0[3:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[3:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[3:0]	MltDiff7[3:0]	MltDiff8[3:0]

Figure 288 MPC Packet Structure for Compression Mode AD (Nona-Cell)

5813

Note:

5814

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color.

5815

5816

5817

5818

Note also that this pseudo-code only describes 3x3 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

Table 91 Decoder for Compression Mode AD (Nona-Cell)

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 1 : (Qstep==1) ? 2 : (Qstep==2) ? 3 : 5
3: shift1 = (Qstep==0) ? 0 : (Qstep==1) ? 1 : (Qstep==2) ? 2 : 3
4: // For decoding the pixel  $p_k(h, v)$ ,  $k=\{0, 1, \dots, 8\}$ 
5: if (Line count < 6) // Vertical line counter, Boundary condition
6:     q =  $p_2(h-2, v)$ 
7: else if (unit count == 0) // unit count is for processing unit block, boundary condition
8:     q =  $p_6(h, v-2)$ 
9: else //Prediction value q of the preview pixel
10:    if  $p_8(h-2, v-2) \geq \max\{p_2(h, v-2), p_6(h-2, v)\}$ 
11:        q =  $\min\{p_2(h, v-2), p_6(h-2, v)\}$ 
12:    else if  $p_8(h-2, v-2) \leq \min\{p_2(h, v-2), p_6(h-2, v)\}$ 
13:        q =  $\max\{p_2(h, v-2), p_6(h-2, v)\}$ 
14:    else
15:        q =  $p_2(h, v-2) + p_6(h-2, v) - p_8(h-2, v-2)$ 
16:    end if
17: end if
18:  $p_{byr}(h, v) = ((q >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$  // Bayer pixel  $p_{byr}$ 
19:  $p_0(h, v) = ((p_{byr} >> shift1 + MltDiff0) << shift1) + (1 << (shift1-1))$ 
20:  $p_1(h, v) = ((p_{byr} >> shift1 + MltDiff1) << shift1) + (1 << (shift1-1))$ 
21:  $p_2(h, v) = ((p_{byr} >> shift1 + MltDiff2) << shift1) + (1 << (shift1-1))$ 
22:  $p_3(h, v) = ((p_{byr} >> shift1 + MltDiff3) << shift1) + (1 << (shift1-1))$ 
23:  $p_4(h, v) = ((p_{byr} >> shift1 + MltDiff4) << shift1) + (1 << (shift1-1))$ 
24:  $p_5(h, v) = ((p_{byr} >> shift1 + MltDiff5) << shift1) + (1 << (shift1-1))$ 
25:  $p_6(h, v) = ((p_{byr} >> shift1 + MltDiff6) << shift1) + (1 << (shift1-1))$ 
26:  $p_7(h, v) = ((p_{byr} >> shift1 + MltDiff7) << shift1) + (1 << (shift1-1))$ 
27:  $p_8(h, v) = ((p_{byr} >> shift1 + MltDiff8) << shift1) + (1 << (shift1-1))$ 

```

I.3.2.2 Standard Mode 1: DGD (Diagonal Direction-based Differential Mode)

Compression Mode DGD references pixels that are located along diagonal directions.

Figure 289 defines the packet structure for Compression Mode DGD (Nona-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 0, and Header field **Mode IDX** shall contain 3'b100.

The Payload shall contain compressed data for the nine pixels of a Nona-Cell:

- Field **Qstep** shall indicate the quantization step size.
- Fields **ByrDiff** and **MltDiff1** through **MltDiff8** shall contain the encoded data between the predicted pixel and each original pixel.

Field **ByrDiff** can also be regarded as **MltDiff0**; it is used for generation of the Bayer image, see **Section I.3.4**.

Table 92 shows pseudo-code for decoding an MPC Packet for Compression Mode DGD (Nona-Cell).

Mode	Header		Payload									
	Prediction IDX	Mode IDX	Qstep[1:0]		ByrDiff[4:0]	MltDiff1[4:0]	MltDiff2[5:0]	MltDiff3[4:0]	MltDiff4[5:0]	MltDiff5[2:0]	MltDiff6[2:0]	MltDiff7[2:0]
DGD	0	1 0 0	Qstep[1:0]	ByrDiff[4:0]	MltDiff1[4:0]	MltDiff2[5:0]	MltDiff3[4:0]	MltDiff4[5:0]	MltDiff5[2:0]	MltDiff6[2:0]	MltDiff7[2:0]	MltDiff8[2:0]

Figure 289 MPC Packet Structure for Compression Mode DGD (Nona-Cell)

5830

Note:

5831

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color. The Red- or Blue-first case is described in comments.

5832

5833

Note also that this pseudo-code only describes 3x3 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

5834
5835

Table 92 Decoder for Compression Mode DGD (Nona-Cell)

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 2 : (Qstep==1) ? 3 : (Qstep==2) ? 4 : 5
3: // direction for pk(h,v)
4: ldiff0 = abs(p3(h-1,v-1) - p7(h-1,v-1)) // color pixel first: abs(p3(h,v-1) - p7(h,v-1))
5: ldiff1 = abs(p4(h-1,v-1) - p8(h-1,v-1)) // color pixel first: abs(p4(h,v-1) - p8(h,v-1))
6: ldiff2 = abs(p3(h+1,v-1) - p7(h+1,v-1)) // color pixel first: abs(p3(h,v-1) - p7(h,v-1))
7: ldiff3 = abs(p4(h+1,v-1) - p8(h+1,v-1)) // color pixel first: abs(p4(h,v-1) - p8(h,v-1))
8: rdiff0 = abs(p4(h-1,v-1) - p6(h-1,v-1)) // color pixel first: abs(p4(h+2,v-1) - p6(h+2,v-1))
9: rdiff1 = abs(p5(h-1,v-1) - p7(h-1,v-1)) // color pixel first: abs(p5(h+2,v-1) - p7(h+2,v-1))
10: rdiff2 = abs(p4(h+1,v-1) - p6(h+1,v-1)) // color pixel first: abs(p4(h+2,v-1) - p6(h+2,v-1))
11: rdiff3 = abs(p5(h+1,v-1) - p7(h+1,v-1)) // color pixel first: abs(p5(h+2,v-1) - p7(h+2,v-1))
12: ldiff = ldiff0 + ldiff1 + ldiff2 + ldiff3
13: rdiff = rdiff0 + rdiff1 + rdiff2 + rdiff3
14: left_en = (ldiff < rdiff) ? 1 : 0
15: // For decoding the green pixel pk(h,v), k={0,1,..,8}
16: if left_en == 1
17:     // Prediction pixel
18:     q0(h,v) = p8(h-1,v-1)           // color pixel first: q0(h,v) = p8(h-2,v-2)
19:     q1(h,v) = p5(h-1,v-1)           // color pixel first: q1(h,v) = p5(h-2,v-2)
20:     q2(h,v) = p2(h-1,v-1)           // color pixel first: q2(h,v) = p2(h-2,v-2)
21:     q3(h,v) = p7(h-1,v-1)           // color pixel first: q3(h,v) = p7(h-2,v-2)
22:     q6(h,v) = p6(h-1,v-1)           // color pixel first: q6(h,v) = p6(h-2,v-2)
23:     // For decoding the pixel
24:     p0(h,v) = ((q0(h,v) >> shift0 + ByrDiff) << shift0) + (1 << (shift0 -1))
25:     p1(h,v) = ((q1(h,v) >> shift0 + MltPixel1) << shift0) + (1 << (shift0 -1))
26:     p2(h,v) = ((q2(h,v) >> shift0 + MltPixel2) << shift0) + (1 << (shift0 -1))
27:     p3(h,v) = ((q3(h,v) >> shift0 + MltPixel3) << shift0) + (1 << (shift0 -1))
28:     p6(h,v) = ((q6(h,v) >> shift0 + MltPixel4) << shift0) + (1 << (shift0 -1))
29:     // Prediction pixel
30:     q4(h,v) = p0(h,v)           // color pixel first: q4(h,v) = p0(h,v)
31:     q5(h,v) = p1(h,v)           // color pixel first: q5(h,v) = p1(h,v)
32:     q7(h,v) = p3(h,v)           // color pixel first: q7(h,v) = p3(h,v)
33:     // For decoding the pixel
34:     p4(h,v) = ((q4(h,v) >> shift0 + MltPixel5) << shift0) + (1 << (shift0 -1))
35:     p5(h,v) = ((q5(h,v) >> shift0 + MltPixel6) << shift0) + (1 << (shift0 -1))
36:     p7(h,v) = ((q7(h,v) >> shift0 + MltPixel7) << shift0) + (1 << (shift0 -1))

```

```

37:      // Prediction pixel
38:       $q_8(h,v) = p_4(h,v)$           // color pixel first:  $q_8(h,v) = p_4(h,v)$ 
39:       $p_8(h,v) = ((q_8(h,v) >> shift0 + MltPixel18) << shift0) + (1 << (shift0 - 1))$ 
40:  else // Right diagonal direction
41:      // Prediction pixel
42:       $q_2(h,v) = p_6(h+1,v-1)$     // color pixel first:  $q_2(h,v) = p_6(h+2,v-2)$ 
43:       $q_1(h,v) = p_3(h+1,v-1)$     // color pixel first:  $q_1(h,v) = p_3(h+2,v-2)$ 
44:       $q_0(h,v) = p_0(h+1,v-1)$     // color pixel first:  $q_0(h,v) = p_0(h+2,v-2)$ 
45:       $q_5(h,v) = p_7(h+1,v-1)$     // color pixel first:  $q_5(h,v) = p_7(h+2,v-2)$ 
46:       $q_8(h,v) = p_8(h+1,v-1)$     // color pixel first:  $q_8(h,v) = p_8(h+2,v-2)$ 
47:  // For decoding the pixel
48:       $p_2(h,v) = ((q_2(h,v) >> shift0 + ByrDiff) << shift0) + (1 << (shift0 - 1))$ 
49:       $p_1(h,v) = ((q_1(h,v) >> shift0 + MltPixel11) << shift0) + (1 << (shift0 - 1))$ 
50:       $p_0(h,v) = ((q_0(h,v) >> shift0 + MltPixel12) << shift0) + (1 << (shift0 - 1))$ 
51:       $p_5(h,v) = ((q_5(h,v) >> shift0 + MltPixel13) << shift0) + (1 << (shift0 - 1))$ 
52:       $p_8(h,v) = ((q_8(h,v) >> shift0 + MltPixel14) << shift0) + (1 << (shift0 - 1))$ 
53:  // Prediction pixel
54:       $q_3(h,v) = p_1(h,v)$         // color pixel first:  $q_3(h,v) = p_1(h,v)$ 
55:       $q_4(h,v) = p_2(h,v)$         // color pixel first:  $q_4(h,v) = p_2(h,v)$ 
56:       $q_7(h,v) = p_5(h,v)$         // color pixel first:  $q_7(h,v) = p_5(h,v)$ 
57:  // For decoding the pixel
58:       $p_3(h,v) = ((q_3(h,v) >> shift0 + MltPixel16) << shift0) + (1 << (shift0 - 1))$ 
59:       $p_4(h,v) = ((q_4(h,v) >> shift0 + MltPixel15) << shift0) + (1 << (shift0 - 1))$ 
60:       $p_7(h,v) = ((q_7(h,v) >> shift0 + MltPixel17) << shift0) + (1 << (shift0 - 1))$ 
61:  // Prediction pixel
62:       $q_6(h,v) = p_4(h,v)$ 
63:       $p_6(h,v) = ((q_6(h,v) >> shift0 + MltPixel18) << shift0) + (1 << (shift0 - 1))$ 
64:  //  $p_1(h,v), \dots, p_8(h,v)$  should be  $p_1(h,v), \dots, p_8(h,v) = p_0(h,v)$ , when you want to get the  $p_{byr}(h,v)$ 
65:       $p_{byr}(h,v) = p_0(h,v)$       // Bayer green pixel  $p_{byr}(h,v)$ 

```

I.3.2.3 Standard Mode 2: FNR (Fixed Quantization and No-Reference Mode)

If the difference between the current pixel and reference pixel is large, then the pixel should be compressed by fixed quantization. In Compression Mode FNR the data is compressed using only quantization, with no difference operation.

Figure 290 defines the packet structure for Compression Mode FNR (Nona-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 0, and Header field **Mode IDX** shall contain 3'b101.

The Payload shall contain compressed data for the nine pixels of a Nona-Cell:

- Fields **ByrPixel** and **MltPixel1** through **MltPixel8** shall contain the quantization results of each Multi-Pixel.

Field **ByrPixel** can also be regarded as **MltPixel0**; it is used for generation of the Bayer image, see **Section I.3.4**.

Table 93 shows pseudo-code for decoding an MPC Packet for Compression Mode PD.

Mode	Header		Payload								
	Prediction IDX	Mode IDX	ByrPixel[4:0]	MltPixel1[3:0]	MltPixel2[4:0]	MltPixel3[3:0]	MltPixel4[4:0]	MltPixel5[3:0]	MltPixel6[4:0]	MltPixel7[3:0]	MltPixel8[4:0]
FNR	0	1 0 1									

Figure 290 MPC Packet Structure for Compression Mode FNR (Nona-Cell)

Note:

In this pseudo-code, any color pixel can appear first.

Note also that this pseudo-code only describes 3x3 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

Table 93 Decoder for Compression Mode FNR (Nona-Cell)

1:	// Quantization step
2:	shift0 = 5
3:	shift1 = 6
4:	// For decoding the pixel $p_k(h, v)$, $k=\{0, 1, \dots, 8\}$
5:	$p_0(h, v) = (\text{ByrPixel} << \text{shift0}) + (1 << (\text{shift0} - 1))$
6:	$p_1(h, v) = (\text{MltPixel1} << \text{shift1}) + (1 << (\text{shift0} - 1))$
7:	$p_2(h, v) = (\text{MltPixel2} << \text{shift0}) + (1 << (\text{shift0} - 1))$
8:	$p_3(h, v) = (\text{MltPixel3} << \text{shift1}) + (1 << (\text{shift0} - 1))$
9:	$p_4(h, v) = (\text{MltPixel4} << \text{shift0}) + (1 << (\text{shift0} - 1))$
10:	$p_5(h, v) = (\text{MltPixel5} << \text{shift1}) + (1 << (\text{shift0} - 1))$
11:	$p_6(h, v) = (\text{MltPixel6} << \text{shift0}) + (1 << (\text{shift0} - 1))$
12:	$p_7(h, v) = (\text{MltPixel7} << \text{shift1}) + (1 << (\text{shift0} - 1))$
13:	$p_8(h, v) = (\text{MltPixel8} << \text{shift0}) + (1 << (\text{shift0} - 1))$
14:	// $p_1(h, v), \dots, p_8(h, v)$ should be $p_1(h, v), \dots, p_8(h, v) = p_0(h, v)$, when you want to get the $p_{byr}(h, v)$
15:	$p_{byr}(h, v) = p_0(h, v)$ // Bayer pixel $p_{byr}(h, v)$

I.3.3 Advanced Prediction Mode (Nona-Cell)

For Nona-Cell image sensors, Advanced Prediction Mode:

- Compresses the image using more extensive direction-aware algorithms to minimize image quality loss.
- Supports Compression Modes eAD, eMPD, eHVD, eHVI, and eSHV as defined below.

I.3.3.1 Extended Mode 0: eAD (extended Average-based Directional Differential Mode)

Compression Mode eAD extends Standard Compression Mode AD (which uses Basic Prediction Mode) by increasing the number of quantization levels.

Figure 291 defines the packet structure for Compression Mode eAD (Nona-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 1, and the high two bits of Header field **Mode IDX** shall contain 2'b00.

The Payload shall contain compressed data for the nine pixels of a Nona-Cell:

- 1-bit field **Qstep** shall indicate the quantization step size.
- Field **ByrDiff** shall contain the quantized difference between a prediction and the average of the nine pixels, and is also used for generation of the Bayer image (see *Section I.3.4*).
- Fields **MltDiff0** through **MltDiff8** shall contain the quantized differences between the average and each Multi-Pixel.

Table 94 shows pseudo-code for decoding an MPC Packet for Compression Mode eAD (Nona-Cell).

Mode	Header			Payload									
	Prediction IDX	Mode IDX	Qstep	ByrDiff[4:0]	MltDiff0[3:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[3:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[3:0]	MltDiff7[3:0]	MltDiff8[3:0]
eAD	1	0	0	ByrDiff[4:0]	MltDiff0[3:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[3:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[3:0]	MltDiff7[3:0]	MltDiff8[3:0]

Figure 291 MPC Packet Structure for Compression Mode eAD (Nona-Cell)

5869

Note:

5870

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color. The Red- or Blue-first case is described in comments.

5871

5872

Note also that this pseudo-code only describes 3x3 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

5873

5874

Table 94 Decoder for Compression Mode eAD (Nona-Cell)

```

1: // Quantization step
2: shift0 = 5
3: shift1 = (Qstep==0) ? 4 : 5
4: // For decoding the pixel  $p_k(h, v)$ ,  $k=\{0, 1, \dots, 8\}$ 
5: if (Line count < 6) // Vertical line counter, Boundary condition
6:   q =  $p_2(h-2, v)$ 
7: else if (unit count == 0) // unit count is for processing unit block, boundary condition
8:   q =  $p_6(h, v-2)$ 
9: else //Prediction value q of the preview pixel
10:  if  $p_8(h-2, v-2) \geq \max\{p_2(h, v-2), p_6(h-2, v)\}$ 
11:    q =  $\min\{p_2(h, v-2), p_6(h-2, v)\}$ 
12:  else if  $p_8(h-2, v-2) \leq \min\{p_2(h, v-2), p_6(h-2, v)\}$ 
13:    q =  $\max\{p_2(h, v-2), p_6(h-2, v)\}$ 
14:  else
15:    q =  $p_2(h, v-2) + p_6(h-2, v) - p_8(h-2, v-2)$ 
16:  end if
17: end if
18: //  $p_1(h, v), \dots, p_8(h, v)$  should be  $p_1(h, v), \dots, p_8(h, v) = p_0(h, v)$ , when you want to get the  $p_{byr}(h, v)$ 
19:  $p_{byr}(h, v) = ((q >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$  // Bayer pixel  $p_{byr}$ 
20:  $p_0(h, v) = ((p_{byr} >> shift1 + MltDiff0) << shift1) + (1 << (shift1-1))$ 
21:  $p_1(h, v) = ((p_{byr} >> shift1 + MltDiff1) << shift1) + (1 << (shift1-1))$ 
22:  $p_2(h, v) = ((p_{byr} >> shift1 + MltDiff2) << shift1) + (1 << (shift1-1))$ 
23:  $p_3(h, v) = ((p_{byr} >> shift1 + MltDiff3) << shift1) + (1 << (shift1-1))$ 
24:  $p_4(h, v) = ((p_{byr} >> shift1 + MltDiff4) << shift1) + (1 << (shift1-1))$ 
25:  $p_5(h, v) = ((p_{byr} >> shift1 + MltDiff5) << shift1) + (1 << (shift1-1))$ 
26:  $p_6(h, v) = ((p_{byr} >> shift1 + MltDiff6) << shift1) + (1 << (shift1-1))$ 
27:  $p_7(h, v) = ((p_{byr} >> shift1 + MltDiff7) << shift1) + (1 << (shift1-1))$ 
28:  $p_8(h, v) = ((p_{byr} >> shift1 + MltDiff8) << shift1) + (1 << (shift1-1))$ 

```

I.3.3.2 Extended Mode 1: eHVD (extended Horizontal or Vertical Direction-based Differential Mode)

Compression Mode eHVD references pixels that are located along the horizontal or vertical direction.

Figure 292 defines the packet structure for Compression Mode eHVD (Nona-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 1, and the high two bits of Header field **Mode IDX** shall contain 2'b01.

The Payload shall contain compressed data for the nine pixels of a Nona-Cell:

- 1-bit field **Dir** shall indicate the reference direction:

If **Dir** contains 0 then the reference direction is the vertical direction

If **Dir** contains 1 then the reference direction is the horizontal direction

- Field **Qstep** shall indicate the quantization step size.
- Fields **ByrDiff** and **MltDiff1** through **MltDiff8** shall contain the quantized differences between the predicted value and each pixel.

Field **ByrPixel** can also be regarded as **MltDiff0**; it is used for generation of the Bayer image, see **Section I.3.4**.

Table 95 shows pseudo-code for decoding an MPC Packet for Compression Mode eHVD (Nona-Cell).

Mode	Header			Payload										
	Prediction IDX	Mode IDX												
eHVD	1	0	1	Dir	Qstep[1:0]	ByrDiff[4:0]	MltDiff1[4:0]	MltDiff2[4:0]	MltDiff3[3:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[3:0]	MltDiff7[3:0]	MltDiff8[3:0]

Figure 292 MPC Packet Structure for Compression Mode eHVD (Nona-Cell)

5890

Note:

5891

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color. The Red- or Blue-first case is described in comments.

5892

Note also that this pseudo-code only describes 3x3 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

5893

5894

5895

Table 95 Decoder for Compression Mode eHVD (Nona-Cell)

```

1: // Quantization step
2: shift0 = (Qstep==1) ? 1 : (Qstep==2) ? 2 : (Qstep==3) ? 3 : 4
3: // For decoding the pixel  $p_i(h, v)$ ,  $i=\{0,1,2,3,4,5,6,7,8\}$ 
4: if Dir == 0 // Vertical direction
5:    $q_0(h, v) = p_6(h, v-2)$  //Prediction value q of the preview pixel
6:    $q_1(h, v) = p_7(h, v-2)$  //Prediction value q of the preview pixel
7:    $q_2(h, v) = p_8(h, v-2)$  //Prediction value q of the preview pixel
8:    $p_0(h, v) = ((q_0(h, v) >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
9:    $p_1(h, v) = ((q_1(h, v) >> shift0 + MltDiff1) << shift1) + (1 << (shift0-1))$ 
10:   $p_2(h, v) = ((q_2(h, v) >> shift0 + MltDiff2) << shift1) + (1 << (shift0-1))$ 
11:   $q_3(h, v) = p_0(h, v)$  //Prediction value q of the preview pixel
12:   $q_4(h, v) = p_1(h, v)$  //Prediction value q of the preview pixel
13:   $q_5(h, v) = p_2(h, v)$  //Prediction value q of the preview pixel
14:   $p_3(h, v) = ((q_3(h, v) >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
15:   $p_4(h, v) = ((q_4(h, v) >> shift0 + MltDiff4) << shift0) + (1 << (shift0-1))$ 
16:   $p_5(h, v) = ((q_5(h, v) >> shift0 + MltDiff5) << shift0) + (1 << (shift0-1))$ 
17:   $q_6(h, v) = p_3(h, v)$  //Prediction value q of the preview pixel
18:   $q_7(h, v) = p_4(h, v)$  //Prediction value q of the preview pixel
19:   $q_8(h, v) = p_5(h, v)$  //Prediction value q of the preview pixel
20:   $p_6(h, v) = ((q_6(h, v) >> shift0 + MltDiff6) << shift0) + (1 << (shift0-1))$ 
21:   $p_7(h, v) = ((q_7(h, v) >> shift0 + MltDiff7) << shift0) + (1 << (shift0-1))$ 
22:   $p_8(h, v) = ((q_8(h, v) >> shift0 + MltDiff8) << shift0) + (1 << (shift0-1))$ 
23: else
24:    $q_0(h, v) = p_2(h-2, v)$  //Prediction value q of the preview pixel
25:    $q_3(h, v) = p_5(h-2, v)$  //Prediction value q of the preview pixel
26:    $q_6(h, v) = p_8(h-2, v)$  //Prediction value q of the preview pixel
27:    $p_0(h, v) = ((q_0(h, v) >> shift0 + ByrDiff) << shift0) + (1 << (shift0-1))$ 
28:    $p_3(h, v) = ((q_3(h, v) >> shift0 + MltDiff1) << shift0) + (1 << (shift0-1))$ 
29:    $p_6(h, v) = ((q_6(h, v) >> shift0 + MltDiff2) << shift0) + (1 << (shift0-1))$ 
30:    $q_1(h, v) = p_0(h, v)$  //Prediction value q of the preview pixel
31:    $q_4(h, v) = p_3(h, v)$  //Prediction value q of the preview pixel
32:    $q_7(h, v) = p_6(h, v)$  //Prediction value q of the preview pixel
33:    $p_1(h, v) = ((q_1(h, v) >> shift0 + MltDiff3) << shift0) + (1 << (shift0-1))$ 
34:    $p_4(h, v) = ((q_4(h, v) >> shift0 + MltDiff4) << shift0) + (1 << (shift0-1))$ 
35:    $p_5(h, v) = ((q_7(h, v) >> shift0 + MltDiff5) << shift0) + (1 << (shift0-1))$ 
36:    $q_2(h, v) = p_1(h, v)$  //Prediction value q of the preview pixel

```

```
37:      $q_5(h,v) = p_4(h,v)$           //Prediction value  $q$  of the preview pixel
38:      $q_8(h,v) = p_7(h,v)$           //Prediction value  $q$  of the preview pixel
39:      $p_2(h,v) = ((q_2(h,v) >> shift0 + MltDiff6) << shift0) + (1 << (shift0-1))$ 
40:      $p_5(h,v) = ((q_5(h,v) >> shift0 + MltDiff7) << shift0) + (1 << (shift0-1))$ 
41:      $p_8(h,v) = ((q_8(h,v) >> shift0 + MltDiff8) << shift0) + (1 << (shift0-1))$ 
42:     //  $p_1(h,v), \dots, p_8(h,v)$  should be  $p_1(h,v), \dots, p_8(h,v) = p_0(h,v)$ , when you want to get the  $p_{byr}(h,v)$ 
43:      $p_{byr}(h,v) = p_0(h,v)$           // Bayer pixel  $p_{byr}$ 
```

I.3.3.3 Extended Mode 2: eHVI (extended Horizontal or Vertical Average-based Intra-Cell Differential Mode)

Compression Mode eHVI takes the horizontal or vertical difference from intra-cell pixels, not from inter-cell pixels.

Figure 294 defines the packet structure for Compression Mode eHVI. For this Compression Mode, Header field **Prediction IDX** shall contain 1, and the high two bits of Header field **Mode IDX** shall contain 2'b10.

The Payload shall contain compressed data for the nine pixels of a Nona-Cell, organized as three 3-pixel triplets (see **Figure 278**):

- 1-bit field **Dir** shall indicate the compression orientation for this MPC Packet:

If Dir contains 0: The orientation shall be vertical, and each decoded pixel triplet shall be 1x3 pixels. The first pixel triplet shall be $p_0(h, v)$, $p_3(h, v)$, and $p_6(h, v)$; the second pixel triplet shall be $p_1(h, v)$, $p_4(h, v)$, and $p_7(h, v)$; and the third pixel triplet shall be $p_2(h, v)$, $p_5(h, v)$, and $p_8(h, v)$.

If Dir contains 1: The orientation shall be horizontal, and each decoded pixel triplet shall be 3x1 pixels. The first pixel triplet shall be $p_0(h, v)$, $p_1(h, v)$, and $p_2(h, v)$; the second pixel triplet shall be $p_3(h, v)$, $p_4(h, v)$, and $p_5(h, v)$; and the third pixel triplet shall be $p_6(h, v)$, $p_7(h, v)$, and $p_8(h, v)$.

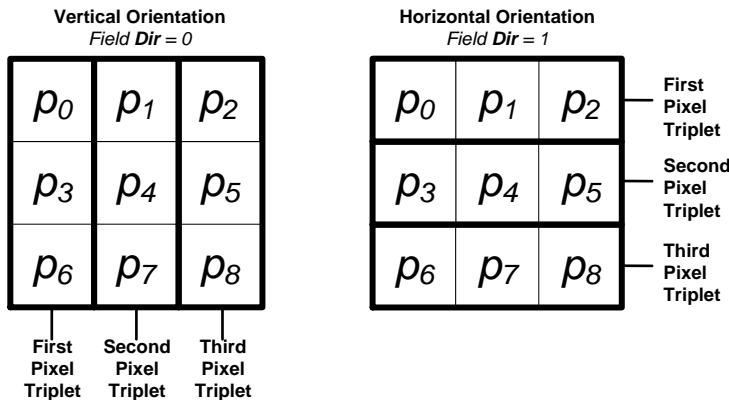


Figure 293 Pixel Triplet Definitions for eHVI Mode

- Field **Qstep** shall indicate the quantization step size.
- Fields **ByrDiff**, **MltDiff1**, and **MltDiff2** shall contain the fixed quantization result of the center pixel of the first pixel triplet, the center pixel of the second pixel triplet, and the center pixel of the third pixel triplet, respectively; no difference operation is involved.

If field Dir indicates vertical orientation (value 0), then:

- Field **ByrDiff** shall contain the fixed quantization result of $p_3(h, v)$
- Field **MltDiff1** shall contain the fixed quantization result of $p_4(h, v)$
- Field **MltDiff2** shall contain the fixed quantization result of $p_5(h, v)$

If field Dir indicates horizontal orientation (value 1), then:

- Field **ByrDiff** shall contain the fixed quantization result of $p_1(h, v)$
- Field **MltDiff1** shall contain the fixed quantization result of $p_4(h, v)$
- Field **MltDiff2** shall contain the fixed quantization result of $p_7(h, v)$

- Fields **MltDiff3** and **MltDiff4** shall contain the quantized differences with the predicted value of **ByrDiff** in the first pixel triplet. Fields **MltDiff5** and **MltDiff6** shall contain the quantized differences with the predicted value of **MltDiff1** in the second pixel triplet. Fields **MltDiff7** and **MltDiff8** shall contain the quantized differences with the predicted value of **MltDiff2** in the third pixel triplet.

Field **ByrDiff** can also be regarded as **MltDiff0**; it is used for generation of the Bayer image, see **Section I.3.4**.

5928

Table 96 shows pseudo-code for decoding an MPC Packet for Compression Mode eHVI.

5929

Mode	Header			Payload									
	Prediction IDX	Mode IDX	Dir	2step[1:0]	ByrDiff[6:0]	MltDiff1[6:0]	MltDiff2[6:0]	MltDiff3[2:0]	MltDiff4[2:0]	MltDiff5[2:0]	MltDiff6[2:0]	MltDiff7[2:0]	MltDiff8[2:0]
eHVI	1	1	0										

Figure 294 MPC Packet Structure for Compression Mode eHVI

5930

Note:

5931
5932
5933

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color.

5934
5935

Note also that this pseudo-code only describes 3x3 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

5936

Table 96 Decoder for Compression Mode eHVI

1:	// Quantization step
2:	shift0 = (<i>Qstep</i> == 0) ? 1 : (<i>Qstep</i> == 1) ? 2 : (<i>Qstep</i> == 2) ? 3 : 4
3:	if <i>Dir</i> == 1 // Horizontal direction
4:	$p_1(h, v) = (\text{ByrDiff} << 3) + (1 << 2)$
5:	$p_4(h, v) = (\text{MltDiff1} << 3) + (1 << 2)$
6:	$p_7(h, v) = (\text{MltDiff2} << 3) + (1 << 2)$
7:	$p_0(h, v) = ((p_1(h, v) >> \text{shift0} + \text{MltDiff3}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
8:	$p_2(h, v) = ((p_1(h, v) >> \text{shift0} + \text{MltDiff4}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
9:	$p_3(h, v) = ((p_4(h, v) >> \text{shift0} + \text{MltDiff5}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
10:	$p_5(h, v) = ((p_4(h, v) >> \text{shift0} + \text{MltDiff6}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
11:	$p_6(h, v) = ((p_7(h, v) >> \text{shift0} + \text{MltDiff7}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
12:	$p_8(h, v) = ((p_7(h, v) >> \text{shift0} + \text{MltDiff8}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
13:	else // Vertical direction
14:	$p_3(h, v) = (\text{ByrDiff} << 3) + (1 << 2)$
15:	$p_4(h, v) = (\text{MltDiff1} << 3) + (1 << 2)$
16:	$p_5(h, v) = (\text{MltDiff2} << 3) + (1 << 2)$
17:	$p_0(h, v) = ((p_3(h, v) >> \text{shift0} + \text{MltDiff3}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
18:	$p_6(h, v) = ((p_3(h, v) >> \text{shift0} + \text{MltDiff4}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
19:	$p_1(h, v) = ((p_4(h, v) >> \text{shift0} + \text{MltDiff5}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
20:	$p_7(h, v) = ((p_4(h, v) >> \text{shift0} + \text{MltDiff6}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
21:	$p_2(h, v) = ((p_5(h, v) >> \text{shift0} + \text{MltDiff7}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
22:	$p_8(h, v) = ((p_5(h, v) >> \text{shift0} + \text{MltDiff8}) << \text{shift0}) + (1 << (\text{shift0} - 1))$
23:	end if
24:	// $p_1(h, v), \dots, p_8(h, v)$ should be $p_1(h, v), \dots, p_8(h, v) = p_0(h, v)$, when you want to get the $p_{byr}(h, v)$
25:	$p_{byr}(h, v) = (\text{ByrDiff} << 3) + (1 << 2) // \text{Bayer pixel } p_{byr}(h, v)$

I.3.3.4 Extended Mode 3: eSHV (extended Slanted Horizontal or Vertical Direction-based Differential Mode)

Compression Mode eSHV references pixels that are located along the slanted horizontal or vertical direction.

Figure 295 defines the packet structure for Compression Mode eSHV (Nona-Cell). For this Compression Mode, Header field **Prediction IDX** shall contain 1, and Header field **Mode IDX** shall contain 3'b110.

The Payload shall contain compressed data for the nine pixels of a Nona-Cell:

- Field **Qstep** shall indicate the quantization step size.
- Fields **ByrDiff** and **MltDiff1** through **MltDiff8** shall contain the quantized differences between the predicted value and each pixel.

Field **ByrDiff** can also be regarded as **MltDiff0**; it is used for generation of the Bayer image, see **Section I.3.4**.

Table 97 shows pseudo-code for decoding an MPC Packet for Compression Mode eSHV (Nona-Cell).

Mode	Header			Payload									
	Prediction IDX	Mode IDX		Qstep[1:0]	ByrDiff[3:0]	MltDiff1[4:0]	MltDiff2[4:0]	MltDiff3[4:0]	MltDiff4[4:0]	MltDiff5[2:0]	MltDiff6[3:0]	MltDiff7[3:0]	MltDiff8[3:0]
eSHV	1	1	1	0									

Figure 295 MPC Packet Structure for Compression Mode eSHV (Nona-Cell)

Note:

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color. The Red- or Blue-first case is described in comments.

Note also that this pseudo-code only describes 3x3 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

Table 97 Decoder for Compression Mode eSHV (Nona-Cell)

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 2 : (Qstep==1) ? 3 : (Qstep==2) ? 4 : 5
3: // direction for  $p_k(h, v)$ 
4: ldiffo = abs( $p_3(h-1, v-1) - p_7(h-1, v-1)$ ) // color pixel first: abs( $p_3(h, v-1) - p_7(h, v-1)$ )
5: ldiff1 = abs( $p_4(h-1, v-1) - p_8(h-1, v-1)$ ) // color pixel first: abs( $p_4(h, v-1) - p_8(h, v-1)$ )
6: ldifff2 = abs( $p_3(h+1, v-1) - p_7(h+1, v-1)$ ) // color pixel first: abs( $p_3(h, v-1) - p_7(h, v-1)$ )
7: ldifff3 = abs( $p_4(h+1, v-1) - p_8(h+1, v-1)$ ) // color pixel first: abs( $p_4(h, v-1) - p_8(h, v-1)$ )
8: rdiffo = abs( $p_4(h-1, v-1) - p_6(h-1, v-1)$ ) // color pixel first: abs( $p_4(h+2, v-1) - p_6(h+2, v-1)$ )
9: rdifff1 = abs( $p_5(h-1, v-1) - p_7(h-1, v-1)$ ) // color pixel first: abs( $p_5(h+2, v-1) - p_7(h+2, v-1)$ )
10: rdifff2 = abs( $p_4(h+1, v-1) - p_6(h+1, v-1)$ ) // color pixel first: abs( $p_4(h+2, v-1) - p_6(h+2, v-1)$ )
11: rdifff3 = abs( $p_5(h+1, v-1) - p_7(h+1, v-1)$ ) // color pixel first: abs( $p_5(h+2, v-1) - p_7(h+2, v-1)$ )
12: ldifff = ldiffo + ldifff1 + ldifff2 + ldifff3
13: rdifff = rdiffo + rdifff1 + rdifff2 + rdifff3
14: left_en = (ldifff < rdifff) ? 1 : 0
15: // High-low direction
16: cent = left_en ?  $p_8(h-1, v-1) : p_6(h+1, v-1)$ 
17: low = left_en ?  $p_7(h-1, v-1) : p_7(h+1, v-1)$ 
18: high = left_en ?  $p_5(h-1, v-1) : p_3(h+1, v-1)$ 
19: ldifff = abs(cent - low)
20: hdifff = abs(cent - high)

```

```

21:   if left_en == 1
22:     if high_en == 1
23:       q0(h,v) = p5(h-1,v-1)          // color pixel first: q0(h,v) = p5(h-2,v-2)
24:       q1(h,v) = p2(h-1,v-1)          // color pixel first: q1(h,v) = p2(h-2,v-2)
25:       q2(h,v) = (p2(h-1,v-1) + p6(h,v-2)) >> 1 // color pixel first: q2(h,v) = p2(h-2,v-2)
26:       q3(h,v) = p8(h-1,v-1)          // color pixel first: q3(h,v) = p8(h-2,v-2)
27:       q6(h,v) = p7(h-1,v-1)          // color pixel first: q6(h,v) = p7(h-2,v-2)
28:     else
29:       q0(h,v) = p7(h-1,v-1)          // color pixel first: q0(h,v) = p7(h-2,v-2)
30:       q1(h,v) = p8(h-1,v-1)          // color pixel first: q1(h,v) = p8(h-2,v-2)
31:       q2(h,v) = p5(h-1,v-1)          // color pixel first: q2(h,v) = p5(h-2,v-2)
32:       q3(h,v) = p6(h-1,v-1)          // color pixel first: q3(h,v) = p6(h-2,v-2)
33:       q6(h,v) = p6(h-1,v-1)          // color pixel first: q6(h,v) = p6(h-2,v-2)
34:   end if
35:   // For decoding the pixel
36:   p0(h,v) = ((q0(h,v) >> shift0 + ByrDiff) << shift0) + (1 << (shift0 -1))
37:   p1(h,v) = ((q1(h,v) >> shift0 + MltPixel1) << shift0) + (1 << (shift0 -1))
38:   p2(h,v) = ((q2(h,v) >> shift0 + MltPixel2) << shift0) + (1 << (shift0 -1))
39:   p3(h,v) = ((q3(h,v) >> shift0 + MltPixel3) << shift0) + (1 << (shift0 -1))
40:   p6(h,v) = ((q6(h,v) >> shift0 + MltPixel4) << shift0) + (1 << (shift0 -1))
41:   // Prediction pixel
42:   q4(h,v) = p0(h,v)
43:   q5(h,v) = p1(h,v)
44:   q7(h,v) = p3(h,v)
45:   // For decoding the pixel
46:   p4(h,v) = ((q4(h,v) >> shift0 + MltPixel5) << shift0) + (1 << (shift0 -1))
47:   p5(h,v) = ((q5(h,v) >> shift0 + MltPixel6) << shift0) + (1 << (shift0 -1))
48:   p7(h,v) = ((q7(h,v) >> shift0 + MltPixel7) << shift0) + (1 << (shift0 -1))
49:   // Prediction pixel
50:   q8(h,v) = p4(h,v)
51:   p8(h,v) = ((q8(h,v) >> shift0 + MltPixel8) << shift0) + (1 << (shift0 -1))
52:   // Prediction pixel
53:   else // right direction
54:     if high_en == 1
55:       q2(h,v) = p3(h+1,v-1)          // color pixel first: q2(h,v) = p3(h+2,v-2)
56:       q1(h,v) = p0(h+1,v-1)          // color pixel first: q1(h,v) = p0(h+2,v-2)
57:       q0(h,v) = (p0(h+1,v-1) + p8(h,v-2)) >> 1 // color pixel first: q0(h,v) = p0(h+2,v-2)
58:       q3(h,v) = p6(h+1,v-1)          // color pixel first: q3(h,v) = p6(h+2,v-2)
59:       q8(h,v) = p7(h+1,v-1)          // color pixel first: q8(h,v) = p7(h+2,v-2)
60:     else
61:       q2(h,v) = p7(h+1,v-1)          // color pixel first: q2(h,v) = p7(h+2,v-2)
62:       q1(h,v) = p6(h+1,v-1)          // color pixel first: q1(h,v) = p6(h+2,v-2)
63:       q0(h,v) = p3(h+1,v-1)          // color pixel first: q0(h,v) = p3(h+2,v-2)

```

```

64:       $q_5(h,v) = p_8(h+1,v-1)$            // color pixel first:  $q_5(h,v) = p_8(h+2,v-2)$ 
65:       $q_8(h,v) = p_8(h+1,v-1)$            // color pixel first:  $q_8(h,v) = p_8(h+2,v-2)$ 
66:  end if
67:  // For decoding the pixel
68:   $p_2(h,v) = ((q_2(h,v) >> shift0 + ByrDiff) << shift0) + (1 << (shift0 -1))$ 
69:   $p_1(h,v) = ((q_1(h,v) >> shift0 + MltPixel1) << shift0) + (1 << (shift0 -1))$ 
70:   $p_0(h,v) = ((q_0(h,v) >> shift0 + MltPixel2) << shift0) + (1 << (shift0 -1))$ 
71:   $p_5(h,v) = ((q_5(h,v) >> shift0 + MltPixel3) << shift0) + (1 << (shift0 -1))$ 
72:   $p_8(h,v) = ((q_8(h,v) >> shift0 + MltPixel4) << shift0) + (1 << (shift0 -1))$ 
73:  // Prediction pixel
74:   $q_3(h,v) = p_1(h,v)$ 
75:   $q_4(h,v) = p_2(h,v)$ 
76:   $q_7(h,v) = p_5(h,v)$ 
77:  // For decoding the pixel
78:   $p_3(h,v) = ((q_3(h,v) >> shift0 + MltPixel6) << shift0) + (1 << (shift0 -1))$ 
79:   $p_4(h,v) = ((q_4(h,v) >> shift0 + MltPixel5) << shift0) + (1 << (shift0 -1))$ 
80:   $p_7(h,v) = ((q_7(h,v) >> shift0 + MltPixel7) << shift0) + (1 << (shift0 -1))$ 
81:  // Prediction pixel
82:   $q_6(h,v) = p_4(h,v)$ 
83:   $p_6(h,v) = ((q_6(h,v) >> shift0 + MltPixel8) << shift0) + (1 << (shift0 -1))$ 
84:  end if
85:  //  $p_1(h,v), \dots, p_8(h,v)$  should be  $p_1(h,v), \dots, p_8(h,v) = p_0(h,v)$ , when you want to get the  $p_{byr}(h,v)$ 
86:   $p_{byr}(h,v) = left\_en ? p_0(h,v) : p_2(h,v)$     // Bayer green pixel  $p_{byr}(h,v)$ 

```

I.3.3.5 Extended Mode 4: eMPD (extended Multi-Pixel-based Directional Differential Mode)

Compared to Standard Compression Mode AD (which uses Basic Prediction Mode), Compression Mode eMPD mode references more pixels and allocates one more bit to each Multi-Pixel in the MPC Packet.

Figure 296 defines the packet structure for Compression Mode PD. For this Compression Mode, Header field **Prediction IDX** shall contain 1, and Header field **Mode IDX** shall contain 3'b111.

The Payload shall contain compressed data for the nine pixels of a Nona-Cell:

- Field **Qstep** shall indicate the quantization step size.
- Fields **ByrDiff** and **MltDiff1** through **MltDiff8** shall contain the quantized differences between the predicted value and each pixel.

Field **ByrPixel** can also be regarded as **MltPixel0**; it is used for generation of the Bayer image, see *Section I.3.4*.

Table 98 shows pseudo-code for decoding an MPC Packet for Compression Mode eMPD.

Mode	Header			Payload									
	Prediction IDX	Mode IDX		Qstep[1:0]	ByrDiff[4:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[4:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[4:0]	MltDiff7[3:0]	MltDiff8[3:0]
eMPD	1	1	1	0step[1:0]	ByrDiff[4:0]	MltDiff1[3:0]	MltDiff2[3:0]	MltDiff3[4:0]	MltDiff4[3:0]	MltDiff5[3:0]	MltDiff6[4:0]	MltDiff7[3:0]	MltDiff8[3:0]

Figure 296 MPC Packet Structure for Compression Mode eMPD (Nona-Cell)

Note:

This pseudo-code assumes that the Green pixel appears first. If the first pixel is Red or Blue then the processing sequence must be changed, but compression is performed in the same way for each color. The Red- or Blue-first case is described in comments.

Note also that this pseudo-code only describes 3x3 pixels $p_k(h, v)$ to be compressed. These should be Green or Red/Blue pixels.

Table 98 Decoder for Compression Mode eMPD (Nona-Cell)

```

1: // Quantization step
2: shift0 = (Qstep==0) ? 1 : (Qstep==1) ? 2 : (Qstep==2) ? 3 : 4
3: // Preprocessing reference data
4: threshold = green pixel ?10 : 20
5: href = abs(pg(h-2,v-2) - p6(h,v-2))
6: vref = abs(pg(h-2,v-2) - p2(h-2,v))
7: mref = href - vref
8: diag_ref = green pixel ? pg(h-1,v-1) : pg(h-1,v-2)
9: // color pixel first: green pixel ? pg(h,v-1) : pg(h-2,v-2)
10: // For decoding p0(h,v)
11: if (Line count < 6) // Vertical line counter, Boundary condition
12:     q0 = p2(h-2,v)
13: else if (unit count == 0) // unit count is for processing unit block, boundary condition
14:     q0 = p6(h,v-2)
15: else //Prediction value q
16:     if (mref < threshold)
17:         q0 = diag_ref
18:     else
19:         if pg(h-2,v-2) >= max{p6(h,v-2), p2(h-2,v)}

```

```

20:            $q_0 = \min\{p_6(h,v-2), p_2(h-2,v)\}$ 
21:           else if  $p_8(h-2,v-2) \leq \min\{p_6(h,v-2), p_2(h-2,v)\}$ 
22:            $q_0 = \max\{p_6(h,v-2), p_2(h-2,v)\}$ 
23:           else
24:            $q_0 = p_6(h,v-2) + p_2(h-2,v) - p_8(h-2,v-2)$ 
25:           end if
26:       end if
27:   end if
28:   // For decoding the pixel  $p_k(h,v)$ ,  $k=\{0,1,\dots,8\}$ 
29:    $p_0(h,v) = ((q_0 >> shift0 + ByrDiff) << shift0) + (1 << (shift0 - 1))$ 
30:   // For decoding the pixel  $p_1(h,v)$ 
31:   if (Line count < 6)           // Vertical line counter, Boundary condition
32:        $q_1 = p_0(h,v)$ 
33:   else if (unit count == 0)    // unit count is for processing unit block, boundary condition
34:        $q_1 = p_7(h,v-2)$ 
35:   else                           //Prediction value q
36:       if  $p_6(h,v-2) \geq \max\{p_0(h,v), p_7(h,v-2)\}$           //Prediction value q of the preview pixel
37:            $q_1(h,v) = \min\{p_0(h,v), p_7(h,v-2)\}$ 
38:       else if  $p_6(h,v-2) \leq \min\{p_0(h,v), p_7(h,v-2)\}$ 
39:            $q_1(h,v) = \max\{p_0(h,v), p_7(h,v-2)\}$ 
40:       else
41:            $q_1(h,v) = p_0(h,v) + p_7(h,v-2) - p_6(h,v-2)$ 
42:       end if
43:   end if
44:    $p_1(h,v) = ((q_1 >> shift0 + MltDiff1) << shift0) + (1 << (shift0 - 1))$ 
45:   // For decoding the pixel  $p_2(h,v)$ 
46:   if (Line count < 6)           // Vertical line counter, Boundary condition
47:        $q_2 = p_1(h,v)$ 
48:   else if (unit count == 0)    // unit count is for processing unit block, boundary condition
49:        $q_2 = p_8(h,v-2)$ 
50:   else                           //Prediction value q
51:       if  $p_7(h,v-2) \geq \max\{p_1(h,v), p_8(h,v-2)\}$           //Prediction value q of the preview pixel
52:            $q_2(h,v) = \min\{p_1(h,v), p_8(h,v-2)\}$ 
53:       else if  $p_7(h,v-2) \leq \min\{p_1(h,v), p_8(h,v-2)\}$ 
54:            $q_2(h,v) = \max\{p_1(h,v), p_8(h,v-2)\}$ 
55:       else
56:            $q_2(h,v) = p_1(h,v) + p_8(h,v-2) - p_7(h,v-2)$ 
57:       end if
58:   end if
59:    $p_2(h,v) = ((q_2 >> shift0 + MltDiff2) << shift0) + (1 << (shift0 - 1))$ 
60:   // For decoding the pixel  $p_3(h,v)$ 
61:    $q_3(h,v) = p_0(h,v)$            //Prediction value q of the preview pixel
62:    $p_3(h,v) = ((q_3 >> shift0 + MltDiff3) << shift0) + (1 << (shift0 - 1))$ 

```

```

63: // For decoding the pixel  $p_4(h,v)$ 
64: if (Line count < 6) // Vertical line counter, Boundary condition
65:    $q_4 = p_3(h,v)$ 
66: else if (unit count == 0) // unit count is for processing unit block, boundary condition
67:    $q_2 = p_1(h,v)$ 
68: else //Prediction value q
69:   if  $p_0(h,v) \geq \max\{p_3(h,v), p_1(h,v)\}$  //Prediction value q of the preview pixel
70:      $q_4(h,v) = \min\{p_3(h,v), p_1(h,v)\}$ 
71:   else if  $p_0(h,v) \leq \min\{p_3(h,v), p_1(h,v)\}$ 
72:      $q_4(h,v) = \max\{p_3(h,v), p_1(h,v)\}$ 
73:   else
74:      $q_4(h,v) = p_3(h,v) + p_1(h,v) - p_0(h,v)$ 
75:   end if
76: end if
77:  $p_4(h,v) = ((q_4 >> shift0 + MltDiff4) << shift0) + (1 << (shift0 - 1))$ 
78: // For decoding the pixel  $p_5(h,v)$ 
79: if (Line count < 6) // Vertical line counter, Boundary condition
80:    $q_5 = p_4(h,v)$ 
81: else if (unit count == 0) // unit count is for processing unit block, boundary condition
82:    $q_5 = p_2(h,v)$ 
83: else //Prediction value q
84:   if  $p_1(h,v) \geq \max\{p_4(h,v), p_2(h,v)\}$  //Prediction value q of the preview pixel
85:      $q_5(h,v) = \min\{p_4(h,v), p_2(h,v)\}$ 
86:   else if  $p_1(h,v) \leq \min\{p_4(h,v), p_2(h,v)\}$ 
87:      $q_5(h,v) = \max\{p_4(h,v), p_2(h,v)\}$ 
88:   else
89:      $q_5(h,v) = p_4(h,v) + p_2(h,v) - p_1(h,v)$ 
90:   end if
91: end if
92:  $p_5(h,v) = ((q_5 >> shift0 + MltDiff5) << shift0) + (1 << (shift0 - 1))$ 
93: // For decoding the pixel  $p_6(h,v)$ 
94:  $q_6(h,v) = p_5(h,v)$  //Prediction value q of the preview pixel
95:  $p_6(h,v) = ((q_6 >> shift0 + MltDiff6) << shift0) + (1 << (shift0 - 1))$ 
96: // For decoding the pixel  $p_7(h,v)$ 
97: if (Line count < 6) // Vertical line counter, Boundary condition
98:    $q_7 = p_6(h,v)$ 
99: else if (unit count == 0) // unit count is for processing unit block, boundary condition
100:    $q_7 = p_4(h,v)$ 
101: else //Prediction value q
102:   if  $p_3(h,v) \geq \max\{p_6(h,v), p_4(h,v)\}$  //Prediction value q of the preview pixel
103:      $q_7(h,v) = \min\{p_6(h,v), p_4(h,v)\}$ 
104:   else if  $p_3(h,v) \leq \min\{p_6(h,v), p_4(h,v)\}$ 
105:      $q_7(h,v) = \max\{p_6(h,v), p_4(h,v)\}$ 

```

```

106:      else
107:           $q_7(h,v) = p_6(h,v) + p_4(h,v) - p_3(h,v)$ 
108:      end if
109:  end if
110:   $p_7(h,v) = ((q_7 >> shift0 + MltDiff7) << shift0) + (1 << (shift0 - 1))$ 
111:  // For decoding the pixel  $p_8(h,v)$ 
112:  if (Line count < 6)           // Vertical line counter, Boundary condition
113:       $q_8 = p_7(h,v)$ 
114:  else if (unit count == 0)    // unit count is for processing unit block, boundary condition
115:       $q_8 = p_5(h,v)$ 
116:  else                           //Prediction value q
117:      if  $p_4(h,v) \geq \max\{p_7(h,v), p_5(h,v)\}$            //Prediction value q of the preview pixel
118:           $q_8(h,v) = \min\{p_7(h,v), p_5(h,v)\}$ 
119:      else if  $p_4(h,v) \leq \min\{p_7(h,v), p_5(h,v)\}$ 
120:           $q_8(h,v) = \max\{p_7(h,v), p_5(h,v)\}$ 
121:      else
122:           $q_8(h,v) = p_7(h,v) + p_5(h,v) - p_4(h,v)$ 
123:      end if
124:  end if
125:   $p_8(h,v) = ((q_8 >> shift0 + MltDiff8) << shift0) + (1 << (shift0 - 1))$ 
126:  //  $p_1(h,v), \dots, p_8(h,v)$  should be  $p_1(h,v), \dots, p_8(h,v) = p_0(h,v)$ , when you want to get the  $p_{byr}(h,v)$ 
127:   $p_{byr}(h,v) = p_0(h,v)$            // Bayer pixel  $p_{byr}(h,v)$ 

```

I.3.4 Low Resolution Image Generation (Nona-Cell)

The MPC architecture for Nona-Cell image sensors supports an additional low-resolution Bayer image. The Bayer image is reconstructed by p_{byr} which is described in the Compression Mode Decoder pseudo-code in **Section I.3.2.1** through **Section I.3.3.5**.

For all Nona-Cell Compression Modes, the Bayer image can be generated using only information that is present in the first half of the MPC Packet. This is shown in **Figure 297**.

- For all other Compression Modes, the Bayer image is generated using field **ByrDiff** or **ByrPixel**.

			Mode Summary for Nona-Cell Image Sensor												
Compression Mode	Header	Mode IDX	Payload												
	Prediction IDX	Mode IDX	Basic Prediction Mode												
AD	0	0	Qstep[1:0]	ByrDiff[4:0]	M00M01[1:0]	M00M02[1:0]	M00M03[1:0]	M00M04[1:0]	M00M05[1:0]	M00M06[1:0]	M00M07[1:0]	M00M08[1:0]	M00M09[1:0]	M00M10[1:0]	
DGO	0	1	0	Qstep[1:0]	ByrDiff[4:0]	M00M11[4:0]	M00M12[4:0]	M00M13[4:0]	M00M14[4:0]	M00M15[4:0]	M00M16[4:0]	M00M17[4:0]	M00M18[4:0]	M00M19[4:0]	
FNR	0	1	0	ByrPixel[4:0]	ByrPixel[1:0]	M00M20[4:0]	M00M21[4:0]	M00M22[4:0]	M00M23[4:0]	M00M24[4:0]	M00M25[4:0]	M00M26[4:0]	M00M27[4:0]	M00M28[4:0]	
-	0	1	1	0	M00M29[4:0]	M00M30[4:0]	M00M31[4:0]	M00M32[4:0]	M00M33[4:0]	M00M34[4:0]	M00M35[4:0]	M00M36[4:0]	M00M37[4:0]	M00M38[4:0]	
-	0	1	1	1	M00M39[4:0]	M00M40[4:0]	M00M41[4:0]	M00M42[4:0]	M00M43[4:0]	M00M44[4:0]	M00M45[4:0]	M00M46[4:0]	M00M47[4:0]	M00M48[4:0]	
REMOVED FOR FUTURE UPDATE															
			Advanced Prediction Mode												
eAD	1	0	0	Qstep	ByrDiff[4:0]	M00M49[1:0]	M00M50[1:0]	M00M51[1:0]	M00M52[1:0]	M00M53[1:0]	M00M54[1:0]	M00M55[1:0]	M00M56[1:0]	M00M57[1:0]	
eHVO	1	0	1	Dir	Qstep[1:0]	ByrDiff[4:0]	M00M58[1:0]	M00M59[1:0]	M00M60[1:0]	M00M61[1:0]	M00M62[1:0]	M00M63[1:0]	M00M64[1:0]	M00M65[1:0]	M00M66[1:0]
eHVI	1	1	0	Dir	Qstep[1:0]	ByrDiff[4:0]	M00M67[1:0]	M00M68[1:0]	M00M69[1:0]	M00M70[1:0]	M00M71[1:0]	M00M72[1:0]	M00M73[1:0]	M00M74[1:0]	M00M75[1:0]
eHSV	1	1	1	0	Qstep[1:0]	ByrDiff[4:0]	M00M76[1:0]	M00M77[1:0]	M00M78[1:0]	M00M79[1:0]	M00M80[1:0]	M00M81[1:0]	M00M82[1:0]	M00M83[1:0]	M00M84[1:0]
eMPD	1	1	1	1	Qstep[1:0]	ByrDiff[4:0]	M00M85[1:0]	M00M86[1:0]	M00M87[1:0]	M00M88[1:0]	M00M89[1:0]	M00M90[1:0]	M00M91[1:0]	M00M92[1:0]	M00M93[1:0]

Figure 297 Low Resolution Image Generation by P_{byr} (Nona-Cell)

5979

I.3.5 Adjustment of Compression Ratio (Nona-Cell)

5980 **Figure 298** and **Figure 299** define the Packet structures for Compression Ratios 10:6 and 10:7, respectively, for Nona-Cell. Differences in the bit allocations are highlighted in red.

Mode Summary for Nona-Cell Image Sensor														
Compression Mode	Header			Payload										
	Prediction IDX	Mode IDX		Basic Prediction Mode										
AD	0	0	0	Qstep[1:0]	ByrDiff[4:0]	MtDiff0[4:0]	MtDiff1[4:0]	MtDiff2[4:0]	MtDiff3[4:0]	MtDiff4[4:0]	MtDiff5[4:0]	MtDiff6[4:0]	MtDiff7[4:0]	MtDiff8[4:0]
DGD	0	1	0	0	Qstep[1:0]	ByrDiff[5:0]	MtDiff1[5:0]	MtDiff2[6:0]	MtDiff3[5:0]	MtDiff4[6:0]	MtDiff5[3:0]	MtDiff6[3:0]	MtDiff7[3:0]	MtDiff8[3:0]
FNR	0	1	0	1	ByrPixel[5:0]	MtPixel1[4:0]	MtPixel2[5:0]	MtPixel3[4:0]	MtPixel4[5:0]	MtPixel5[4:0]	MtPixel6[5:0]	MtPixel7[4:0]	MtPixel8[5:0]	
-	0	1	0											
-	0	1	1											
RESERVED FOR FUTURE UPDATE														
RESERVED FOR FUTURE UPDATE														
Advanced Prediction Mode														
eAD	1	0	0	2step	ByrDiff[5:0]	MtDiff0[4:0]	MtDiff1[4:0]	MtDiff2[4:0]	MtDiff3[4:0]	MtDiff4[4:0]	MtDiff5[4:0]	MtDiff6[4:0]	MtDiff7[4:0]	MtDiff8[4:0]
eHV0	1	0	1	Dir	Qstep[1:0]	ByrDiff[5:0]	MtDiff1[5:0]	MtDiff2[5:0]	MtDiff3[4:0]	MtDiff4[4:0]	MtDiff5[4:0]	MtDiff6[4:0]	MtDiff7[4:0]	MtDiff8[4:0]
eHV1	1	1	0	Dir	Qstep[1:0]	ByrDiff[7:0]	MtDiff1[7:0]	MtDiff2[7:0]	MtDiff3[7:0]	MtDiff4[7:0]	MtDiff5[3:0]	MtDiff6[3:0]	MtDiff7[3:0]	MtDiff8[3:0]
eSHV	1	1	1	0	Qstep[1:0]	ByrDiff[4:0]	MtDiff1[5:0]	MtDiff2[5:0]	MtDiff3[5:0]	MtDiff4[5:0]	MtDiff5[3:0]	MtDiff6[4:0]	MtDiff7[4:0]	MtDiff8[4:0]
eMPD	1	1	1	1	Qstep[1:0]	ByrDiff[5:0]	MtDiff1[4:0]	MtDiff2[4:0]	MtDiff3[5:0]	MtDiff4[4:0]	MtDiff5[4:0]	MtDiff6[5:0]	MtDiff7[4:0]	MtDiff8[4:0]

5982 **Figure 298 MPC Packet Structures for Compression Ratio 10:6 (Nona-Cell)**

Mode Summary for Nona-Cell Image Sensor														
Compression Mode	Header			Payload										
	Prediction IDX	Mode IDX		Basic Prediction Mode										
AD	0	0	0	Qstep[1:0]	ByrDiff[4:0]	MtDiff0[5:0]	MtDiff1[5:0]	MtDiff2[5:0]	MtDiff3[5:0]	MtDiff4[5:0]	MtDiff5[5:0]	MtDiff6[5:0]	MtDiff7[5:0]	MtDiff8[5:0]
DGD	0	1	0	0	Qstep[1:0]	ByrDiff[6:0]	MtDiff1[6:0]	MtDiff2[6:0]	MtDiff3[6:0]	MtDiff4[7:0]	MtDiff5[4:0]	MtDiff6[4:0]	MtDiff7[4:0]	MtDiff8[4:0]
FNR	0	1	0	1	ByrPixel[6:0]	MtPixel1[5:0]	MtPixel2[6:0]	MtPixel3[5:0]	MtPixel4[6:0]	MtPixel5[5:0]	MtPixel6[6:0]	MtPixel7[5:0]	MtPixel8[6:0]	
-	0	1	0											
-	0	1	1											
RESERVED FOR FUTURE UPDATE														
RESERVED FOR FUTURE UPDATE														
Advanced Prediction Mode														
eAD	1	0	0	2step	ByrDiff[4:0]	MtDiff0[5:0]	MtDiff1[5:0]	MtDiff2[5:0]	MtDiff3[5:0]	MtDiff4[5:0]	MtDiff5[5:0]	MtDiff6[5:0]	MtDiff7[5:0]	MtDiff8[5:0]
eHV0	1	0	1	Dir	Qstep[1:0]	ByrDiff[6:0]	MtDiff1[6:0]	MtDiff2[6:0]	MtDiff3[5:0]	MtDiff4[5:0]	MtDiff5[5:0]	MtDiff6[5:0]	MtDiff7[5:0]	MtDiff8[5:0]
eHV1	1	1	0	Dir	Qstep[1:0]	ByrDiff[8:0]	MtDiff1[8:0]	MtDiff2[8:0]	MtDiff3[8:0]	MtDiff4[8:0]	MtDiff5[4:0]	MtDiff6[4:0]	MtDiff7[4:0]	MtDiff8[4:0]
eSHV	1	1	1	0	Qstep[1:0]	ByrDiff[5:0]	MtDiff1[6:0]	MtDiff2[8:0]	MtDiff3[6:0]	MtDiff4[6:0]	MtDiff5[4:0]	MtDiff6[5:0]	MtDiff7[5:0]	MtDiff8[5:0]
eMPD	1	1	1	1	Qstep[1:0]	ByrDiff[6:0]	MtDiff1[5:0]	MtDiff2[5:0]	MtDiff3[6:0]	MtDiff4[5:0]	MtDiff5[5:0]	MtDiff6[6:0]	MtDiff7[5:0]	MtDiff8[5:0]

5983 **Figure 299 MPC Packet Structures for Compression Ratio 10:7 (Nona-Cell)**

5984
 5985
 5986 **Figure 300** defines the values of *shift0* and *shift1* to use (i.e., in the respective Decoder pseudo-code per Compression Mode) for each combination of Nona-Cell Compression Mode, value of field **Qstep**, and Compression Ratio.

Quantization Bits Map							
Basic Profile							
Mode	Quant. IDX (Qstep)	Compression Ratio 10:5		Compression Ratio 10:6		Compression Ratio 10:7	
		Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)
AD	0	1	0	1	0	1	0
	1	2	1	2	1	2	1
	2	3	2	3	2	3	2
	3	5	3	5	3	5	3
DGD	0	2	-	1	-	0	-
	1	3	-	2	-	1	-
	2	4	-	3	-	2	-
	3	5	-	4	-	3	-
FNR	-	5	6	4	5	3	4
Advanced Profile							
Mode	Quant. IDX (Qstep)	Compression Ratio 10:5		Compression Ratio 10:6		Compression Ratio 10:7	
		Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)	Shift0 bits (<i>shift0</i>)	Shift1 bits (<i>shift1</i>)
eAD	0	5	4	5	4	5	4
	1	5	5	5	5	5	5
eMPD	0	1	-	0	-	0	-
	1	2	-	1	-	1	-
	2	3	-	2	-	2	-
	3	4	-	3	-	3	-
eHVD	0	1	-	0	-	0	-
	1	2	-	1	-	1	-
	2	3	-	2	-	2	-
	3	4	-	3	-	3	-
eHVI	0	1	-	0	-	0	-
	1	2	-	1	-	1	-
	2	3	-	2	-	2	-
	3	4	-	3	-	3	-
eSHV	0	2	-	1	-	0	-
	1	3	-	2	-	1	-
	2	4	-	3	-	2	-
	3	5	-	4	-	3	-

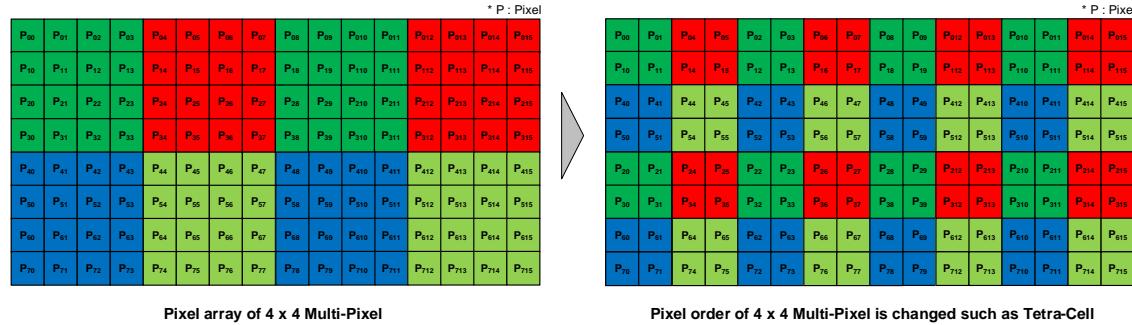
5987

Figure 300 Quantization Bits for Compression Ratios (Nona-Cell)

I.4 MPC for N x N Multi-Pixel Image Sensor [Informative]

MPC provides scalability for N x N multi-pixel image sensors. Regarding N x N, compression for Tetra-Cell provides scalability for 2N x 2N ($N \geq 2$) Multi-Pixel sensor compression. And compression for Nona-Cell provides scalability for 3N x 3N ($N \geq 2$) Multi-Pixel sensor compression.

For example, MPC compresses 4x4 Multi-Pixel images using the Tetra-Cell compression scheme. If the pixel order of 4x4 Multi-Pixels is changed such as the Tetra-Cell example illustrated in **Figure 301**, then MPC compresses 4x4 Multi-Pixel images. After compression, the pixel order is reverted to the original pixel order.



Participants

The list below includes those persons who participated in the Working Group that developed this Specification and who consented to appear on this list.

Andy Baldman, Keysight Technologies Inc.	Takashi Miyamoto, Sony Group Corporation
Nadav Banet, Valens Semiconductor	Alexander Mokhoria, Robert Bosch GmbH
Craig Bezdek, Teledyne LeCroy	Makoto Nariya, Sony Group Corporation
Thomas Blon, Silicon Line GmbH	Kavitha Naveen, Tektronix, Inc.
Steven Chang, MediaTek Inc.	Allan Paul, Cadence Design Systems, Inc.
Geraud Cheenne, STMicroelectronics	Joao Pereira, Synopsys, Inc.
Wei Cheng Ku, MediaTek Inc.	Radu Pitigoi-Aron, Qualcomm Incorporated
Stephen Creaney, Cadence Design Systems, Inc.	Alex Qiu, NVIDIA
Yoshihiko Deoka, Sony Group Corporation	Loren Reiss, Cadence Design Systems, Inc.
James Goel, Qualcomm Incorporated	Teong Rong Chua, Sony Group Corporation
Jason Gonzalez, Qualcomm Incorporated	Matthew Ronning, Sony Group Corporation
Philip Hawkes, Qualcomm Incorporated	Hugo Santos, Synopsys, Inc.
Thomas Hogenmueller, Robert Bosch GmbH	Frank Seto, Samsung Electronics, Co.
Timo Kaikumaa, Intel Corporation	Greg Stewart, Analogix Semiconductor, Inc.
JaeHyuck Kang, Samsung Electronics, Co.	Dale Stoltzka, Samsung Electronics, Co.
Soichi Kayamori, Sony Group Corporation	Hiroo Takahashi, Sony Group Corporation
Samson Kim, Qualcomm Incorporated	Giuseppe Tofanicchio, STMicroelectronics
Tom Kopet, onsemi	Guizhen Wang, HiSilicon Technologies Co. Ltd.
Radha Krishna Atukula, NVIDIA	Frank Wang, OmniVision Technologies, Inc.
Raj Kumar Nagpal, Synopsys, Inc.	Rick Wietfeldt, Qualcomm Incorporated
Daechul Kwon, Samsung Electronics, Co.	George Wiley, Qualcomm Incorporated
Ariel Lasry, Qualcomm Incorporated	Stephen Wong, Intel Corporation
Ethan Lau, MediaTek Inc.	Charles Wu, OmniVision Technologies, Inc.
Wonseok Lee, Samsung Electronics, Co.	Kevin Yee, Samsung Electronics, Co.
William Lo, Axonne Inc.	Michel Yeh, MediaTek Inc.
Larry Madar, Google LLC	Naoki Yokoshima, Sony Group Corporation
Cedric Marta, Synopsys, Inc.	Sang Young Youn, Google LLC
Nuno Martins, onsemi	Eunseung Yun, Samsung Electronics, Co.

Past Contributors to v4.0:

Sakari Ailus, Intel Corporation
Radha Krishna Atukula, NVIDIA
Nadav Banet, Valens Semiconductor
Gyeonghan Cha, Samsung Electronics, Co.
Geraud Cheenne, STMicroelectronics
Teong Rong Chua, Sony Corporation
Edo Cohen, Valens Semiconductor
Tomer Cohen, Samsung Electronics, Co.
Al Czamara, Test Evolution
James Goel, Qualcomm Incorporated
Takayuki Hirama, Sony Corporation
Natsuko Ibuki, Google, LLC
Jae Hyuck Kang, Samsung Electronics, Co.
Soichi Kayamori, Sony Corporation
Tom Kopet, ON Semiconductor
Daechul Kwon, Samsung Electronics, Co.
Ariel Lasry, Qualcomm Incorporated
Wonseok Lee, Samsung Electronics, Co.
Mark Lewis, Cadence Design Systems, Inc.
Larry Madar, Google, LLC

Cedric Marta, Synopsys, Inc.
Adi Menachem, Valens Semiconductor
Takashi Miyamoto, Sony Corporation
Yoshitomo Osawa, Sony Corporation
Josh Pan, MediaTek Inc.
Radu Pitigoi-Aron, Qualcomm Incorporated
QuanLin Qiu, NVIDIA
Michael Ripley, Intel Corporation
Sabarish Sridhar, Google, LLC
Dale Stoltzka, Samsung Electronics, Co.
Hiroo Takahashi, Sony Corporation
Haran Thanigasalam, Intel Corporation
Giuseppe Tofanicchio, STMicroelectronics
Ayshwarya Venkataraman, Robert Bosch
GmbH
Rick Wietfeldt, Qualcomm Incorporated
George Wiley, Qualcomm Incorporated
Charles Wu, OmniVision Technologies, Inc.
Eunseung Yun, Samsung Electronics, Co.

Past Contributors to v3.0:

Sakari Ailus, Intel Corporation
Rob Anhofer, MIPI Alliance (staff)
Radha Krishna Atukula, NVIDIA
Uwe Beutnagel-Buchner, Robert Bosch GmbH
Gyeonghan Cha, Samsung Electronics, Co.
Eric Ching, Microchip Technology
Niharika Singh Chouhan, L&T Technology Services
Teong Rong Chua, Sony Corporation
Zhang Chunrong, Advanced Micro Devices, Inc.
Tomer Cohen, Samsung Electronics, Co.
Al Czamara, Test Evolution
Chris Grigg, MIPI Alliance (staff)
Jason Hawken, Advanced Micro Devices, Inc.
Hsiehchang Ho, MediaTek Inc.
Toshihisa Hyakudai, Sony Corporation
Kai Ruben Josefsen, OmniVision Technologies, Inc.
Tom Kopet, ON Semiconductor
Mark Lewis, Cadence Design Systems, Inc.
Kenneth Ma, HiSilicon Technologies Co. Ltd.
Kumaravel Manickam, L&T Technology Services
Cedric Marta, Synopsys, Inc.
Mikko Muukki, HiSilicon Technologies Co. Ltd.
Manuel Ortiz, Intel Corporation
Josh Pan, MediaTek Inc.
Prachi Patel, Cadence Design Systems, Inc.
Allan Paul, Cadence Design Systems, Inc.
Radu Pitigoi-Aron, Qualcomm Incorporated
Kondalarao Polisetty, Xilinx Inc.
Quan Lin Qiu, NVIDIA
Rahul R, L&T Technology Services
Matthew Ronning, Sony Corporation
Yon Jun Shin, Samsung Electronics, Co.
Richard Sproul, Cadence Design Systems, Inc.
Vinoth Srinivasan, L&T Technology Services
Hiroo Takahashi, Sony Corporation
Haran Thanigasalam, Intel Corporation
Rick Wietfeldt, Qualcomm Incorporated
George Wiley, Qualcomm Incorporated
David Woolf, University of New Hampshire
InterOperability Lab (UNH-IOL)
Charles Wu, OmniVision Technologies, Inc.
Long Wu, Synopsys, Inc.

Past Contributors to v2.1:

Sakari Ailus, Intel Corporation
Rob Anhofer, MIPI Alliance (staff)
Radha Krishna Atukula, NVIDIA
Chua Teong Rong, Sony Corporation
Gyeonghan Cha, Samsung Electronics, Co.
Zhang Chunrong, Advanced Micro Devices, Inc.
Chris Grigg, MIPI Alliance (staff)
Jason Hawken, Advanced Micro Devices, Inc.
Tom Kopet, ON Semiconductor
Cedric Marta, Synopsys, Inc.
Mikko Muukki, HiSilicon Technologies Co. Ltd.
Manuel Ortiz, Intel Corporation
Prachi Patel, Cadence Design Systems, Inc.
Matthew Ronning, Sony Corporation
Yakov Simhony, Cadence Design Systems, Inc.
Richard Sproul, Cadence Design Systems, Inc.
Hiroo Takahashi, Sony Corporation
Haran Thanigasalam, Intel Corporation
George Wiley, Qualcomm Incorporated

Past Contributors to v2.0:

Hirofumi Adachi, Teradyne Inc.
Sakari Ailus, Intel Corporation
Rob Anhofer, MIPI Alliance
Radha Krishna Atukula, NVIDIA
Kaberi Banerjee, Lattice Semiconductor Corp.
Thierry Berdah, Cadence Design Systems, Inc.
Thomas Blon, Silicon Line GmbH
Teong Rong Chua, Sony Corporation
Zhang Chunrong, Advanced Micro Devices, Inc.
Tatsuyuki Fukushima, Teradyne Inc.
Karan Galhotra, Synopsys, Inc.
Vaibhav Gupta, Mentor Graphics
Mattias Gustafsson, OmniVision Technologies, Inc.
Mohamed Hafed, Introspect Test Technology Inc.
Will Harris, Advanced Micro Devices, Inc.
Jason Hawken, Advanced Micro Devices, Inc.
Hsieh Chang Ho, MediaTek Inc.
Norihiko Ichimaru, Sony Corporation
Henrik Icking, Intel Corporation
Yukichi Inoue, Teradyne Inc.
Kayoko Ishiwata, Toshiba Corporation
Grant Jennings, Lattice Semiconductor Corp.
Jacob Joseph, NVIDIA
Mrudula Kanuri, NVIDIA
Nadine Kolment, Introspect Test Technology Inc.
Tom Kopet, ON Semiconductor
Thomas Krause, Texas Instruments Incorporated
Yoav Lavi, VLSI Plus Ltd.
Cedric Marta, Synopsys, Inc.
Mikko Muukki, HiSilicon Technologies Co. Ltd.
Raj Kumar Nagpal, Synopsys, Inc.
Amir Naveed, Qualcomm Incorporated
Laurent Pinchart, Google, Inc.
Alex Qiu, NVIDIA
Matthew Ronning, Sony Corporation
Yaron Schwartz, Cadence Design Systems, Inc.
Sho Sengoku, Qualcomm Incorporated
Yakov Simhony, Cadence Design Systems, Inc.
Gaurav Singh, Synopsys, Inc.
Richard Sproul, Cadence Design Systems, Inc.
Tatsuya Sugioka, Sony Corporation
Hiroo Takahashi, Sony Corporation
Haran Thanigasalam, Intel Corporation
Bruno Trematore, Toshiba Corporation
Rick Wietfeldt, Qualcomm Incorporated
George Wiley, Qualcomm Incorporated
Charles Wu, OmniVision Technologies, Inc.
Hirofumi Yoshida, Sony Corporation
MinJun Zhao, HiSilicon Technologies Co. Ltd