

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

Version 1.3

29 May 2014

MIPI Board Adopted 07 October 2014

This document is subject to further editorial and technical development.

NOTICE OF DISCLAIMER

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

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

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

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

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

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

Contents

C	ontents	5	ii
F	igures.		vii
T	ables		xii
		History	
1		oduction	
1	1.1	Scope	
	1.1	Purpose	
2		ninology	
4	2.1	Use of Special Terms	
	2.2	Definitions	
	2.3	Abbreviations	
	2.4	Acronyms	3
3	Refe	rences	5
4		view of CSI-2	
5		2 Layer Definitions	
		·	
6		era Control Interface (CCI)	
	6.1 6.1.1	Data Transfer Protocol	
	6.1.2		
	6.2	CCI Slave Addresses.	
	6.3	CCI Multi-Byte Registers	
	6.3.1	Overview	15
	6.3.2	, , , , , , , , , , , , , , , , , , ,	
	6.3.3	Multi-Byte Register Protocol	
	6.4	Electrical Specifications and Timing for I/O Stages	22
7	Phys	ical Layer	
	7.1	D-PHY Physical Layer Option	
	7.2	C-PHY Physical Layer Option	
8	Mult	ti-Lane Distribution and Merging	
	8.1	Lane Distribution for the D-PHY Physical Layer Option	
	8.2	Lane Distribution for the C-PHY Physical Layer Option	
	8.3	Multi-Lane Interoperability	
9		Level Protocol	
	9.1	Low Level Protocol Packet Format	
	9.1.1 9.1.2	Low Level Protocol Long Packet FormatLow Level Protocol Short Packet Format	
	9.1.2	Data Identifier (DI)	
	9.3	Virtual Channel Identifier	
	9.4	Data Type (DT)	
	9.5	Packet Header Error Correction Code for D-PHY Physical Layer Option	
	9.5.1	General Hamming Code Applied to Packet Header	

9.5.2	2 Hamming-Modified Code	48
9.5.3		
9.5.4	4 Applying ECC on RX Side	51
9.6	Checksum Generation.	53
9.7	Packet Spacing	55
9.8	Synchronization Short Packet Data Type Codes	56
9.8.1	• • • • • • • • • • • • • • • • • • • •	
9.8.2	2 Line Synchronization Packets	57
9.9	Generic Short Packet Data Type Codes	57
9.10	Packet Spacing Examples	
9.11	Packet Data Payload Size Rules	60
9.12	Frame Format Examples	61
9.13	Data Interleaving	63
9.13	.1 Data Type Interleaving	63
9.13	.2 Virtual Channel Identifier Interleaving	66
10 C	olor Spaces	68
10.1	RGB Color Space Definition	
10.2	YUV Color Space Definition	
11 D:	ata Formats	
11.1		
11.1	• • • • • • • • • • • • • • • • • • • •	
11.1	· · · · · · · · · · · · · · · · · · ·	
11.1		
11.2	e	
11.2	~ ·	
11.2		
11.2		
11.2		
11.2		
11.3	6	
11.3		
11.3		
11.3		
11.3		
11.4	RAW Image Data	
11.4		
11.4		
11.4		
11.4	.4 RAW10	90
11.4	.5 RAW12	91
11.4	.6 RAW14	92
11.5	User Defined Data Formats	93
12 R	ecommended Memory Storage	96
12.1	General/Arbitrary Data Reception	
12.1	RGB888 Data Reception	
12.2	RGB666 Data Reception	
12.3	RGB565 Data Reception	
- -	2.02000 Daw 1.000ptott	

12.5	RGB555 Data Reception	98
12.6	RGB444 Data Reception	98
12.7	YUV422 8-bit Data Reception	
12.8	YUV422 10-bit Data Reception	
12.9	YUV420 8-bit (Legacy) Data Reception	
12.10	YUV420 8-bit Data Reception	
12.11	YUV420 10-bit Data Reception	
12.12	RAW6 Data Reception	
12.13	RAW7 Data Reception	
12.14	RAW8 Data Reception	
12.15	RAW10 Data Reception	
12.16	RAW12 Data Reception	
12.17	RAW14 Data Reception	
Annex A		
A.1	Introduction	
A.2	JPEG Data Definition	
A.3	Image Status Information	
A.4	Embedded Images	
A.5	JPEG8 Non-standard Markers	
A.6	JPEG8 Data Reception	
Annex B	F	
B.1	Overview	
B.2	CSI-2 Transmitter Detailed Block Diagram	
B.3	CSI-2 Receiver Detailed Block Diagram	
B.4	Details on the D-PHY implementation	
B.4.1		
B.4.2		
B.4.3		
B.4.4		
Annex C		
C.1	Overview	
C.2	D-PHY Level Error	
C.3	Packet Level Error	
C.4	Protocol Decoding Level Error	
Annex D	1	
D.1	Overview	
D.2	SLM Command Phase	
D.3	SLM Entry Phase	
D.4	SLM Exit Phase	125
Annex E	Jr	
E.1	Predictors	
E.1.1		
E.1.2		
E.2	Encoders	
E.2.1	1	
E.2.2	Coder for 10–7–10 Data Compression	130

Annex F	JPEG Interleaving (informative)	157
E.3.6	Decoder for 12–6–12 Data Compression	153
	Decoder for 12–7–12 Data Compression	
	*	
	Decoder for 12–8–12 Data Compression	
E.3.3	Decoder for 10–6–10 Data Compression	145
E.3.2	Decoder for 10–7–10 Data Compression	143
E.3.1	Decoder for 10–8–10 Data Compression	141
E.3 I	Decoders	141
E.2.6	Coder for 12–6–12 Data Compression	139
	Coder for 12–7–12 Data Compression	
	Coder for 12–8–12 Data Compression	
	Coder for 10–6–10 Data Compression	

Figures

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

Figure 32 One Lane Transmitter and N-Lane Receiver Example for D-PHY	37
Figure 33 M-Lane Transmitter and N-Lane Receiver Example (M <n) d-phy<="" for="" td=""><td>37</td></n)>	37
Figure 34 M-Lane Transmitter and One Lane Receiver Example for D-PHY	38
Figure 35 M-Lane Transmitter and N-Lane Receiver Example (N <m) d-phy<="" for="" td=""><td>38</td></m)>	38
Figure 36 Low Level Protocol Packet Overview	39
Figure 37 Long Packet Structure for D-PHY Physical Layer Option	40
Figure 38 Long Packet Structure for C-PHY Physical Layer Option	41
Figure 39 Packet Header Lane Distribution for C-PHY Physical Layer Option	41
Figure 40 Minimal Filler Byte Insertion Requirements for Three Lane C-PHY	43
Figure 41 Short Packet Structure for D-PHY Physical Layer Option	44
Figure 42 Short Packet Structure for C-PHY Physical Layer Option	44
Figure 43 Data Identifier Byte	45
Figure 44 Logical Channel Block Diagram (Receiver)	45
Figure 45 Interleaved Video Data Streams Examples	46
Figure 46 24-bit ECC Generation Example	47
Figure 47 64-bit ECC Generation on TX Side	51
Figure 48 24-bit ECC Generation on TX Side	51
Figure 49 64-bit ECC on RX Side Including Error Correction	52
Figure 50 24-bit ECC on RX Side Including Error Correction	53
Figure 51 Checksum Transmission Byte Order	53
Figure 52 Checksum Generation for Long Packet Payload Data	54
Figure 53 Definition of 16-bit CRC Shift Register	54
Figure 54 16-bit CRC Software Implementation Example	55
Figure 55 Packet Spacing	56
Figure 56 Multiple Packet Example	58
Figure 57 Single Packet Example	58
Figure 58 Line and Frame Blanking Definitions	59
Figure 59 Vertical Sync Example	60
Figure 60 Horizontal Sync Example	60
Figure 61 General Frame Format Example	61
Figure 62 Digital Interlaced Video Example	62
Figure 63 Digital Interlaced Video with Accurate Synchronization Timing Information	63
Figure 64 Interleaved Data Transmission using Data Type Value	64

Figure 65 Packet Level Interleaved Data Transmission	65
Figure 66 Frame Level Interleaved Data Transmission	66
Figure 67 Interleaved Data Transmission using Virtual Channels	67
Figure 68 Byte Packing Pixel Data to C-PHY Symbol Illustration	70
Figure 69 Frame Structure with Embedded Data at the Beginning and End of the Frame	71
Figure 70 Legacy YUV420 8-bit Transmission	72
Figure 71 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration	73
Figure 72 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1	73
Figure 73 Legacy YUV420 8-bit Frame Format	74
Figure 74 YUV420 8-bit Data Transmission Sequence	74
Figure 75 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration	75
Figure 76 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1	75
Figure 77 YUV420 Spatial Sampling for MPEG 2 and MPEG 4	76
Figure 78 YUV420 8-bit Frame Format	76
Figure 79 YUV420 10-bit Transmission	77
Figure 80 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration	77
Figure 81 YUV420 10-bit Frame Format	78
Figure 82 YUV422 8-bit Transmission	78
Figure 83 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration	79
Figure 84 YUV422 Co-sited Spatial Sampling	79
Figure 85 YUV422 8-bit Frame Format	80
Figure 86 YUV422 10-bit Transmitted Bytes	80
Figure 87 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration	81
Figure 88 YUV422 10-bit Frame Format	81
Figure 89 RGB888 Transmission	82
Figure 90 RGB888 Transmission in CSI-2 Bus Bitwise Illustration	82
Figure 91 RGB888 Frame Format	83
Figure 92 RGB666 Transmission with 18-bit BGR Words	83
Figure 93 RGB666 Transmission on CSI-2 Bus Bitwise Illustration	84
Figure 94 RGB666 Frame Format	84
Figure 95 RGB565 Transmission with 16-bit BGR Words	
Figure 96 RGB565 Transmission on CSI-2 Bus Bitwise Illustration	85
Figure 97 RGB565 Frame Format	85

Figure 98 RGB555 Transmission on CSI-2 Bus Bitwise Illustration	86
Figure 99 RGB444 Transmission on CSI-2 Bus Bitwise Illustration	86
Figure 100 RAW6 Transmission	87
Figure 101 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration	88
Figure 102 RAW6 Frame Format	88
Figure 103 RAW7 Transmission	88
Figure 104 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration	89
Figure 105 RAW7 Frame Format	89
Figure 106 RAW8 Transmission	89
Figure 107 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration	90
Figure 108 RAW8 Frame Format	90
Figure 109 RAW10 Transmission	90
Figure 110 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration	91
Figure 111 RAW10 Frame Format	91
Figure 112 RAW12 Transmission	92
Figure 113 RAW12 Transmission on CSI-2 Bus Bitwise Illustration	92
Figure 114 RAW12 Frame Format	92
Figure 115 RAW14 Transmission	93
Figure 116 RAW14 Transmission on CSI-2 Bus Bitwise Illustration	93
Figure 117 RAW14 Frame Format	93
Figure 118 User Defined 8-bit Data (128 Byte Packet)	94
Figure 119 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration	94
Figure 120 Transmission of User Defined 8-bit Data	94
Figure 121 General/Arbitrary Data Reception	96
Figure 122 RGB888 Data Format Reception	97
Figure 123 RGB666 Data Format Reception	97
Figure 124 RGB565 Data Format Reception	98
Figure 125 RGB555 Data Format Reception	98
Figure 126 RGB444 Data Format Reception	99
Figure 127 YUV422 8-bit Data Format Reception	99
Figure 128 YUV422 10-bit Data Format Reception	100
Figure 129 YUV420 8-bit Legacy Data Format Reception	101
Figure 130 YUV420 8-bit Data Format Reception	102

X

Version 1.3 29-May-2014

Figure 131 YUV420 10-bit Data Format Reception	103
Figure 132 RAW6 Data Format Reception	104
Figure 133 RAW7 Data Format Reception	104
Figure 134 RAW8 Data Format Reception	105
Figure 135 RAW10 Data Format Reception	105
Figure 136 RAW12 Data Format Reception	106
Figure 137 RAW 14 Data Format Reception	106
Figure 138 JPEG8 Data Flow in the Encoder	107
Figure 139 JPEG8 Data Flow in the Decoder	107
Figure 140 EXIF Compatible Baseline JPEG DCT Format	108
Figure 141 Status Information Field in the End of Baseline JPEG Frame	109
Figure 142 Example of TN Image Embedding Inside the Compressed JPEG Data Block	110
Figure 143 JPEG8 Data Format Reception	111
Figure 144 Implementation Example Block Diagram and Coverage	112
Figure 145 CSI-2 Transmitter Block Diagram	113
Figure 146 CSI-2 Receiver Block Diagram	114
Figure 147 D-PHY Level Block Diagram	115
Figure 148 CSI-2 Clock Lane Transmitter	116
Figure 149 CSI-2 Clock Lane Receiver	117
Figure 150 CSI-2 Data Lane Transmitter	118
Figure 151 CSI-2 Data Lane Receiver	119
Figure 152 SLM Synchronization	125
Figure 153 Data Compression System Block Diagram	127
Figure 154 Pixel Order of the Original Image	127
Figure 155 Example Pixel Order of the Original Image	127
Figure 156 Data Type Interleaving: Concurrent JPEG and YUV Image Data	157
Figure 157 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data	158
Figure 158 Example JPEG and YUV Interleaving Use Cases	159

Tables

Table 1 CCI I/O Characteristics	22
Table 2 CCI Timing Specification	24
Table 3 Data Type Classes	46
Table 4 ECC Syndrome Association Matrix	48
Table 5 ECC Parity Generation Rules	49
Table 6 Synchronization Short Packet Data Type Codes	56
Table 7 Generic Short Packet Data Type Codes	57
Table 8 Primary and Secondary Data Formats Definitions	69
Table 9 Generic 8-bit Long Packet Data Types	70
Table 10 YUV Image Data Types	71
Table 11 Legacy YUV420 8-bit Packet Data Size Constraints	72
Table 12 YUV420 8-bit Packet Data Size Constraints	74
Table 13 YUV420 10-bit Packet Data Size Constraints	77
Table 14 YUV422 8-bit Packet Data Size Constraints	78
Table 15 YUV422 10-bit Packet Data Size Constraints	80
Table 16 RGB Image Data Types	81
Table 17 RGB888 Packet Data Size Constraints	82
Table 18 RGB666 Packet Data Size Constraints	83
Table 19 RGB565 Packet Data Size Constraints	84
Table 20 RAW Image Data Types	87
Table 21 RAW6 Packet Data Size Constraints	87
Table 22 RAW7 Packet Data Size Constraints	88
Table 23 RAW8 Packet Data Size Constraints	89
Table 24 RAW10 Packet Data Size Constraints	90
Table 25 RAW12 Packet Data Size Constraints	91
Table 26 RAW14 Packet Data Size Constraints	92
Table 27 User Defined 8-bit Data Types	94
Table 28 Status Data Padding	109
Table 29 JPEG8 Additional Marker Codes Listing	111

Release History

Date	Version	Description
2005-11-29	v1.00	Initial Board-approved release.
2010-11-09	v1.01.00	Board-approved release.
2013-01-22	v1.1	Board approved release.
2014-09-10	v1.2	Board approved release.
2014-10-07	v1.3	Board approved release.

Specification for CSI-2 Version 1.3 29-May-2014

This page intentionally left blank.

1 Introduction

1.1 Scope

- 1 The Camera Serial Interface 2 Specification defines an interface between a peripheral device (camera) and
- 2 a host processor (baseband, application engine). The purpose of this document is to specify a standard
- 3 interface between a camera and a host processor for mobile applications.
- 4 This Revision of the Camera Serial Interface 2 Specification leverages [MIPI01] D-PHY 1.2 and introduces
- 5 [MIPI02] C-PHY 1.0, both with improved skew tolerance and higher data rates. These enhancements
- 6 enable higher interface bandwidth and more flexibility in channel layout. The CSI-2 1.3 Specification was
- 7 designed to ensure interoperability with CSI-2 1.2 when the former uses the D-PHY physical layer. If the
- 8 C-PHY physical layer only is used, backwards compatibility cannot be maintained.
- 9 A host processor in this document refers to the hardware and software that performs essential core
- functions for telecommunication or application tasks. The engine of a mobile terminal includes hardware
- 11 and the functions, which enable the basic operation of the mobile terminal. These include, for example, the
- 12 printed circuit boards, RF components, basic electronics, and basic software, such as the digital signal
- 13 processing software.

1.2 Purpose

- 14 Demand for increasingly higher image resolutions is pushing the bandwidth capacity of existing host
- 15 processor-to-camera sensor interfaces. Common parallel interfaces are difficult to expand, require many
- 16 interconnects and consume relatively large amounts of power. Emerging serial interfaces address many of
- 17 the shortcomings of parallel interfaces while introducing their own problems. Incompatible, proprietary
- 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
- 20 industry standard can slow innovation and inhibit new product market entry.
- 21 CSI-2 provides the mobile industry a standard, robust, scalable, low-power, high-speed, cost-effective
- 22 interface that supports a wide range of imaging solutions for mobile devices.

Specification for CSI-2 Version 1.3 29-May-2014

2 **Terminology**

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

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**

- 41 Lane: A unidirectional, point-to-point, 2- or 3-wire interface used for high-speed serial clock or data
- 42 transmission; the number of wires is determined by the PHY specification in use (i.e. either D-PHY or C-43
- PHY, respectively). A CSI-2 camera interface using the D-PHY physical layer consists of one clock Lane 44 and one or more data Lanes. A CSI-2 camera interface using the C-PHY physical layer consists of one or
- 45 more Lanes, each of which transmits both clock and data information. Note that when describing features
- 46 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. 47
- 48 Packet: A group of bytes organized in a specified way to transfer data across the interface. All packets have
- 49 a minimum specified set of components. The byte is the fundamental unit of data from which packets are
- 50 made.
- 51 Payload: Application data only - with all sync, header, ECC and checksum and other protocol-related
- 52 information removed. This is the "core" of transmissions between application processor and peripheral.
- 53 **Sleep Mode:** Sleep mode (SLM) is a leakage level only power consumption mode.
- 54 Transmission: The time during which high-speed serial data is actively traversing the bus. A transmission
- 55 is comprised of one or more packets. A transmission is bounded by SoT (Start of Transmission) and EoT
- 56 (End of Transmission) at beginning and end, respectively.
- 57 Virtual Channel: Multiple independent data streams for up to four peripherals are supported by this
- 58 Specification. The data stream for each peripheral is a Virtual Channel. These data streams may be
- 59 interleaved and sent as sequential packets, with each packet dedicated to a particular peripheral or channel.
- 60 Packet protocol includes information that links each packet to its intended peripheral.

29-May-2014

2.3 Abbreviations

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

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

2.4 Acronyms

- 63 BER Bit Error Rate
- 64 CCI Camera Control Interface
- 65 CIL Control and Interface Logic
- 66 CRC Cyclic Redundancy Check
- 67 CSI Camera Serial Interface
- 68 CSPS Chroma Shifted Pixel Sampling
- 69 DDR Dual Data Rate
- 70 DI Data Identifier
- 71 DT Data Type
- 72 ECC Error Correction Code
- 73 EoT End of Transmission
- 74 EXIF Exchangeable Image File Format
- 75 FE Frame End
- 76 FS Frame Start
- 77 HS High Speed; identifier for operation mode
- 78 HS-RX High-Speed Receiver
- 79 HS-TX High-Speed Transmitter
- 80 I2C Inter-Integrated Circuit
- 81 JFIF JPEG File Interchange Format
- 82 JPEG Joint Photographic Expert Group
- 83 LE Line End
- 84 LLP Low Level Protocol
- 85 LS Line Start
- 86 LSB Least Significant Bit
- 87 LSS Least Significant Symbol
- 88 LP Low-Power; identifier for operation mode
- 89 LP-RX Low-Power Receiver (Large-Swing Single Ended)
- 90 LP-TX Low-Power Transmitter (Large-Swing Single Ended)
- 91 MSB Most Significant Bit
- 92 MSS Most Significant Symbol

Specification for CSI-2 Version 1.3
29-May-2014

93	PF	Packet Footer
94	PH	Packet Header
95	PI	Packet Identifier
96	PT	Packet Type
97	PHY	Physical Layer
98	PPI	PHY Protocol Interface
99	RGB	Color representation (Red, Green, Blue)
100	RX	Receiver
101	SCL	Serial Clock (for CCI)
102	SDA	Serial Data (for CCI)
103	SLM	Sleep Mode
104	SoT	Start of Transmission
105	TX	Transmitter
106	ULPS	Ultra low Power State
107	VGA	Video Graphics Array
108	YUV	Color representation (Y for luminance, U & V for chrominance)

29-May-2014

3 References

109 110	[NXP01]	UM10204, <i>I2C-bus specification and user manual</i> , Revision 03, NXP B.V., 19 June 2007.
111 112	[MIPI01]	MIPI Alliance Specification for D PHY, version 1.2, MIPI Alliance, Inc., 10 September 2014.
113 114	[MIPI02]	MIPI Alliance Specification for C PHY, version 1.0, MIPI Alliance, Inc., 07 October 2014.

Specification for CSI-2 Version 1.3
29-May-2014

This page intentionally left blank.

115116

117

118

119

120

121

122

123

124

125

126

127

128

4 Overview of CSI-2

The CSI-2 Specification defines standard data transmission and control interfaces between transmitter and receiver. Two high-speed serial data transmission interface options are defined. The first option, referred to in this specification as the "D-PHY physical layer option", is a unidirectional differential interface with one 2-wire clock Lane and one or more 2-wire data Lanes. The physical layer of this interface is defined by the MIPI Alliance Specification for D-PHY [MIPI01]. Figure 1 illustrates the connections for this option between a CSI-2 transmitter and receiver, which typically are a camera module and a receiver module, part of the mobile phone engine.

The second high-speed data transmission interface option, referred to in this specification as the "C-PHY physical layer option", consists of one or more unidirectional 3-wire serial data Lanes, each of which has its own embedded clock. The physical layer of this interface is defined by the *MIPI Alliance Specification for C-PHY* [MIPI02]. Figure 2 illustrates the CSI transmitter and receiver connections for this option.

The Camera Control Interface (CCI) for both physical layer options is a bi-directional control interface compatible with the I2C standard [NXP01].

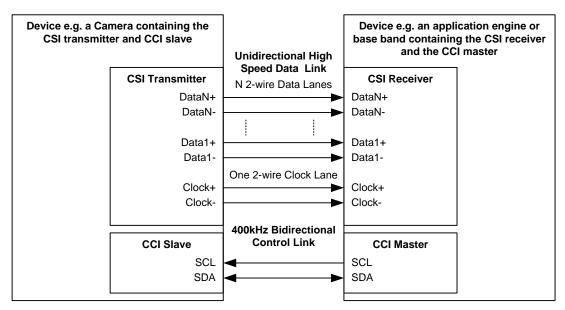


Figure 1 CSI-2 and CCI Transmitter and Receiver Interface for D-PHY

Specification for CSI-2 Version 1.3

29-May-2014

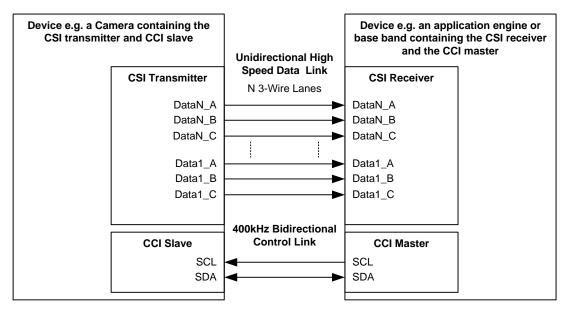


Figure 2 CSI-2 and CCI Transmitter and Receiver Interface for C-PHY

8

132

134

135

136

137

138

139

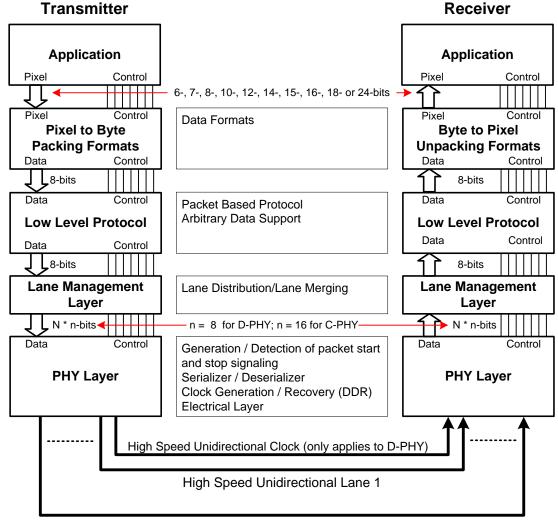
140

141

142

143

131 5 CSI-2 Layer Definitions



High Speed Unidirectional Lane N

Figure 3 CSI-2 Layer Definitions

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

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

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

The PHY layer is described in [MIPI01] and [MIPI02].

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

- Pixel/Byte Packing/Unpacking Layer. The CSI-2 specification supports image applications with varying pixel formats from six to twenty-four bits per pixel. In the transmitter this layer packs pixels from the Application layer into bytes before sending the data to the Low Level Protocol layer. In the receiver this layer unpacks bytes from the Low Level Protocol layer into pixels before sending the data to the Application layer. Eight bits per pixel data is transferred unchanged by this layer.
- Low Level Protocol. The Low Level Protocol (LLP) includes the means of establishing bit-level and byte-level synchronization for serial data transferred between SoT (Start of Transmission) and EoT (End of Transmission) events and for passing data to the next layer. The minimum data granularity of the LLP is one byte. The LLP also includes assignment of bit-value interpretation within the byte, i.e. the "Endian" assignment.
- Lane Management. CSI-2 is Lane-scalable for increased performance. The number of data Lanes is not limited by this specification and may be chosen depending on the bandwidth requirements of the application. The transmitting side of the interface distributes ("distributor" function) bytes from the outgoing data stream to one or more Lanes. On the receiving side, the interface collects bytes from the Lanes and merges ("merger" function) them together into a recombined data stream that restores the original stream sequence. For the C-PHY physical layer option, this layer exclusively distributes or collects byte pairs (i.e. 16-bits) to or from the data Lanes.
- Data within the Protocol layer is organized as packets. The transmitting side of the interface appends header and error-checking information on to data to be transmitted at the Low Level Protocol layer. On the receiving side, the header is stripped off at the Low Level Protocol layer and interpreted by corresponding logic in the receiver. Error-checking information may be used to test the integrity of incoming data.
- **Application Layer.** This layer describes higher-level encoding and interpretation of data contained in the data stream and is beyond the scope of this specification. The CSI-2 Specification describes the mapping of pixel values to bytes.
- The normative sections of the Specification only relate to the external part of the Link, e.g. the data and bit patterns that are transferred across the Link. All internal interfaces and layers are purely informative.

Version 1.3 Specification for CSI-2

29-May-2014

177 6 Camera Control Interface (CCI)

- 178 CCI is a two-wire, bi-directional, half duplex, serial interface for controlling the transmitter. CCI is
- compatible with the fast mode variant of the I2C interface. CCI shall support 400kHz operation and 7-bit
- 180 Slave Addressing.
- A CSI-2 receiver shall be configured as a master and a CSI-2 transmitter shall be configured as a slave on
- the CCI bus. CCI is capable of handling multiple slaves on the bus. However, multi-master mode is not
- supported by CCI. Any I2C commands that are not described in this section shall be ignored and shall not
- 184 cause unintended device operation. Note that the terms master and slave, when referring to CCI, should not
- be confused with similar terminology used for D-PHY's operation; they are not related.
- Typically, there is a dedicated CCI interface between the transmitter and the receiver.
- 187 CCI is a subset of the I2C protocol, including the minimum combination of obligatory features for I2C
- 188 slave devices specified in the I2C specification. Therefore, transmitters complying with the CCI
- specification can also be connected to the system I2C bus. However, care must be taken so that I2C masters
- do not try to utilize those I2C features that are not supported by CCI masters and CCI slaves
- Each CCI transmitter may have additional features to support I2C, but that is dependent on implementation.
- 192 Further details can be found on a particular device's data sheet.
- 193 This Specification does not attempt to define the contents of control messages sent by the CCI master. As
- such, it is the responsibility of the CSI-2 implementer to define a set of control messages and corresponding
- frame timing and I2C latency requirements, if any, that must be met by the CCI master when sending such
- control messages to the CCI slave.
- 197 The CCI defines an additional data protocol layer on top of I2C. The data protocol is presented in the
- 198 following sections.

199 6.1 Data Transfer Protocol

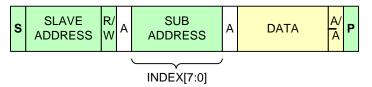
- 200 The data transfer protocol is according to I2C standard. The START, REPEATED START and STOP
- 201 conditions as well as data transfer protocol are specified in *The I^2C Specification* [NXP01].

202 6.1.1 Message Type

- A basic CCI message consists of START condition, slave address with read/write bit, acknowledge from
- slave, sub address (index) for pointing at a register inside the slave device, acknowledge signal from slave,
- in write operation data byte from master, acknowledge/negative acknowledge from slave and STOP
- 206 condition. In read operation data byte comes from slave and acknowledge/negative acknowledge from
- master. This is illustrated in Figure 4.
- The slave address in the CCI is 7-bit.
- The CCI supports 8-bit index with 8-bit data or 16-bit index with 8-bit data. The slave device in question
- 210 defines what message type is used.

Specification for CSI-2 Version 1.3
29-May-2014

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



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

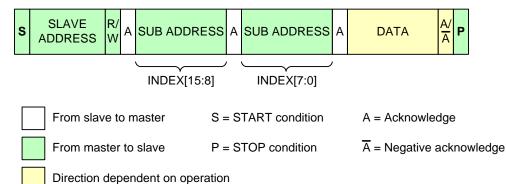


Figure 4 CCI Message Types

6.1.2 Read/Write Operations

211

212

213

214

215

216

217

218

219

220

221

222

223

224

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

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

6.1.2.1 Single Read from Random Location

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

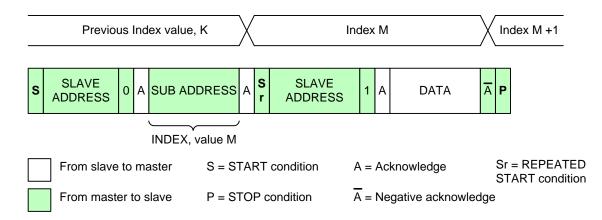


Figure 5 CCI Single Read from Random Location

Version 1.3 29-May-2014

225

226

227

228

229

230

231

232

233

234

235

236

237

6.1.2.2 Single Read from the Current Location

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

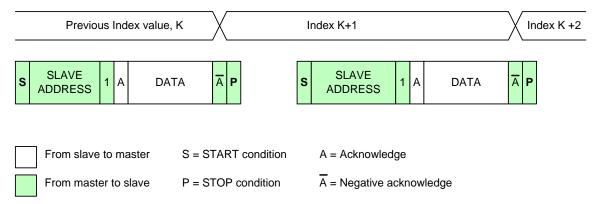


Figure 6 CCI Single Read from Current Location

6.1.2.3 Sequential Read Starting from a Random Location

The sequential read starting from a random location is illustrated in Figure 7. The master does a dummy write to the desired index, issues a repeated start condition after an acknowledge from the slave and then addresses the slave again with a read operation. If a master issues an acknowledge after received data it acts as a signal to the slave that the read operation continues from the next index. When the master has read the last data byte it issues a negative acknowledge and stop condition.

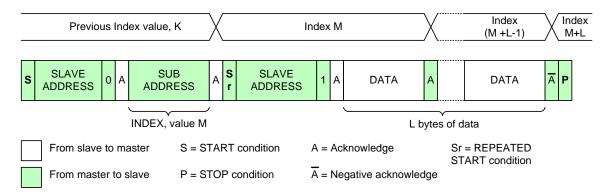


Figure 7 CCI Sequential Read Starting from a Random Location

6.1.2.4 Sequential Read Starting from the Current Location

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

Specification for CSI-2 Version 1.3
29-May-2014

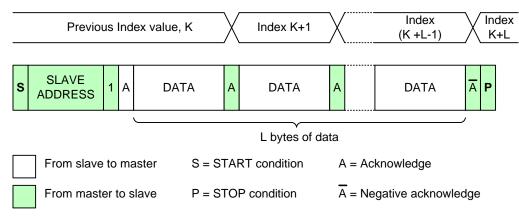


Figure 8 CCI Sequential Read Starting from the Current Location

6.1.2.5 Single Write to a Random Location

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

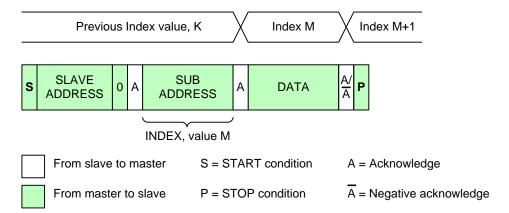


Figure 9 CCI Single Write to a Random Location

247 6.1.2.6 Sequential Write

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

241

242

243

244

245

Version 1.3 Specification for CSI-2

29-May-2014

250

251

254

255

260

262

263

264

265

266

267

268

277

278

279

280

281

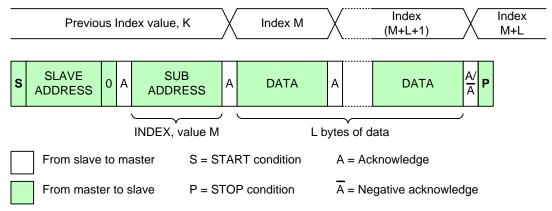


Figure 10 CCI Sequential Write Starting from a Random Location

CCI Slave Addresses 6.2

252 For camera modules having only raw Bayer output the 7-bit slave address should be 011011Xb, where X =253

0 or 1. For all other camera modules the 7-bit slave address should be 011110Xb.

CCI Multi-Byte Registers 6.3

6.3.1 Overview

256 Peripherals contain a wide range of different register widths for various control and setup purposes. The 257 CSI-2 Specification supports the following register widths:

- 258 • 8-bit – generic setup registers
- 259 • 16-bit – parameters like line-length, frame-length and exposure values
 - 32-bit high precision setup values
- 261 • 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 multibyte registers will not be contiguous. For example, a 32-bit register with its first byte at address 0x8000 can be read by means of a sequential read of four bytes, starting at random address 0x8000. If there is an additional 4-byte register with its first byte at 0x8004, it could then be accessed using a four-byte Sequential Read from the Current Location protocol.

269 The motivation for a general multi-byte protocol rather than fixing the registers at 16-bits width is 270 flexibility. The protocol described in the following paragraphs provides a way of transferring 16-bit, 32-bit 271 or 64-bit values over a 16-bit index, 8-bit data, two-wire serial link while ensuring that the bytes of data 272 transferred for a multi-byte register value are always consistent (temporally coherent).

273 Using this protocol a single CCI message can contain one, two or all of the different register widths used 274 within a device.

275 The MS byte of a multi-byte register shall be located at the lowest address and the LS byte at the highest 276 address.

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

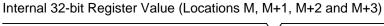
Specification for CSI-2 Version 1.3

29-May-2014

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

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

- For a Write operation, the updating of the register shall be deferred to a time when the last bit of the last byte has been received
- For a Read operation, the value read shall reflect the status of all bytes at the time that the first bit of the first byte has been read
- Section 6.3.3 describes example behavior for the re-timing of multi-byte register accesses.
- Without re-timing, data may be corrupted as illustrated in Figure 11 and Figure 12.



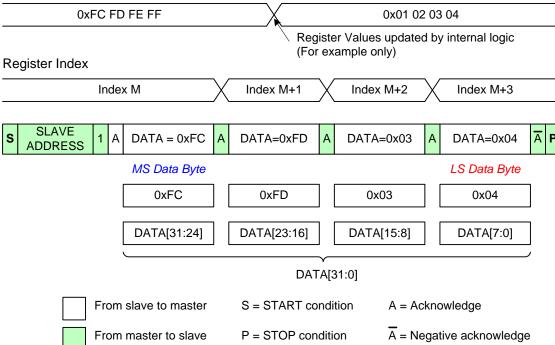


Figure 11 Corruption of a 32-bit Wide Register during a Read Message

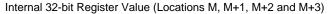
285

286

287

288

289



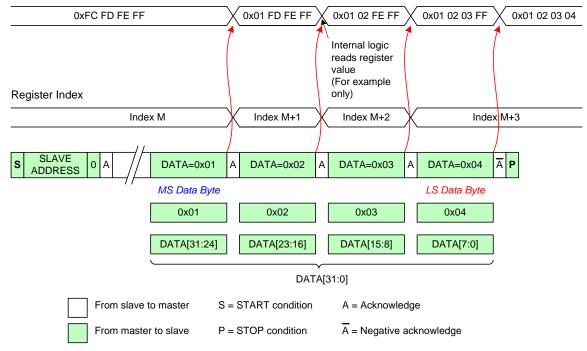


Figure 12 Corruption of a 32-bit Wide Register during a Write Message

6.3.2 The Transmission Byte Order for Multi-byte Register Values

This is a normative section.

293

294

296

297

298

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

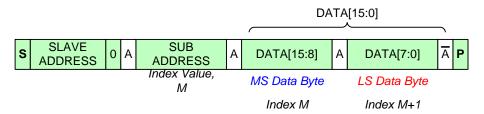


Figure 13 Example 16-bit Register Write

Specification for CSI-2 Version 1.3 29-May-2014

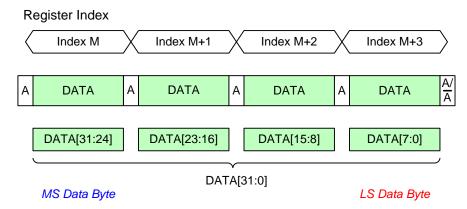


Figure 14 Example 32-bit Register Write (address not shown)

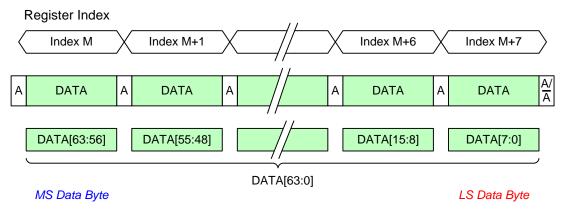


Figure 15 Example 64-bit Register Write (address not shown)

6.3.3 Multi-Byte Register Protocol

This is an informative section.

299

300

301

302

303

304

305

306

307

308

309

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

6.3.3.1 Reading Multi-byte Registers

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

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

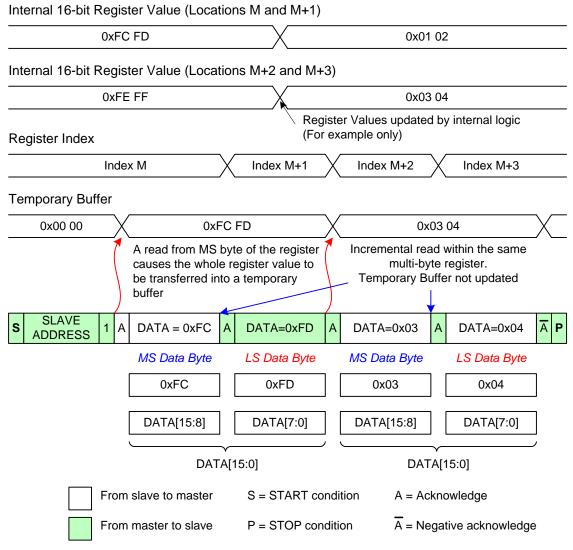


Figure 16 Example 16-bit Register Read

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

- Examples of when the temporary buffer is updated are as follows:
- The MS byte of a register is accessed

312

313

314

315

316

317

318

319

- The index has crossed a multi-byte register boundary
- Successive single byte reads from the same index location
- The index value for the byte about to be read is the same or less than the previous index
- Unless the contents of a multi-byte register are accessed in an incremental manner the values read back are not guaranteed to be consistent.
- The contents of the temporary buffer are reset to zero by START and STOP conditions.

Specification for CSI-2 Version 1.3 29-May-2014

Internal 32-bit Register Value (Locations M, M+1, M+2 and M+3) 0xFC FD FE FF 0x01 02 03 04 Register Values updated by internal logic (For example only) Register Index Index M Index M+1 Index M+2 Index M+3 **Temporary Buffer** 0x00 00 00 00 0xFC FD FE FF A read from MS byte of the register Incremental read within the same causes the whole register value to multi-byte register. Temporary Buffer not updated be transferred into a temporary **SLAVE** DATA = 0xFCDATA=0xFD DATA=0xFE DATA=0xFF **ADDRESS** MS Data Byte LS Data Byte 0xFC 0xFD 0xFE 0xFF DATA[31:24] DATA[23:16] DATA[15:8] DATA[7:0] DATA[31:0] From slave to master S = START condition A = Acknowledge

Figure 17 Example 32-bit Register Read

P = STOP condition

A = Negative acknowledge

6.3.3.2 Writing Multi-byte Registers

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

Figure 18 and Figure 19 illustrate multi-byte register write operations.

From master to slave

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.

324

325

326

327

332

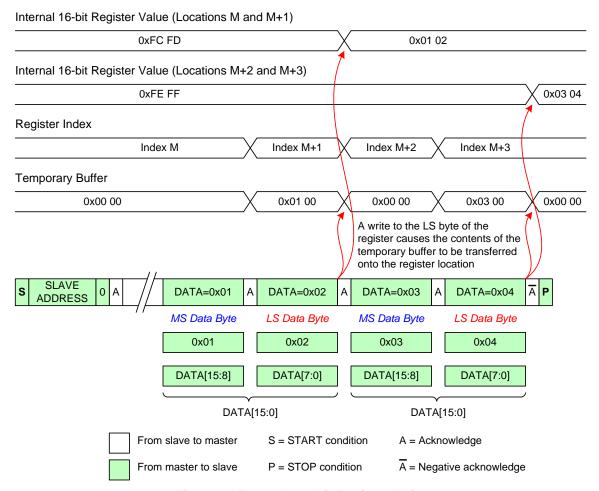


Figure 18 Example 16-bit Register Write

Confidential

Specification for CSI-2 Version 1.3 29-May-2014

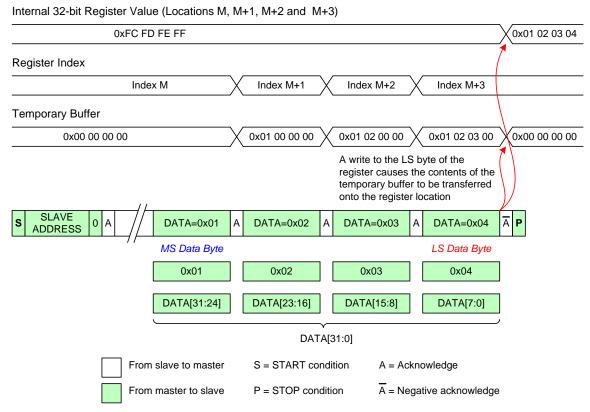


Figure 19 Example 32-bit Register Write

6.4 Electrical Specifications and Timing for I/O Stages

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

Table 1 CCI I/O Characteristics

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

333

334335

336

Parameter	Symbol Standard-mode			Fast	Unit	
		Min.	Max.	Min.	Max.	
HIGH level output voltage	Vон	N/A	N/A	0.8V _{DD}		V
Output fall time from V _{IHmin} to V _{ILmax} with bus capacitance from 10 pF to 400 pF	t _{OF}	-	250	20+0.1C _B Note 2	250	ns
Pulse width of spikes which shall be suppressed by the input filter	tsp	N/A	N/A	0	50	ns
Input current each I/O pin with an input voltage between 0.1 V _{DD} and 0.9 V _{DD}	I ₁	-10	10	-10 Note 3	10 Note 3	μА
Input/Output capacitance (SDA)	C _{I/O}	-	8	-	8	pF
Input capacitance (SCL)	CI	-	6	-	6	pF

Note:

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

Table 2 CCI Timing Specification

Parameter	Symbol	Stand	ard-mode	Fast	Fast-mode		
		Min.	Max.	Min.	Max.		
SCL clock frequency	fscL	0	100	0	400	kHz	
Hold time (repeated) START condition. After this period, the first clock pulse is generated	thd;sta	0.4	-	0.6	-	μs	
LOW period of the SCL clock	tLOW	4.7	-	1.3	-	μs	
HIGH period of the SCL clock	t _{HIGH}	4.0	-	0.6	-	μs	
Setup time for a repeated START condition	tsu;sta	4.7	-	0.6	-	μs	
Data hold time	t _{HD;DAT}	0 Note 2	3.45 Note 3	0 Note 2	0.9 Note 3	μs	
Data set-up time	tsu;dat	250	-	100 Note 4	-	ns	
Rise time of both SDA and SCL signals	t _R	-	1000	20+0.1C _B Note 5	300	ns	
Fall time of both SDA and SCL signals	tF	-	300	20+0.1C _B Note 5	300	ns	
Set-up time for STOP condition	tsu;sto	4.0	-	0.6	-	μs	
Bus free time between a STOP and START condition	tBUF	4.7	-	1.3	-	μs	
Capacitive load for each bus line	Св	-	400	-	400	pF	
Noise margin at the LOW level for each connected device (including hysteresis)	V _{nL}	0.1V _{DD}	-	0.1V _{DD}	-	V	
Noise margin at the HIGH level for each connected device (including hysteresis)	V _{nH}	0.2V _{DD}	-	0.2V _{DD}	-	V	

Note:

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

The CCI timing is illustrated in Figure 20.

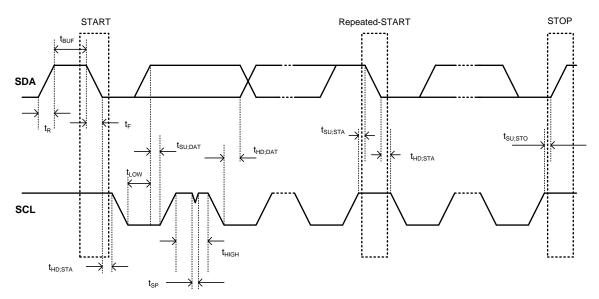


Figure 20 CCI Timing

Specification for CSI-2 Version 1.3

29-May-2014

7 Physical Layer

- 344 The CSI-2 lane management layer interfaces with the D-PHY and/or C-PHY physical layers described in
- [MIPI01] and [MIPI02], respectively. A device shall implement either the C-PHY 1.0 or the D-PHY 1.2
- 346 physical layer and may implement both. A practical constraint is that the PHY technologies used at both
- ends of the Link need to match: a D-PHY transmitter cannot operate with a C-PHY receiver, or vice versa.

7.1 D-PHY Physical Layer Option

- 349 The D-PHY physical layer for a CSI-2 implementation is composed of a number of unidirectional data
- Lanes and one clock Lane. All CSI-2 transmitters and receivers implementing the D-PHY physical layer
- 351 shall support continuous clock behavior on the Clock Lane, and optionally may support non-continuous
- 352 clock behavior.
- 353 For continuous clock behavior the Clock Lane remains in high-speed mode, generating active clock signals
- between the transmission of data packets.
- For non-continuous clock behavior the Clock Lane enters the LP-11 state between the transmission of data
- 356 packets.

343

348

362

- 357 The minimum D-PHY physical layer requirement for a CSI-2 transmitter is
- Data Lane Module: Unidirectional master, HS-TX, LP-TX and a CIL-MFEN function
- Clock Lane Module: Unidirectional master, HS-TX, LP-TX and a CIL-MCNN function
- The minimum D-PHY physical layer requirement for a CSI-2 receiver is
- Data Lane Module: Unidirectional slave, HS-RX, LP-RX, and a CIL-SFEN function
 - Clock Lane Module: Unidirectional slave, HS-RX, LP-RX, and a CIL-SCNN function
- 363 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
- 366 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.
- 368 Since deskew calibration is only valid at a given transmit frequency:
- 369 For initial calibration sequence the Transmitter shall be programmed with the desired frequency for
- 370 calibration. It will then transmit the deskew calibration pattern and the Receiver will autonomously detect
- this pattern and tune the deskew function to achieve optimum performance.
- For any transmitter frequency changes the deskew calibration shall be rerun.
- 373 Some transmitters and/or receiver may require deskew calibration to be rerun periodically and it is
- 374 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
- 376 transmitter not supporting the deskew calibration pattern the receiver may be instructed to bypass the
- deskew mechanism.

378 7.2 C-PHY Physical Layer Option

- The C-PHY physical layer for a CSI-2 implementation is composed of one or more unidirectional Lanes.
- The minimum C-PHY physical layer requirement for a CSI-2 transmitter Lane module is:
- Unidirectional master, HS-TX, LP-TX and a CIL-MFEN function
- Support for Sync Word insertion during data payload transmission
- The minimum C-PHY physical layer requirement for a CSI-2 receiver Lane module is:

Version 1.3 Specification for CSI-2

29-May-2014

• Unidirectional slave, HS-RX, LP-RX, and a CIL-SFEN function

• Support for Sync Word detection during data payload reception

386 All CSI-2 implementations supporting the C-PHY physical layer option shall support forward escape ULPS

on all C-PHY Lanes.

8 Multi-Lane Distribution and Merging

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

Conceptually, between the PHY and higher functional layers is a layer that handles multi-Lane configurations. As shown in Figure 21 and Figure 22 for the D-PHY and C-PHY physical layer options, respectively, the CSI-2 transmitter incorporates a Lane Distribution Function (LDF) which accepts a sequence of packet bytes from the low level protocol layer and distributes them across N Lanes, where each Lane is an independent unit of physical-layer logic (serializers, etc.) and transmission circuitry. Similarly, as shown in Figure 23 and Figure 24 for the D-PHY and C-PHY physical layer options, respectively, the CSI-2 receiver incorporates a Lane Merging Function (LMF) which collects incoming bytes from N Lanes and consolidates (merges) them into complete packets to pass into the packet decomposer in the receiver's low level protocol layer.

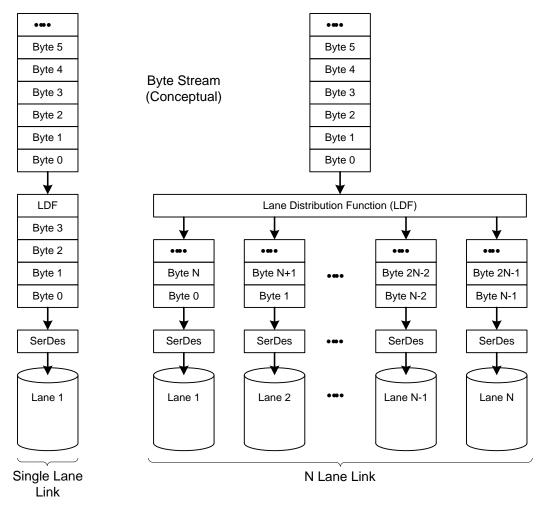


Figure 21 Conceptual Overview of the Lane Distributor Function for D-PHY

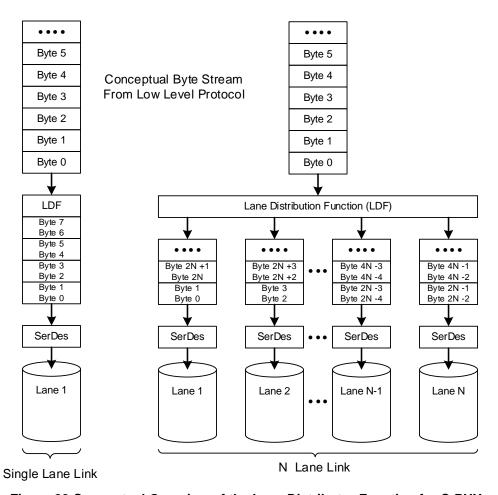


Figure 22 Conceptual Overview of the Lane Distributor Function for C-PHY

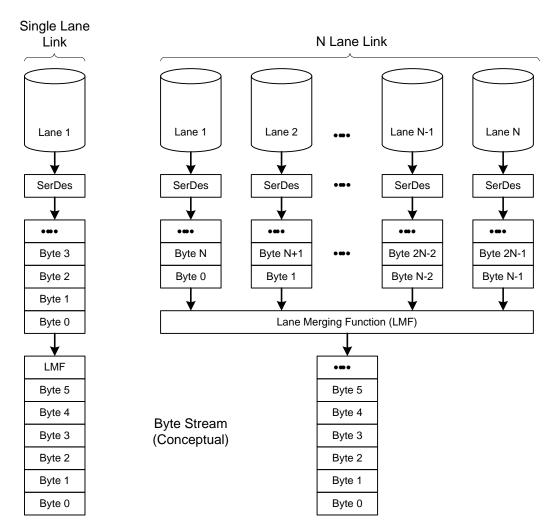


Figure 23 Conceptual Overview of the Lane Merging Function for D-PHY

407 408

409

410

411

412

413 414

415

416

417

418

419

420

421

422

423

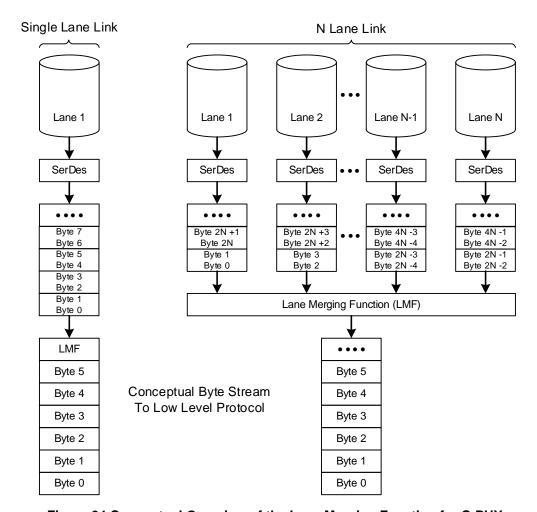


Figure 24 Conceptual Overview of the Lane Merging Function for C-PHY

The Lane distributor takes a transmission of arbitrary byte length, buffers up N*b bytes (where N= number of Lanes and b=1 or 2 for the D-PHY or C-PHY physical layer option, respectively), and then sends groups of N*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 25, Figure 26, Figure 27, and Figure 28:

- 2-Lane system (Figure 25): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to Lane 1, byte 3 goes to Lane 2, byte 4 goes to Lane 1, and so on.
- 3-Lane system (Figure 26): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to Lane 3, byte 3 goes to Lane 1, byte 4 goes to Lane 2, and so on.
- N-Lane system (Figure 27): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte N-1 goes to Lane N, byte N goes to Lane 1, byte N+1 goes to Lane 2, and so on.
- N-lane system (Figure 28) with N>4 short packet (4 bytes) transmission: byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 goes to Lane 3, byte 3 goes to Lane 4, and Lanes 5 to N do not receive bytes and stay in LPS state.

Specification for CSI-2 Version 1.3

29-May-2014

- At the end of the transmission, there may be "extra" bytes since the total byte count may not be an integer multiple of the number of Lanes, N. One or more Lanes may send their last bytes before the others. The Lane distributor, as it buffers up the final set of less-than-N bytes in parallel for sending to N data Lanes, de-asserts 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.
- 430 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:

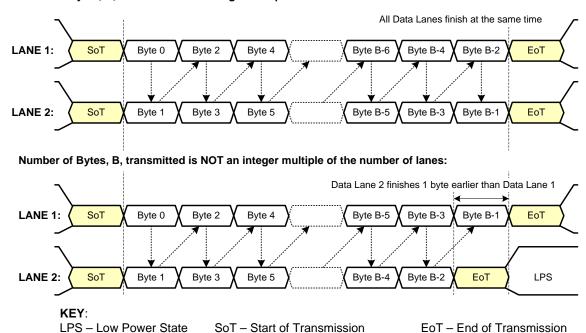
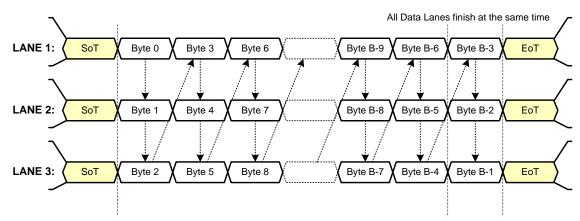


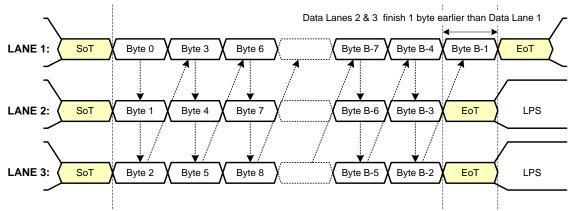
Figure 25 Two Lane Multi-Lane Example for D-PHY

Confidential

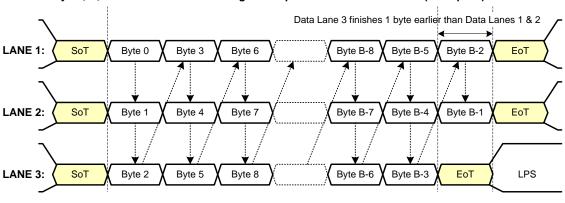
Number of Bytes, B, transmitted is an integer multiple of the number of lanes:



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:

437

LPS - Low Power State

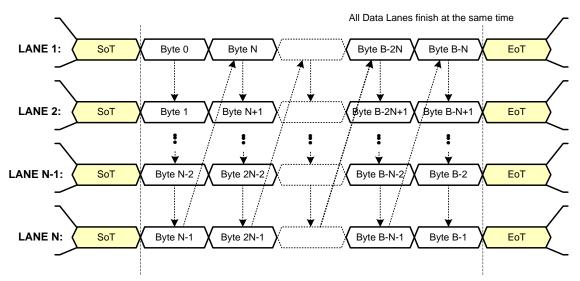
SoT - Start of Transmission

EoT - End of Transmission

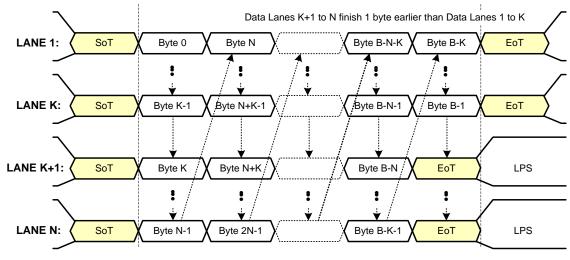
Figure 26 Three Lane Multi-Lane Example for D-PHY

Specification for CSI-2 Version 1.3 29-May-2014

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



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



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

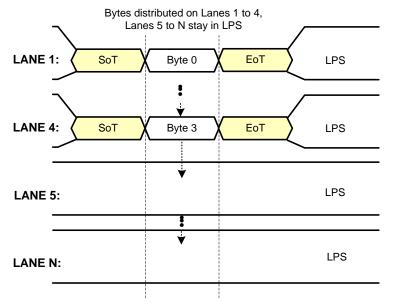
EoT - End of Transmission

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

Version 1.3 Specification for CSI-2

29-May-2014

Short packet of 4 bytes Transmitted on N lanes > 4



KEY:

439

440 441

442

443

444

445

446

447

448

449

450

451

LPS - Low Power State

SoT – Start of Transmission

EoT - End of Transmission

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

8.2 Lane Distribution for the C-PHY Physical Layer Option

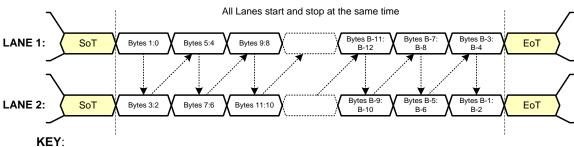
Examples are shown in Figure 29 and Figure 30:

- 2-Lane system (Figure 29): bytes 1 and 0 of the packet are sent as a 16-bit word to the Lane 1 C-PHY module, bytes 3 and 2 are sent to Lane 2, bytes 5 and 4 are sent to Lane 1, bytes 7 and 6 are sent to Lane 2, bytes 9 and 8 are sent to Lane 1, and so on.
- 3-Lane system (Figure 30): bytes 1 and 0 of the packet are sent as a 16-bit word to the Lane 1 C-PHY module, bytes 3 and 2 are sent to Lane 2, bytes 5 and 4 are sent to Lane 3, bytes 7 and 6 are sent to Lane 1, bytes 9 and 8 are sent to Lane 2, and so on.

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

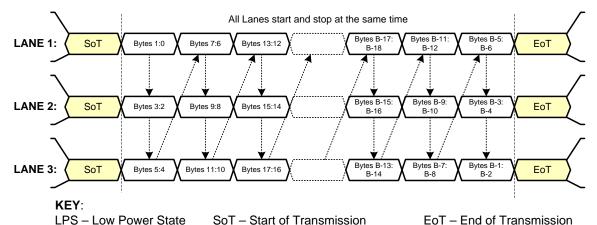
- For an N-Lane transmitter, the C-PHY module for Lane n $(1 \le n \le N)$ shall transmit the following sequence of {ms byte : ls byte} byte pairs from a B-byte packet generated by the low level protocol layer: {Byte 2*(k*N+n)-1 : Byte 2*(k*N+n)-2}, for k=0,1,2,...,B/(2N)-1, where Byte 0 is the first byte in the packet. The low level protocol shall guarantee that B is an integer multiple of 2N.
- That is, at the end of the packet transmission, there shall be no "extra" bytes since the total byte count is always an even multiple of the number of Lanes, N. The Lane distributor, after sending the final set of 2N bytes in parallel to the N Lanes, simultaneously de-asserts its "valid data" signal to all Lanes, signaling to each C-PHY Lane module that it may start its EoT sequence.
- Each C-PHY Lane module operates autonomously, but packet data transmission starts and stops at the same time on all Lanes.
- The N C-PHY receiver modules on the receiving end of the link collect byte pairs in parallel, and feed them into the Lane-merging layer. This reconstitutes the original sequence of bytes in the transmission, which can then be partitioned into individual packets for the packet decoder layers.

Specification for CSI-2 Version 1.3 29-May-2014



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

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



EFS - Low Fower State Sof - Start of Haristinission Lot - Life

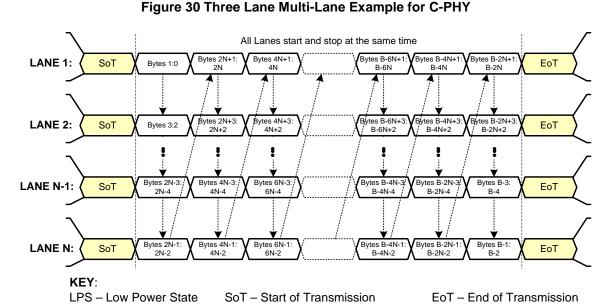


Figure 31 General N-Lane Multi-Lane Distribution for C-PHY

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.

465

466

467

468 469

Version 1.3 29-May-2014

471 An "N" data Lane receiver shall be connected with an "M" data Lane transmitter, by CCI configuration of 472 the Lane distribution and merging layers within the CSI-2 transmitter and receiver when more than one data 473 Lane is used. Thus, if M<=N a receiver with N data Lanes shall work with transmitters with M data Lanes. 474 Likewise, if M>=N a transmitter with M Lanes shall work with receivers with N data Lanes. Transmitter 475

Lanes 1 to M shall be connected to the receiver Lanes 1 to N.

476 Two cases:

477

478

479

480

481

482

- If M<=N then there is no loss of performance the receiver has sufficient data Lanes to match the transmitter (Figure 32 and Figure 33).
- If M> N then there may be a loss of performance (e.g. frame rate) as the receiver has fewer data Lanes than the transmitter (Figure 34 and Figure 35).
- Note that while the examples shown are for the D-PHY physical layer option, the C-PHY physical layer option is handled similarly, except there is no clock Lane.

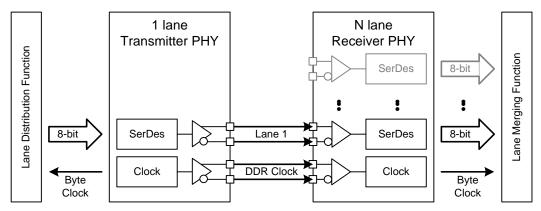


Figure 32 One Lane Transmitter and N-Lane Receiver Example for D-PHY

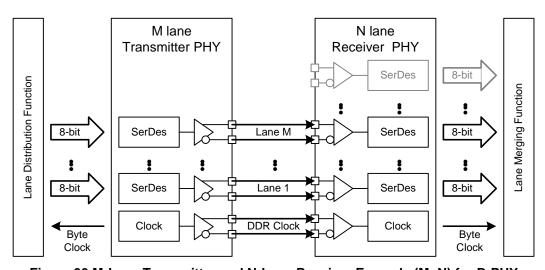


Figure 33 M-Lane Transmitter and N-Lane Receiver Example (M<N) for D-PHY

483

Specification for CSI-2 Version 1.3 29-May-2014

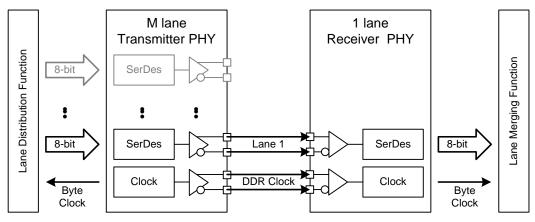


Figure 34 M-Lane Transmitter and One Lane Receiver Example for D-PHY

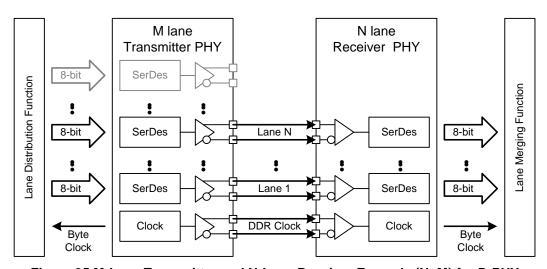


Figure 35 M-Lane Transmitter and N-Lane Receiver Example (N<M) for D-PHY

485

Version 1.3 29-May-2014

487

491

492

493

496

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

9 Low Level Protocol

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

Low Level Protocol Features:

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

DATA:

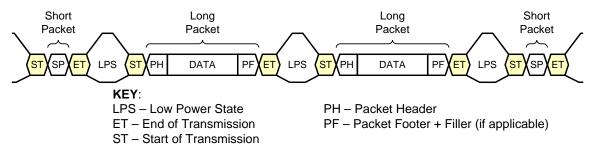


Figure 36 Low Level Protocol Packet Overview

9.1 Low Level Protocol Packet Format

As shown in Figure 36, two packet structures are defined for low-level protocol communication: Long packets and Short packets. The format and length of Short and Long Packets depends 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 37 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 0x37. See Table 3 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 three elements: an 8-bit Data Identifier, a 16-bit Word Count field and an 8-bit ECC. The Packet footer has one element, a 16-bit checksum (CRC). See Section 9.2 through Section 9.5 for further descriptions of the packet elements.

Specification for CSI-2 Version 1.3
29-May-2014

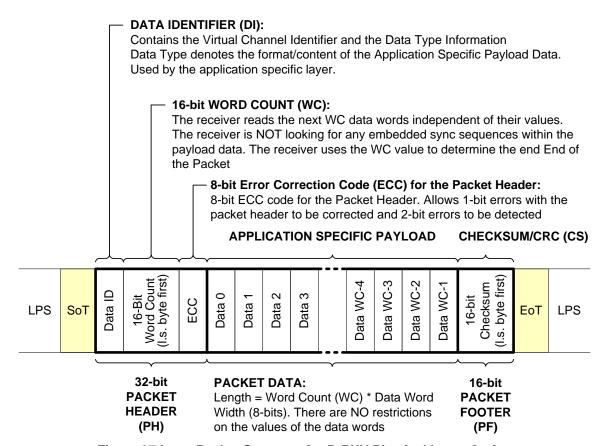


Figure 37 Long Packet Structure for D-PHY Physical Layer Option

Figure 38 shows the Long Packet structure for the C-PHY physical layer option; it shall consist of four elements: a Packet Header (PH), an application specific Data Payload with a variable number of 8-bit data words, a 16-bit Packet Footer (PF), and zero or more Filler bytes (FILLER). The Packet Header is 6N x 16-bits long, where N is the number of C-PHY physical layer Lanes. As shown in Figure 38, the Packet Header consists of two identical 6N-byte halves, where each half consists of N sequential copies of each of the following fields: a 16-bit field containing eight Reserved bits plus the 8-bit Data Identifier (DI); the 16-bit Packet Data Word Count (WC); and a 16-bit Packet Header checksum (PH-CRC) which is computed over the previous four bytes. The value of each Reserved bit shall be zero. The Packet Footer consists of a 16-bit checksum (CRC) computed over the Packet Data using the same CRC polynomial as the Packet Header CRC and the Packet Footer used in the D-PHY physical layer option. Packet Filler bytes are inserted after the Packet Footer, if needed, to ensure that the Packet Footer ends on a 16-bit word boundary and that each C-PHY physical layer Lane transports the same number of 16-bit words (i.e. byte pairs).

16-bit words: FC may be 0

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

Figure 38 Long Packet Structure for C-PHY Physical Layer Option

As shown in Figure 39, the Packet Header structure depicted in Figure 38 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.

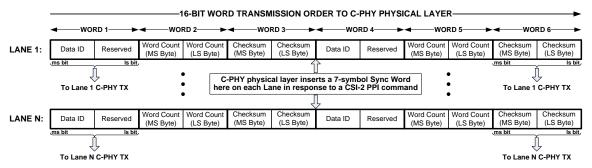


Figure 39 Packet Header Lane Distribution for C-PHY Physical Layer Option

For both physical layer options, the 8-bit Data Identifier field defines the Virtual Channel for the data and the Data Type for the application specific payload data.

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 Error Correction Code (ECC) byte allows single-bit errors to be corrected and 2-bit errors to be detected in the Packet Header. This includes both the Data Identifier value and the Word Count value.

The ECC byte 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, 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 39) 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
- 552 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
- 556 bytes.
- For both physical layer options, once the CSI-2 receiver has read the Data Payload, it then reads the 16-bit
- 558 checksum (CRC) in the Packet Footer and compares it against its own calculated checksum to determine if
- any Data Payload errors have occurred.
- 560 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
- insert additional Filler bytes, if needed, to ensure that each C-PHY Lane transports the same number of 16-
- bit words. The latter rules require the total number of Filler bytes, FC, to be greater than or equal to (WC
- 566 $\mod 2$) + {{N (([WC + 2 + (WC mod 2)] / 2) mod N)} mod N} * 2, where N is the number of Lanes.
- Note that it is possible for FC to be zero.
- Figure 40 illustrates the Lane distribution of the minimal number of Filler bytes required for packets of
- various lengths transmitted over three C-PHY Lanes. The total number of Filler bytes required per packet
- ranges from 0 to 5, depending on the value of the Packet Data Word Count (WC). In general, the minimal
- number of Filler bytes required per packet ranges from 0 to 2N-1 for an N-Lane C-PHY system.
- For the D-PHY physical layer option, the CSI-2 Lane Distributor function shall pass each byte to the
- 573 physical layer which then serially transmits it least significant bit first.
- For the C-PHY physical layer option, the Lane Distributor function shall group each pair of consecutive
- 575 bytes 2n and 2n+1 (for $n \ge 0$) received from the Low Level Protocol into a 16-bit word (whose least
- significant byte is byte 2n) and then pass this word to a physical layer Lane module. The C-PHY Lane
- 577 module maps each 16-bit word into a 7-symbol word which it then serially transmits least significant
- 578 symbol first.
- For both physical layer options, payload data may be presented to the Lane Distributor function in any byte
- order restricted only by data format requirements. Multi-byte protocol elements such as Word Count,
- 581 Checksum and the Short packet 16-bit Data Field shall be presented to the Lane Distributor function least
- significant byte first.
- After the EoT sequence the receiver begins looking for the next SoT sequence.

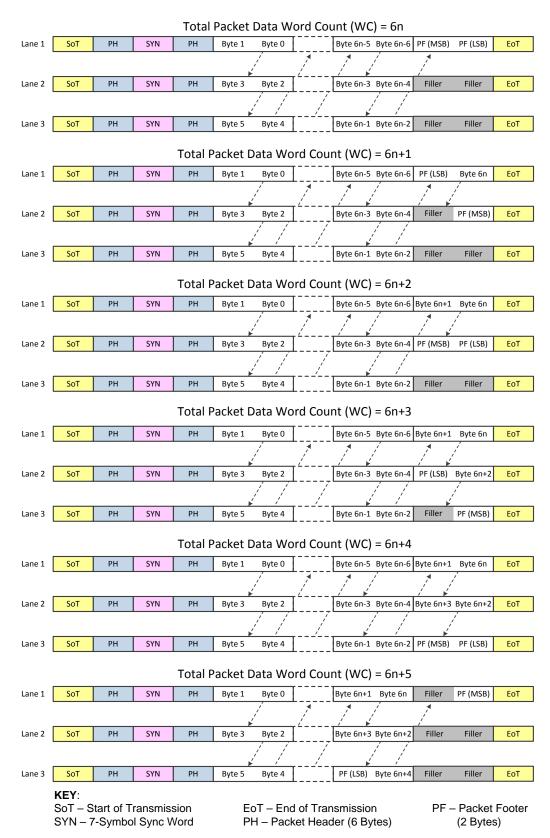


Figure 40 Minimal Filler Byte Insertion Requirements for Three Lane C-PHY

29-May-2014

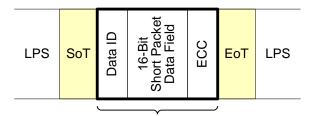
9.1.2 Low Level Protocol Short Packet Format

Figure 41 and Figure 42 show the Low Level Protocol Short Packet structures for the D-PHY and C-PHY physical layer options, respectively. For each option, the Short Packet structure matches the Packet Header of the corresponding Low Level Protocol Long Packet structure with the exception that the Packet Header Word Count (WC) field shall be replaced by the Short Packet Data Field. A Short Packet shall be identified by Data Types 0x00 to 0x0F. See Table 3 for a description of the Data Types. A Short Packet shall contain only a Packet Header; neither Packet Footer nor Packet Filler bytes shall be present.

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

595 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) byte 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.



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

601

602

44

585

586

587

588

589

590

591

592

593

594

596

597

598

599

600

Figure 41 Short Packet Structure for D-PHY Physical Layer Option

CSI-2 "Insert Sync Word" PPI Command:

The physical layer simultaneously inserts a 7-symbol Sync Word on all N Lanes at this point in response to a single CSI-2 PPI command.

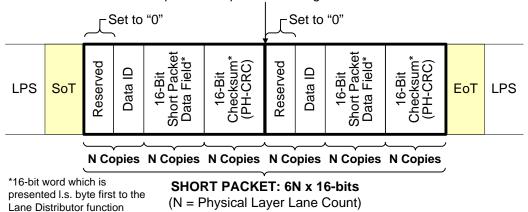


Figure 42 Short Packet Structure for C-PHY Physical Layer Option

Version 1.3 29-May-2014

603

604

605

606

607

608 609

610

611

612

613

614

615

616

9.2 Data Identifier (DI)

The Data Identifier byte contains the Virtual Channel Identifier (VC) value and the Data Type (DT) value as illustrated in Figure 43. The Virtual Channel Identifier is contained in the two MS bits of the Data Identifier Byte. The Data Type value is contained in the six LS bits of the Data Identifier Byte.

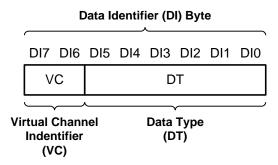


Figure 43 Data Identifier Byte

9.3 Virtual Channel Identifier

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

The Virtual channel identifier number is in the most significant two bits of the Data Identifier Byte. The Receiver will monitor the virtual channel identifier and de-multiplex the interleaved video streams to their appropriate channel. A maximum of four data streams is supported; valid channel identifiers are 0 to 3. The virtual channel identifiers in the peripherals should be programmable to allow the host processor to control how the data streams are de-multiplexed. The principle of logical channels is presented in the Figure 44.

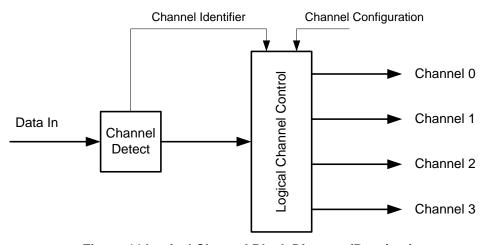


Figure 44 Logical Channel Block Diagram (Receiver)

Figure 45 illustrates an example of data streams utilizing virtual channel support.

Specification for CSI-2 Version 1.3

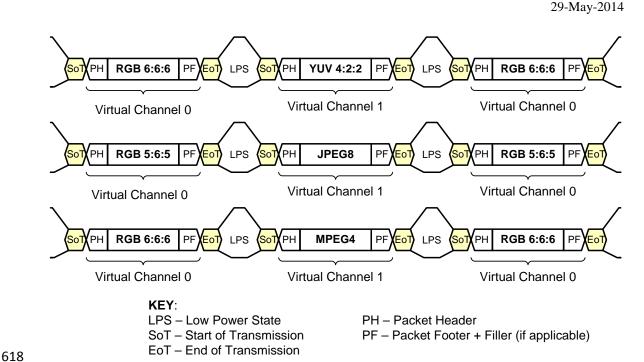


Figure 45 Interleaved Video Data Streams Examples

9.4 Data Type (DT)

619

622

623

624

627

628 629

630

631

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

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

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

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

Table 3 Data Type Classes

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

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

The correct interpretation of the data identifier and word count values is vital to the packet structure. The Packet Header Error Correction Code (ECC) byte allows single-bit errors in the data identifier and the word

Version 1.3 29-May-2014

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

count 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. The 24-bit subset of the code described in Section 9.5.2 shall be used. Therefore, bits 7 and 6 of the ECC byte shall be zero. The error state based on ECC decoding shall be available at the Application layer in the receiver.

The Data Identifier field DI[7:0] shall map to D[7:0] of the ECC input, the Word Count LS Byte (WC[7:0]) to D[15:8] and the Word Count MS Byte (WC[15:8]) to D[23:16]. This mapping is shown in Figure 46, which also serves as an ECC calculation example.

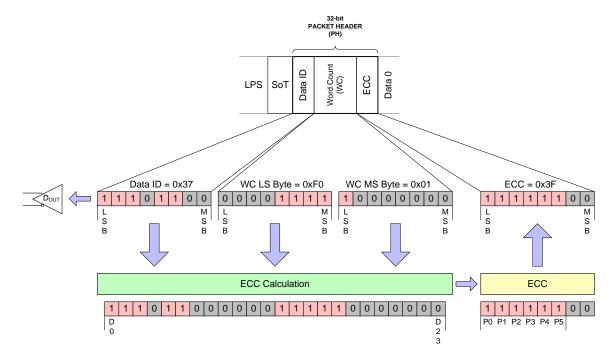


Figure 46 24-bit ECC Generation Example

9.5.1 General Hamming Code Applied to Packet Header

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

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

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

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

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

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

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

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

9.5.2 Hamming-Modified Code

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

666667668

653

654

656

657

658 659

660

661

662

663

664 665

Table 4 ECC Syndrome Association Matrix

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

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

- 671 e.g. 0x07=0b0000_0111=P7P6P5P4P3P2P1P0
- The top row defines the three LSB of data position bit, and the left column defines the three MSB of data position bit (there are 64-bit positions in total).
- e.g. 37th bit position is encoded 0b100_101 and has the syndrome 0x68.
- To derive the parity P0 for 24-bits, the P0's in the orange rows will define if the corresponding bit position is used in P0 parity or not.
- 677 e.g. $P0_{24\text{-bits}} = D0^D1^D2^D4^D5^D7^D10^D11^D13^D16^D20^D21^D22^D23$

29-May-2014

682

683

684

685 686

687

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

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

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

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

Table 5 ECC Parity Generation Rules

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
0	0	0	0	0	0	1	1	1	0x07
1	0	0	0	0	1	0	1	1	0x0B
2	0	0	0	0	1	1	0	1	0x0D
3	0	0	0	0	1	1	1	0	0x0E
4	0	0	0	1	0	0	1	1	0x13
5	0	0	0	1	0	1	0	1	0x15
6	0	0	0	1	0	1	1	0	0x16
7	0	0	0	1	1	0	0	1	0x19
8	0	0	0	1	1	0	1	0	0x1A
9	0	0	0	1	1	1	0	0	0x1C
10	0	0	1	0	0	0	1	1	0x23
11	0	0	1	0	0	1	0	1	0x25
12	0	0	1	0	0	1	1	0	0x26
13	0	0	1	0	1	0	0	1	0x29
14	0	0	1	0	1	0	1	0	0x2A
15	0	0	1	0	1	1	0	0	0x2C
16	0	0	1	1	0	0	0	1	0x31
17	0	0	1	1	0	0	1	0	0x32
18	0	0	1	1	0	1	0	0	0x34
19	0	0	1	1	1	0	0	0	0x38
20	0	0	0	1	1	1	1	1	0x1F
21	0	0	1	0	1	1	1	1	0x2F
22	0	0	1	1	0	1	1	1	0x37
23	0	0	1	1	1	0	1	1	0x3B
24	0	1	0	0	0	0	1	1	0x43
25	0	1	0	0	0	1	0	1	0x45
26	0	1	0	0	0	1	1	0	0x46
27	0	1	0	0	1	0	0	1	0x49

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
28	0	1	0	0	1	0	1	0	0x4A
29	0	1	0	0	1	1	0	0	0x4C
30	0	1	0	1	0	0	0	1	0x51
31	0	1	0	1	0	0	1	0	0x52
32	0	1	0	1	0	1	0	0	0x54
33	0	1	0	1	1	0	0	0	0x58
34	0	1	1	0	0	0	0	1	0x61
35	0	1	1	0	0	0	1	0	0x62
36	0	1	1	0	0	1	0	0	0x64
37	0	1	1	0	1	0	0	0	0x68
38	0	1	1	1	0	0	0	0	0x70
39	1	0	0	0	0	0	1	1	0x83
40	1	0	0	0	0	1	0	1	0x85
41	1	0	0	0	0	1	1	0	0x86
42	1	0	0	0	1	0	0	1	0x89
43	1	0	0	0	1	0	1	0	0x8A
44	0	0	1	1	1	1	0	1	0x3D
45	0	0	1	1	1	1	1	0	0x3E
46	0	1	0	0	1	1	1	1	0x4F
47	0	1	0	1	0	1	1	1	0x57
48	1	0	0	0	1	1	0	0	0x8C
49	1	0	0	1	0	0	0	1	0x91
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

29-May-2014

689

692

9.5.3 ECC Generation on TX Side

This is an informative section.

The ECC can be easily implemented using a parallel approach as depicted in Figure 47 for a 64-bit header.

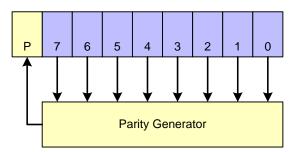
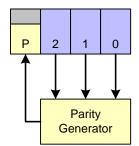


Figure 47 64-bit ECC Generation on TX Side

And Figure 48 for a 24-bit header:



694

697 698

699

700

Figure 48 24-bit ECC Generation on TX Side

The parity generators are based on Table 5.

e.g. P3_{24-bit} = D1^D2^D3^D7^D8^D9^D13^D14^D15^D19^D20^D21^D23

9.5.4 Applying ECC on RX Side

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

Specification for CSI-2 Version 1.3

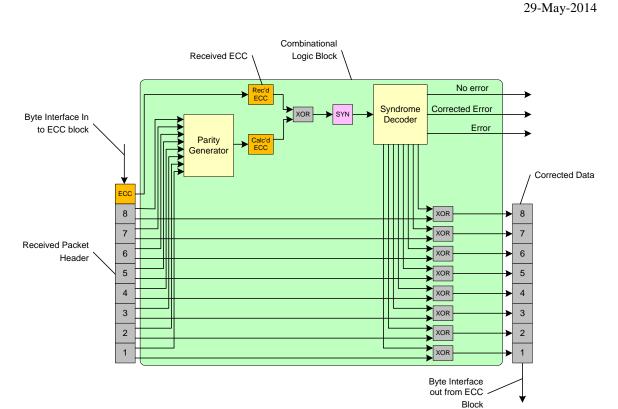


Figure 49 64-bit ECC on RX Side Including Error Correction

Decoding the syndrome has three aspects:

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

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

701

702

703

704

705

706

707

708

709

710

711

714 715

716

717

718

719

720

721

722

723

724

725

726

727

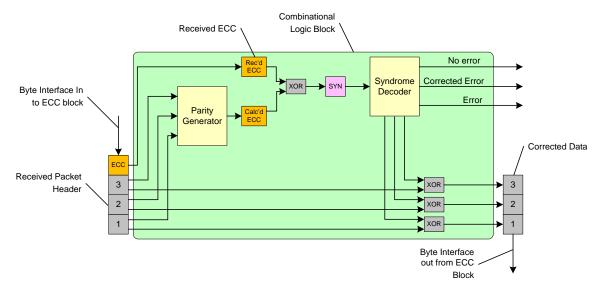


Figure 50 24-bit ECC on RX Side Including Error Correction

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, 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 37, Figure 38, and Figure 42.

The order in which the checksum bytes are presented to the Lane Distributor function is illustrated in Figure 51.

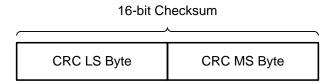


Figure 51 Checksum Transmission Byte Order

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, 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.

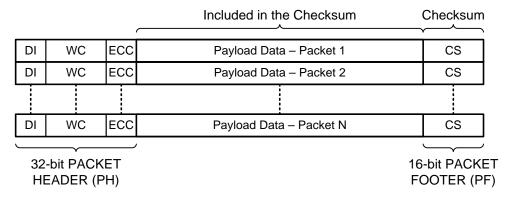
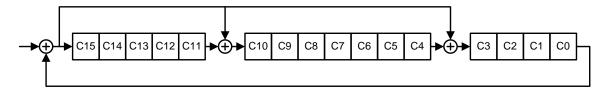


Figure 52 Checksum Generation for Long Packet Payload Data

The definition of a serial CRC implementation is presented in Figure 53. The CRC implementation shall be functionally equivalent with the C code presented in Figure 54. 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 54 equals the final contents of the C[15:0] shift register shown in Figure 53. 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.



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

Figure 53 Definition of 16-bit CRC Shift Register

54

728

729

730

731

732

733

734

735

736

```
#define POLY 0x8408
                      /* 1021H bit reversed */
unsigned short crc16(char *data_p, unsigned short length)
  unsigned char i;
  unsigned int data;
  unsigned int crc = 0xffff;
   if (length == 0)
      return (unsigned short)(crc);
  do
      for (i=0, data=(unsigned int)0xff & *data_p++;
        i < 8;i++, data >>= 1)
         if ((crc & 0x0001) ^ (data & 0x0001))
            crc = (crc >> 1) ^ POLY;
         else
            crc >>= 1;
   } while (--length);
   // Uncomment to change from little to big Endian
// crc = ((crc & 0xff) << 8) | ((crc & 0xff00) >> 8);
   return (unsigned short)(crc);
}
```

Figure 54 16-bit CRC Software Implementation Example

Beginning with index 0, the contents of the input data array in Figure 54 are given by WC 8-bit payload data words for packet data CRC computations and by the four 8-bit Reserved, Data Identifier, WC (LS byte), and WC (MS byte) fields for packet header CRC computations.

744 CRC computation examples:

739

740

741

742

743

754

```
745
      Input Data Bytes:
746
      FF 00 00 02 B9 DC F3 72 BB D4 B8 5A C8 75 C2 7C 81 F8 05 DF FF 00 00 01
747
      Checksum LS byte and MS byte:
748
      F0 00
749
750
      Input Data Bytes:
751
      FF 00 00 00 1E F0 1E C7 4F 82 78 C5 82 E0 8C 70 D2 3C 78 E9 FF 00 00 01
752
      Checksum LS byte and MS byte:
753
      69 E5
```

9.7 Packet Spacing

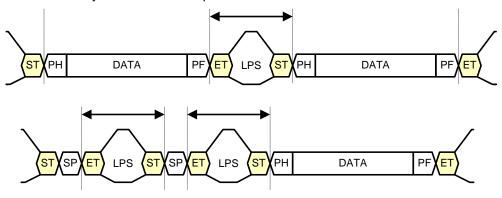
- Between Low Level Protocol packets there must always be a transition into and out of the Low Power State (LPS). Figure 55 illustrates the packet spacing with the LPS.
- The packet spacing does not have to be a multiple of 8-bit data words as the receiver will resynchronize to the correct byte boundary during the SoT sequence prior to the Packet Header of the next packet.

Specification for CSI-2 Version 1.3

29-May-2014

SHORT / LONG PACKET SPACING:

Variable - always a LPS between packets



KEY:

759

760 761

762

765

766

767

768

775

776

777

LPS – Low Power State PH – Packet Header

ST – Start of Transmission PF – Packet Footer + Filler (if applicable)

ET – End of Transmission SP – Short Packet

Figure 55 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.

763 Table 6 Synchronization Short Packet Data Type Codes

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

764 9.8.1 Frame Synchronization Packets

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

For FS and FE synchronization packets the Short Packet Data Field shall contain a 16-bit frame number.

This frame number shall be the same for the FS and FE synchronization packets corresponding to a given frame.

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

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

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

Version 1.3 29-May-2014

779

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

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

9.8.2 Line Synchronization Packets

- 780 Line synchronization packets are optional.
- 781 For Line Start (LS) and Line End (LE) synchronization packets the Short Packet Data Field shall contain a
- 782 16-bit line number. This line number shall be the same for the LS and LE packets corresponding to a given
- 783 line. Line numbers are logical line numbers and are not necessarily equal to the physical line numbers
- The 16-bit line number, when used, shall be non-zero to distinguish it from the case where line number is inoperative and remains set to zero.
- The behavior of the 16-bit line number shall be as one of the following:
 - Line number is always zero line number is inoperative.
 - Line number increments by one for every LS packet within the same Virtual Channel and the same Data Type. The line number is periodically reset to one for the first LS packet after a FS packet. The intended usage is for progressive scan (non- interlaced) video data streams. The line number must be a non-zero value.
 - Line number increments by the same arbitrary step value greater than one for every LS packet within the same Virtual Channel and the same Data Type. The line number is periodically reset to a non-zero arbitrary start value for the first LS packet after a FS packet. The arbitrary start value may be different between successive frames. The intended usage is for interlaced video data streams.

9.9 Generic Short Packet Data Type Codes

Table 7 lists the Generic Short Packet Data Types.

Table 7 Generic Short Packet Data Type Codes

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

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

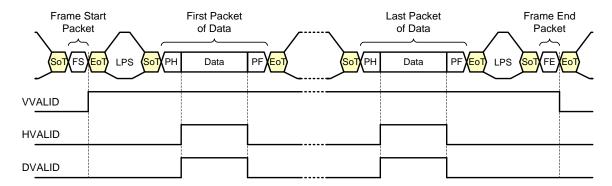
9.10 Packet Spacing Examples

- Packets 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 56 and Figure 57 contain examples of data frames composed of multiple packets and a single packet, respectively.

Specification for CSI-2 Version 1.3

29-May-2014

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:

810

811

812

813

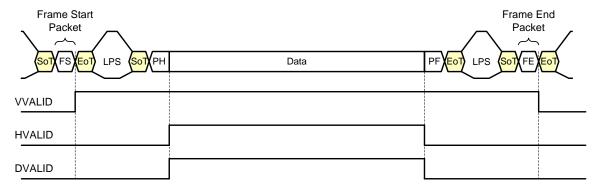
814

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

PH – Packet Header PF – Packet Footer + Filler (if applicable)

FS – Frame Start
LS – Line Start
LE – Line End

Figure 56 Multiple Packet Example



KEY:

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

PH – Packet Header PF – Packet Footer + Filler (if applicable)

 $\begin{array}{lll} \text{FS} - \text{Frame Start} & \text{FE} - \text{Frame End} \\ \text{LS} - \text{Line Start} & \text{LE} - \text{Line End} \\ \end{array}$

Figure 57 Single Packet Example

815

820

821

822

823

825

826

827

828

829

830

831

832

833

834

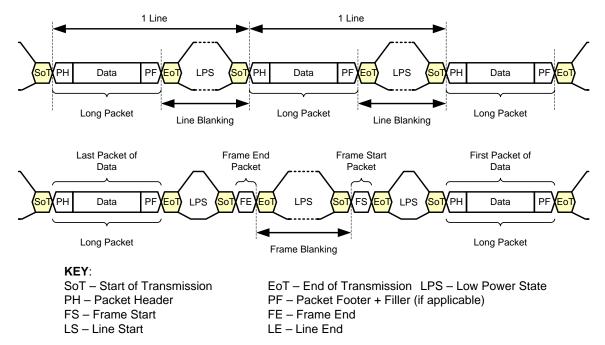


Figure 58 Line and Frame Blanking Definitions

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 58).

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 59.

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

The positions of the Line Start and Line End packets, if present, can be varied within the Line Blanking Period in order to provide pixel accurate horizontal synchronization timing information. See Figure 60.

Specification for CSI-2 Version 1.3

29-May-2014

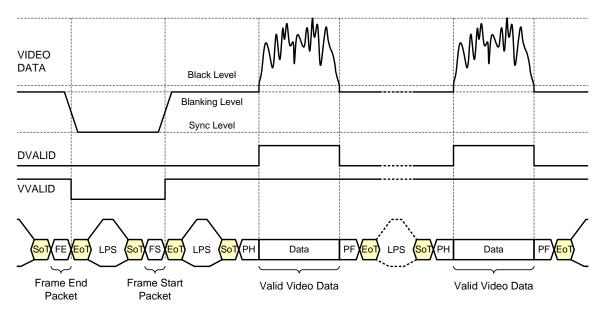


Figure 59 Vertical Sync Example

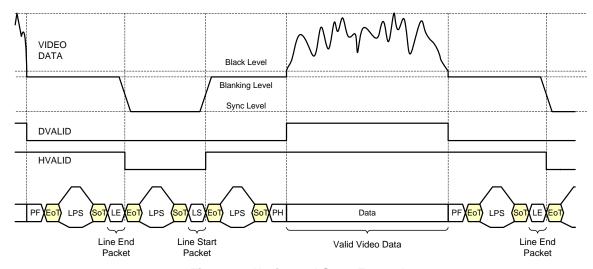


Figure 60 Horizontal Sync Example

9.11 Packet Data Payload Size Rules

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

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

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

839

840

841 842

843

844

845

846

847

848

849

850

851

Version 1.3 29-May-2014

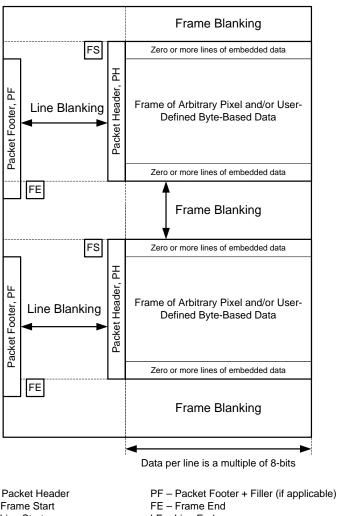
853 required by the RAW10 transmission format as described in Section 11.4.4. The values of such padding 854 pixels are not specified.

Frame Format Examples

This is an informative section.

855 856

- 857 This section contains three examples to illustrate how the CSI-2 features can be used.
- 858 • General Frame Format Example, Figure 61
- 859 • Digital Interlaced Video Example, Figure 62
- 860 • Digital Interlaced Video with accurate synchronization timing information, Figure 63



KEY:

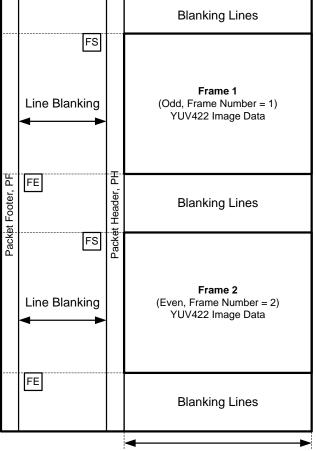
PH - Packet Header

FS - Frame Start LS - Line Start LE - Line End

Figure 61 General Frame Format Example

Specification for CSI-2 Version 1.3

29-May-2014



Data per line is a multiple of 16-bits (YUV422)

KEY:

PH – Packet Header PF – Packet Footer + Filler (if applicable)

FS – Frame Start FE – Frame End

LS – Line Start LE – Line End

Figure 62 Digital Interlaced Video Example

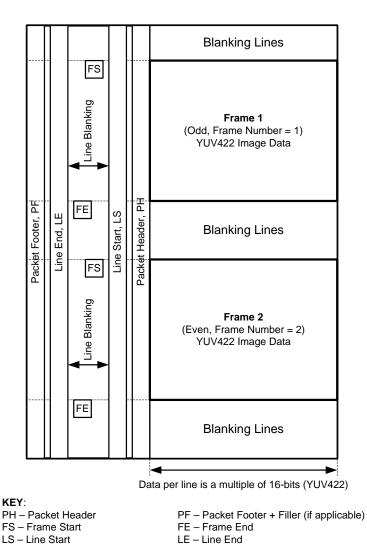


Figure 63 Digital Interlaced Video with Accurate Synchronization Timing Information

9.13 Data Interleaving

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

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

Data Type

863

864 865

866

868

869

871

• Virtual Channel Identifier

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

9.13.1 Data Type Interleaving

The Data Type value uniquely defines the data format for that packet of data. The receiver uses the Data Type value in the packet header to de-multiplex data packets containing different data formats as illustrated in Figure 64. 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.

876

877

878

879

880

881

882

883

884 885

886

887

888

889

890

891

892

893

894

895

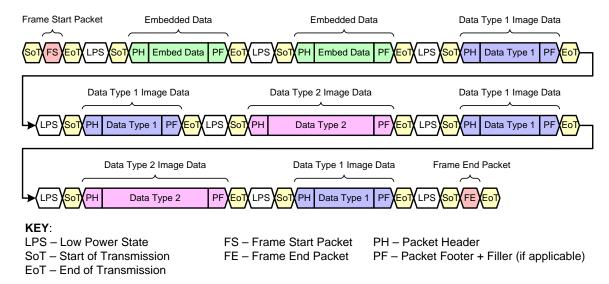
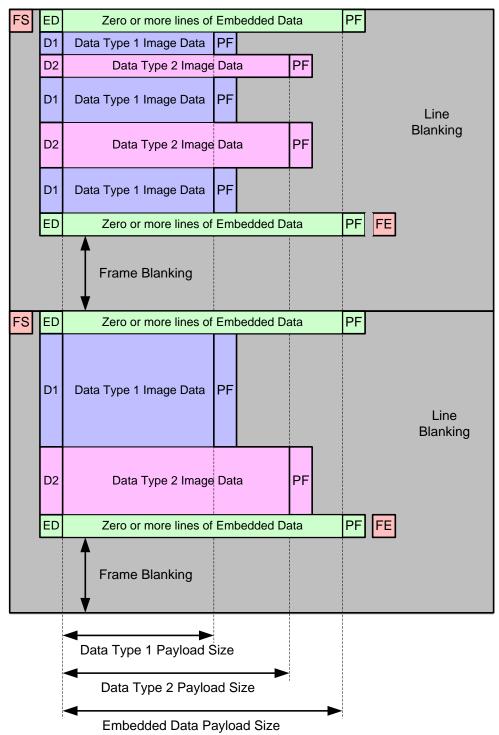


Figure 64 Interleaved Data Transmission using Data Type Value

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

Packets of different data types may be interleaved at either the packet level as illustrated in Figure 65 or the frame level as illustrated in Figure 66. Data formats are defined in Section 11.



KEY:

896

LPS - Low Power State

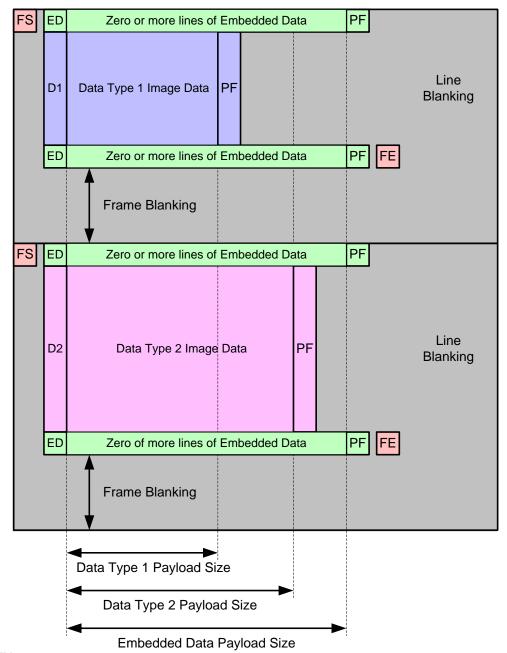
ED – Packet Header containing Embedded Data type code

FS - Frame Start FE - Frame End

D1 - Packet Header containing Data Type 1 Image Data Code D2 - Packet Header containing Data Type 2 Image Data Code

PF – Packet Footer + Filler (if applicable)

Figure 65 Packet Level Interleaved Data Transmission



KEY:

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

PF – Packet Footer + Filler (if applicable)

Figure 66 Frame Level Interleaved Data Transmission

Virtual Channel Identifier Interleaving 9.13.2

The Virtual Channel Identifier allows different data types within a single data stream to be logically separated from each other. Figure 67 illustrates data interleaving using the Virtual Channel Identifier.

897

898

899

Version 1.3 29-May-2014

901

902

903 904

905

906

907

908

909

910

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

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

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

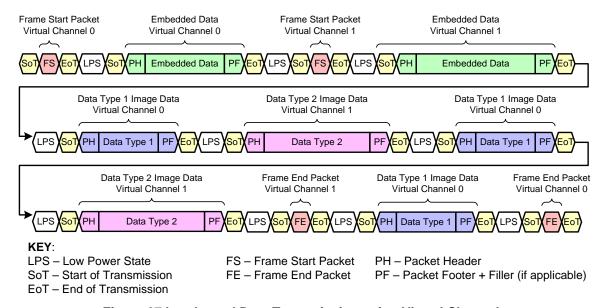


Figure 67 Interleaved Data Transmission using Virtual Channels

911 10 Color Spaces

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

921

915 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

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

924

929

930

931

932

933

934

11 Data Formats

The intent of this section is to provide a definitive reference for data formats typically used in CSI-2 applications. Table 8 summarizes the formats, followed by individual definitions for each format. Generic data types not shown in the table are described in Section 11.1. For simplicity, all examples are single Lane configurations.

The formats most widely used in CSI-2 applications are distinguished by a "primary" designation in Table 8. Transmitter implementations of CSI-2 should support at least one of these primary formats. Receiver implementations of CSI-2 should support all of the primary formats.

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

Table 8 Primary and Secondary Data Formats Definitions

Data Format	Primary	Secondary
YUV420 8-bit (legacy)		S
YUV420 8-bit		S
YUV420 10-bit		S
YUV420 8-bit (CSPS)		S
YUV420 10-bit (CSPS)		S
YUV422 8-bit	Р	
YUV422 10-bit		S
RGB888	Р	
RGB666		S
RGB565	Р	
RGB555		S
RGB444		S
RAW6		S
RAW7		S
RAW8	Р	
RAW10	Р	
RAW12		S
RAW14		S
Generic 8-bit Long Packet Data Types	Р	
User Defined Byte-based Data (Note 1)	Р	

Note:

- 1. Compressed image data should use the user defined, byte-based data type codes
- For clarity the Start of Transmission and End of Transmission sequences in the figures in this section have been omitted.
- The balance of this section details how sequences of pixels and other application data conforming to each of the data types listed in Table 8 are converted into equivalent byte sequences by the CSI-2 Pixel to Byte Packing Formats layer shown in Figure 3.
- Various figures in this section depict these byte sequences as shown at the top of Figure 68, where Byte n always precedes Byte m for n < m. Also note that even though each byte is shown in LSB-first order, this is

not meant to imply that the bytes themselves are bit-reversed by the Pixel to Byte Packing Formats layer prior to output.

For the D-PHY physical layer option, each byte in the sequence is serially transmitted LSB-first, whereas for the C-PHY physical layer option, successive byte pairs in the sequence are encoded and then serially transmitted LSS-first. Figure 68 illustrates these options for a single-Lane system.

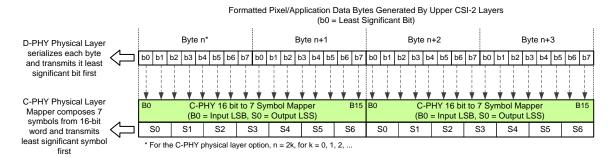


Figure 68 Byte Packing Pixel Data to C-PHY Symbol Illustration

11.1 Generic 8-bit Long Packet Data Types

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

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

951 11.1.1 Null and Blanking Data

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

11.1.2 Embedded Information

- It is possible to embed extra lines containing additional information to the beginning and to the end of each picture frame as presented in the Figure 69. 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.

944

945

946

947

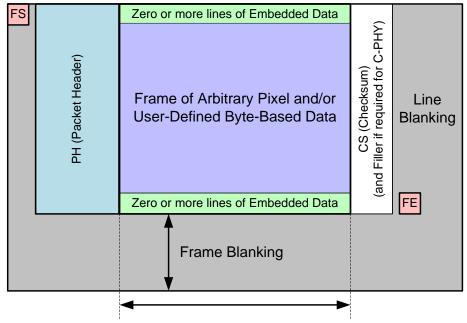
948

949 950

956

957

958



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

KEY:

964

965 966

967

968

969

970

Figure 69 Frame Structure with Embedded Data at the Beginning and End of the Frame

11.2 YUV Image Data

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

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

Table 10 YUV Image Data Types

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

11.2.1 Legacy YUV420 8-bit

972

978

981

P73 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

975 (2, 4, 6 ...). This sequence is illustrated in Figure 70.

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

Table 11 Legacy YUV420 8-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

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

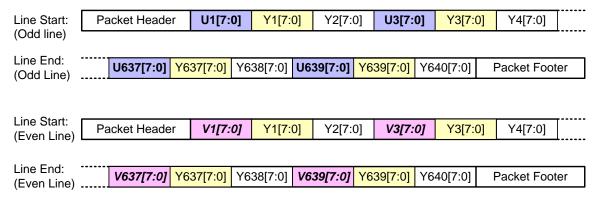


Figure 70 Legacy YUV420 8-bit Transmission

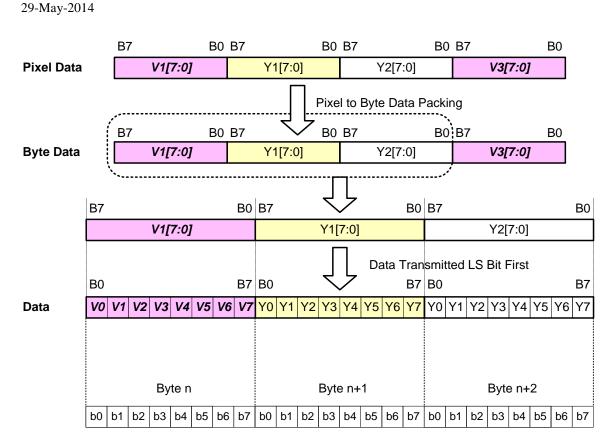


Figure 71 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

There is one spatial sampling option

982

983

984

985

• H.261, H.263 and MPEG1 Spatial Sampling (Figure 72).

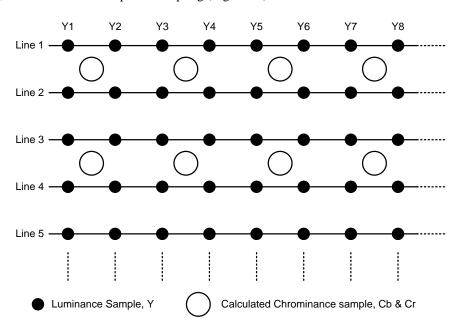


Figure 72 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1

FS		U	Υ	Υ	U	Υ	Υ	 U	Υ	Υ		
		V	Υ	Υ	V	Υ	Υ	 V	Υ	Υ		
		U	Υ	Υ	U	Υ	Υ	 J	Υ	Υ		
	РН	V	Υ	Υ	V	Υ	Υ	 V	Υ	Υ	씸	
	er,	J	Υ	Υ	ט	Υ	Υ	 5	Υ	Υ	er,	
	Header,	V	Υ	Υ	V	Υ	Υ	 V	Υ	Υ	Footer	
	H	U	Υ	Υ	C	Υ	Υ	 5	Υ	Υ	Ψ. Ψ.	
	Packet	V	Υ	Υ	V	Υ	Υ	 V	Υ	Υ	Packet	
	Рас	כ	Υ	Υ	J	Υ	Υ	 >	Υ	Υ	Ра	
		V	Υ	Υ	V	Υ	Υ	 ٧	Υ	Υ		
		J	Υ	Υ	J	Υ	Υ	 5	Υ	Υ		
		V	Υ	Υ	V	Υ	Υ	 V	Υ	Υ		FE

Figure 73 Legacy YUV420 8-bit Frame Format

11.2.2 YUV420 8-bit

986

987

988

989

990

991

992

993

994

997

998

999

1000

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 74.

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

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

Table 12 YUV420 8-bit Packet Data Size Constraints

Odd Lines (1, 3, 5) Luminance Only, Y			ven Lines (2, 4, 6. ce and Chrominar	•	
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 75.

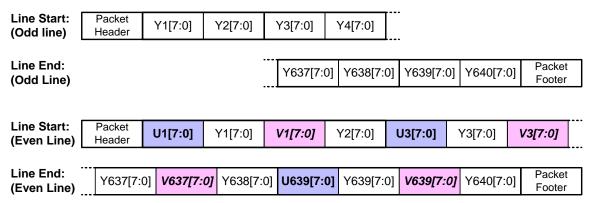
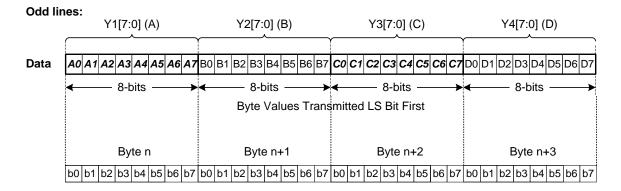


Figure 74 YUV420 8-bit Data Transmission Sequence

29-May-2014



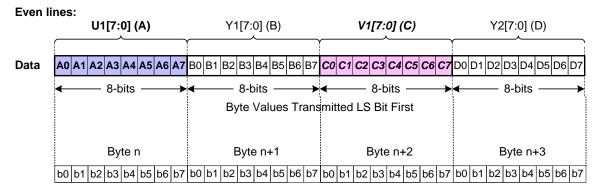


Figure 75 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

There are two spatial sampling options

- H.261, H.263 and MPEG1 Spatial Sampling (Figure 76).
- Chroma Shifted Pixel Sampling (CSPS) for MPEG2, MPEG4 (Figure 77).

Figure 78 shows the YUV420 frame format.

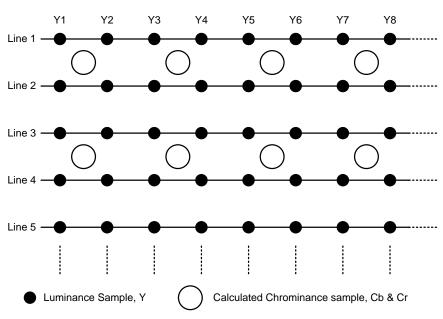
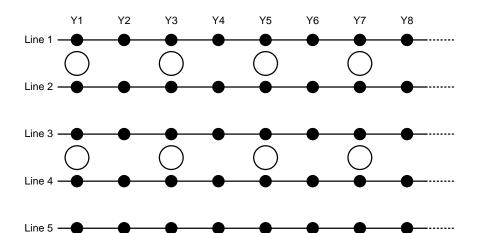


Figure 76 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1

1001

1003



Luminance Sample, Y
 Calculated Chrominance sample, Cb & Cr

Figure 77 YUV420 Spatial Sampling for MPEG 2 and MPEG 4

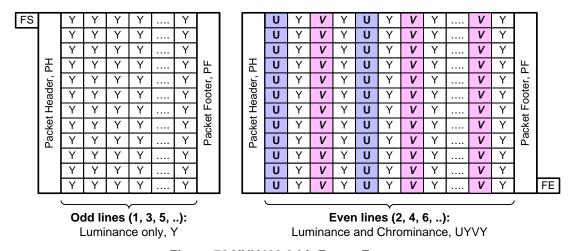


Figure 78 YUV420 8-bit Frame Format

1009 11.2.3 YUV420 10-bit

1007

1008

1010

1011

1012

1013

1014

YUV420 10-bit data transmission is performed by transmitting YYYY... / UYVYUYVY... sequences in odd / even lines. Only the luminance component (Y) is transferred in odd lines (1, 3, 5...) and both luminance (Y) and chrominance (U and V) components transferred in even lines (2, 4, 6...). The format for the even lines (UYVY) is identical to the YUV422 –10-bit data format. The sequence is illustrated in Figure 79.

- The payload data size, in bytes, for even lines (UYVY) is double the payload data size for odd lines (Y).

 This is exception to the general CSI-2 rule that each line shall have an equal length.
- Table 13 specifies the packet size constraints for YUV420 10-bit packets. The length of each packet must be a multiple of the values in the table.

Version 1.3 29-May-2014

1019

1022

1023

Table 13 YUV420 10-bit Packet Data Size Constraints

Odd Lines (1, 3, 5) Luminance Only, Y			ven Lines (2, 4, 6. ce and Chrominar		
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 80.

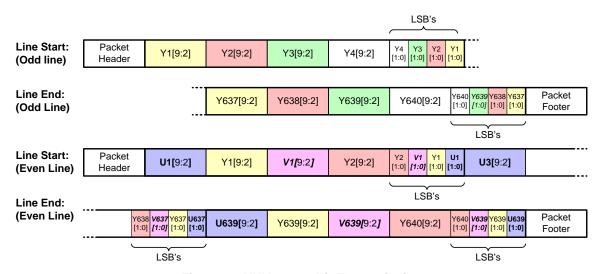
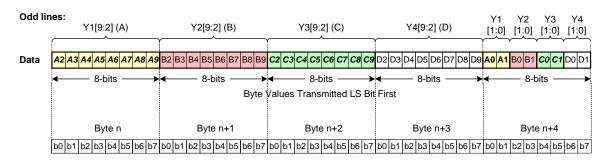


Figure 79 YUV420 10-bit Transmission



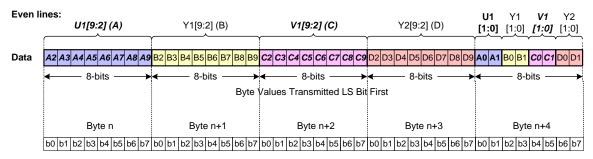


Figure 80 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration

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

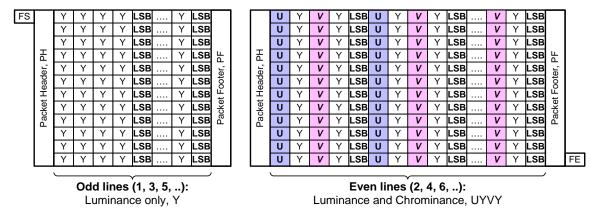


Figure 81 YUV420 10-bit Frame Format

11.2.4 YUV422 8-bit

1025

1026

1027

1028

1029

1030

1031

1034

YUV422 8-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in Figure 82.

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

Table 14 YUV422 8-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	4	32

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in Figure 83.

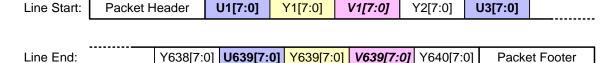


Figure 82 YUV422 8-bit Transmission

29-May-2014

1035

1036

1037

1038

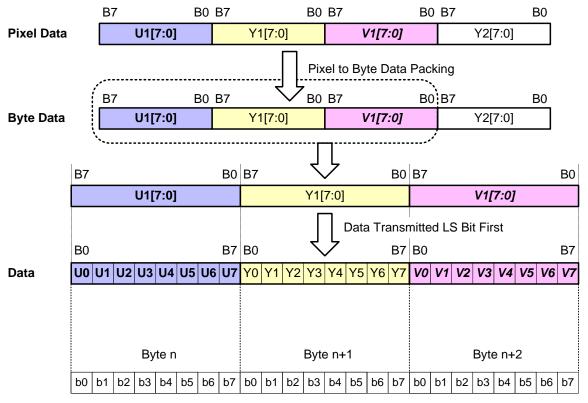


Figure 83 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration

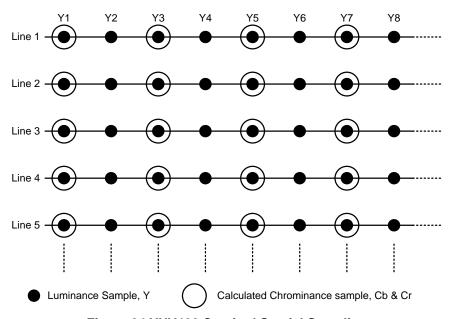


Figure 84 YUV422 Co-sited Spatial Sampling

The pixel spatial alignment is the same as in CCIR-656 standard. The frame format for YUV422 is presented in Figure 85.

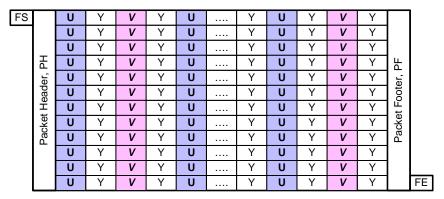


Figure 85 YUV422 8-bit Frame Format

1040 11.2.5 YUV422 10-bit

1039

1045

1048

YUV422 10-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in Figure 86.

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

Table 15 YUV422 10-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	5	40

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in Figure 87.

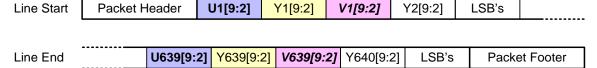


Figure 86 YUV422 10-bit Transmitted Bytes

Pixel Data:

1049

1050

1051

1052

10531054

1055

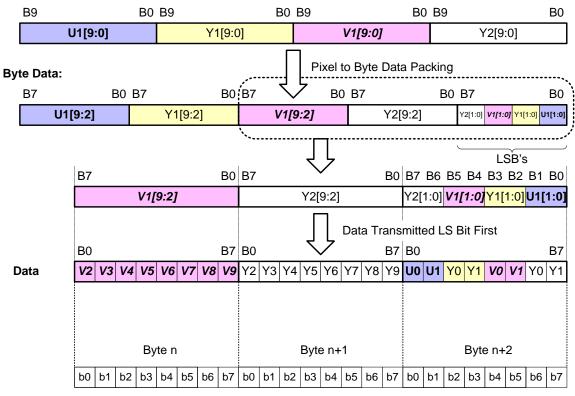


Figure 87 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration

The pixel spatial alignment is the same as in the YUV422 8-bit data case. The frame format for YUV422 is presented in the Figure 88.

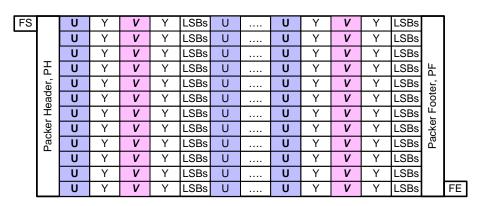


Figure 88 YUV422 10-bit Frame Format

11.3 RGB Image Data

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

Table 16 RGB Image Data Types

Data Type	Description
0x20	RGB444

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

11.3.1 RGB888

1056

1057

1058

1059

1060

1061

1064

RGB888 data transmission is performed by transmitting a BGR byte sequence. This sequence is illustrated in Figure 89. The RGB888 frame format is illustrated in Figure 91.

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

Table 17 RGB888 Packet Data Size Constraints

Pixels	Bytes	Bits
1	3	24

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in Figure 90.

Line Start Packet Header **B1[7:0]** G1[7:0] **R1[7:0] B2[7:0]** G2[7:0] **R2[7:0]**

Line End B639[7:0] G639[7:0] R639[7:0] B640[7:0] G640[7:0] R640[7:0] Packet Footer

Figure 89 RGB888 Transmission

24-bit RGB pixel

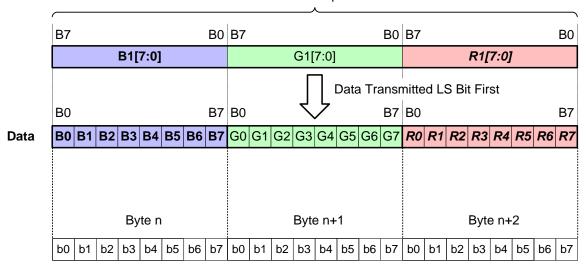


Figure 90 RGB888 Transmission in CSI-2 Bus Bitwise Illustration

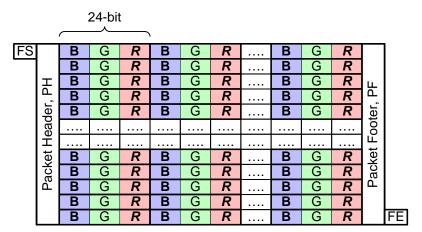


Figure 91 RGB888 Frame Format

RGB666

11.3.2

1066

1067

1070

1071

1073

1074

1075

1076

29-May-2014

RGB666 data transmission is performed by transmitting a B0...5, G0...5, and R0...5 (18-bit) sequence.

This sequence is illustrated in Figure 92. The frame format for RGB666 is presented in the Figure 94.

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

1072 Table 18 RGB666 Packet Data Size Constraints

Pixels	Bytes	Bits
4	9	72

Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB666 case the length of one data word is 18-bits, not eight bits. The word-wise flip is done for 18-bit BGR words; i.e. instead of flipping each byte (8-bits), each 18-bits pixel value is flipped. This is illustrated in Figure 93.

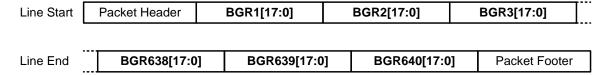


Figure 92 RGB666 Transmission with 18-bit BGR Words

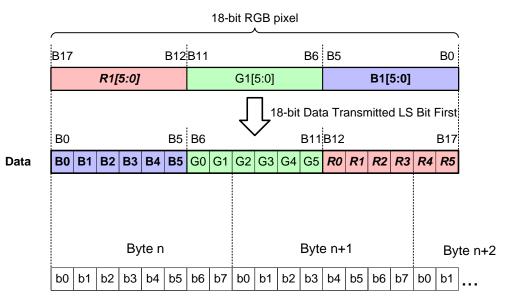


Figure 93 RGB666 Transmission on CSI-2 Bus Bitwise Illustration

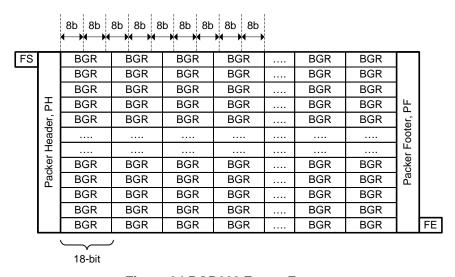


Figure 94 RGB666 Frame Format

11.3.3 **RGB565** 1079

1077

1078

1082

1083

1084

1085

1086

1087

1080 RGB565 data transmission is performed by transmitting B0...B4, G0...G5, R0...R4 in a 16-bit sequence. 1081 This sequence is illustrated in Figure 95. The frame format for RGB565 is presented in the Figure 97.

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

Table 19 RGB565 Packet Data Size Constraints

Pixels	Bytes	Bits
1	2	16

Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB565 case the length of one data word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words; i.e. instead of flipping each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 96.

Version 1.3 Specification for CSI-2

29-May-2014

1088

1089

1090

1092

1093

1094

1095

1096

1097

1098

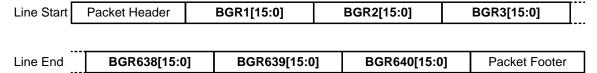


Figure 95 RGB565 Transmission with 16-bit BGR Words

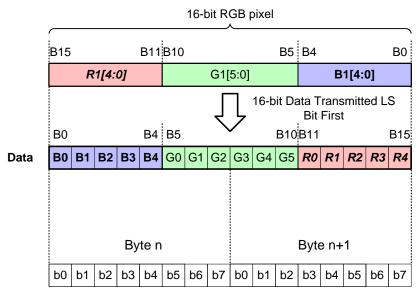


Figure 96 RGB565 Transmission on CSI-2 Bus Bitwise Illustration

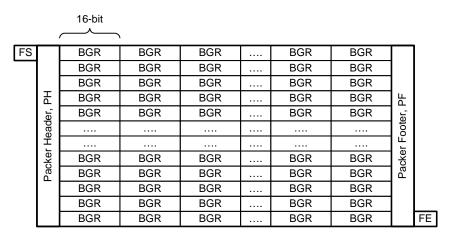


Figure 97 RGB565 Frame Format

1091 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 98.

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 98.

Specification for CSI-2 Version 1.3

29-May-2014

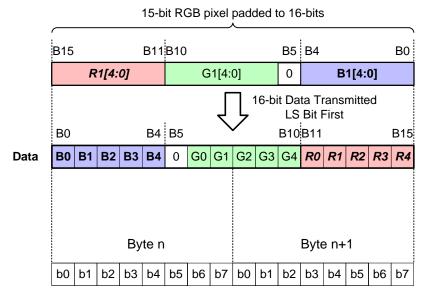


Figure 98 RGB555 Transmission on CSI-2 Bus Bitwise Illustration

11.3.5 **RGB444**

1101 RGB444 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB444 data 1102 should be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs 1103 of each color component as illustrated in Figure 99.

1104 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 99.

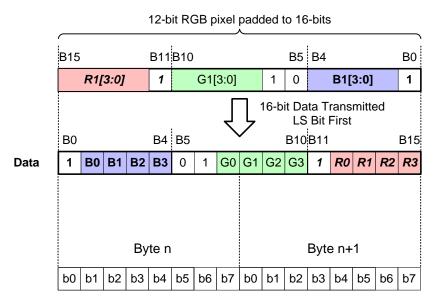


Figure 99 RGB444 Transmission on CSI-2 Bus Bitwise Illustration

1108

1099

1100

1105

1106

1109 11.4 RAW Image Data

- 1110 The RAW 6/7/8/10/12/14 modes are used for transmitting Raw image data from the image sensor.
- 1111 The intent is that Raw image data is unprocessed image data (i.e. Raw Bayer data) or complementary color
- 1112 data, but RAW image data is not limited to these data types.
- 1113 It is possible to transmit e.g. light shielded pixels in addition to effective pixels. This leads to a situation
- 1114 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). 1115
- 1116 Table 20 defines the data type codes for RAW data formats described in this section.

1117

Table 20 RAW Image Data Types

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

1118 11.4.1 RAW6

- 1119 The 6-bit Raw data transmission is done by transmitting the pixel data over CSI-2 bus. Each line is
- 1120 separated by line start / end synchronization codes. This sequence is illustrated in Figure 100 (VGA case).
- 1121 Table 21 specifies the packet size constraints for RAW6 packets. The length of each packet must be a 1122 multiple of the values in the table.

1123

1125

Table 21 RAW6 Packet Data Size Constraints

Pixels	Bytes	Bits
4	3	24

1124 Each 6-bit pixel is sent LSB first. This is an exception to general CSI-2 rule byte wise LSB first.

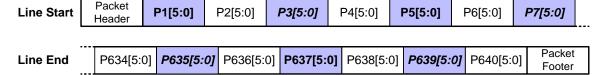


Figure 100 RAW6 Transmission

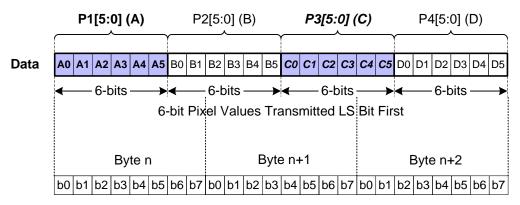


Figure 101 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration

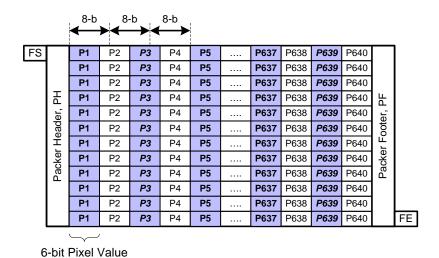


Figure 102 RAW6 Frame Format

1128 11.4.2 RAW7

The 7-bit Raw data transmission is done by transmitting the pixel data over CSI-2 bus. Each line is separated by line start / end synchronization codes. This sequence is illustrated in Figure 103 (VGA case). Table 22 specifies the packet size constraints for RAW7 packets. The length of each packet must be a

multiple of the values in the table.

Table 22 RAW7 Packet Data Size Constraints

Pixels	Bytes	Bits				
8	7	56				

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

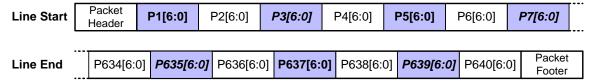


Figure 103 RAW7 Transmission

88

1126

1127

1129

1130

1131

Version 1.3 29-May-2014

1136

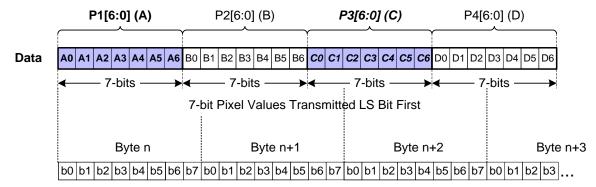
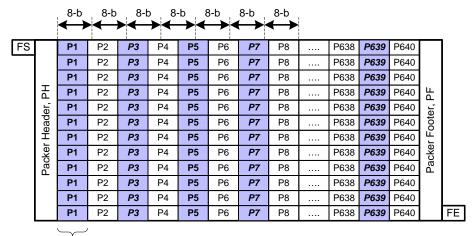


Figure 104 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration



1137 7-bit Pixel Value

Figure 105 RAW7 Frame Format

1138 11.4.3 RAW8

1139

1140

1141

1142

1145

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

Table 23 RAW8 Packet Data Size Constraints

Pixels	Bytes	Bits
1	1	8

- 1143 This sequence is illustrated in Figure 106 (VGA case).
- Bit order in transmission follows the general CSI-2 rule, LSB first.

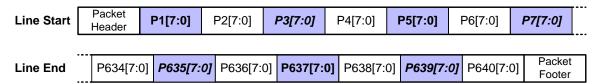


Figure 106 RAW8 Transmission

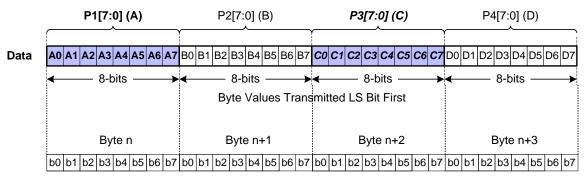


Figure 107 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration

FS		P1	P2	P3	P4	P5	P637	P638	P639	P640		1
го		PI	PZ	P3	P4	Po	 P037	P038	P039	P640		i
		P1	P2	P3	P4	P5	 P637	P638	P639	P640		
		P1	P2	P3	P4	P5	 P637	P638	P639	P640		
	ЬН	P1	P2	P3	P4	P5	 P637	P638	P639	P640	H	
	er,	P1	P2	P3	P4	P5	 P637	P638	P639	P640	e,	
	eader,	P1	P2	P3	P4	P5	 P637	P638	P639	P640	ooter	
	r He	P1	P2	P3	P4	P5	 P637	P638	P639	P640	l F	
	Packei	P1	P2	P3	P4	P5	 P637	P638	P639	P640	ackei	
	Рас	P1	P2	P3	P4	P5	 P637	P638	P639	P640	Ра	
		P1	P2	P3	P4	P5	 P637	P638	P639	P640		
		P1	P2	P3	P4	P5	 P637	P638	P639	P640		
		P1	P2	P3	P4	P5	 P637	P638	P639	P640		FE

Figure 108 RAW8 Frame Format

1148 11.4.4 RAW10

1146

1147

1152

1155

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

Table 24 specifies the packet size constraints for RAW10 packets. The length of each packet must be a multiple of the values in the table.

_

Table 24 RAW10 Packet Data Size Constraints

Pixels	Bytes	Bits
4	5	40

1153 This sequence is illustrated in Figure 109 (VGA case).

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

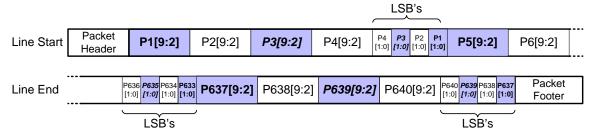


Figure 109 RAW10 Transmission

1156

1157

1159

1160

1161

1162

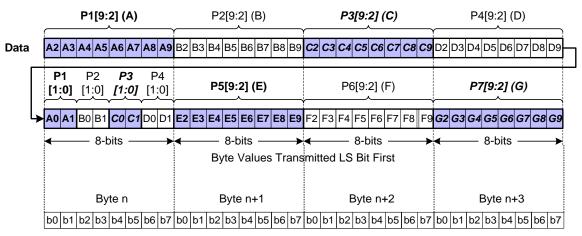


Figure 110 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration

FS		D4	DO	DO	D4	L CD-	D.C.	DCOZ	DCOO	DCCC	DC 40	L CD-		1
гэ		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		i
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		
	PH	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	H	
	er,	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	eľ,	
	Header	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	ooter,	
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	L.	
	acker	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	cke	
	Рас	P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs	Ра	
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		
		P1	P2	P3	P4	LSBs	P5	 P637	P638	P639	P640	LSBs		FE

Figure 111 RAW10 Frame Format

1158 11.4.5 RAW12

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

Table 25 RAW12 Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

- 1163 This sequence is illustrated in Figure 112 (VGA case).
- Bit order in transmission follows the general CSI-2 rule, LSB first.

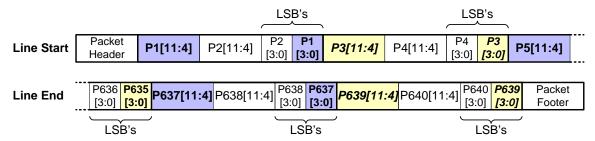


Figure 112 RAW12 Transmission

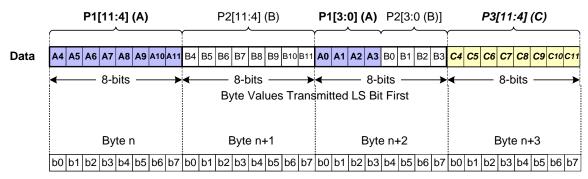


Figure 113 RAW12 Transmission on CSI-2 Bus Bitwise Illustration

_														•
FS		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	ĺ	
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	İ	
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	İ	
	PH	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	出	
	er,	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	er,	
	Header	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	oote	
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	l <u>Ľ</u>	
	ke	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	cke	
	Packer	P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	Pa	
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	ĺ	
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs	İ	
		P1	P2	LSBs	P3	P4	LSBs	 P638	LSBs	P639	P640	LSBs		FE

Figure 114 RAW12 Frame Format

1168 11.4.6 RAW14

1165

1166

1167

1169

1170

1171

1172

1174

1175

1176

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

Table 26 RAW14 Packet Data Size Constraints

Pixels	Bytes	Bits
4	7	56

1173 The sequence is illustrated in Figure 115 (VGA case).

The LS bits for P1, P2, P3 and P4 are distributed in three bytes as shown in Figure 116. The same is true for the LS bits for P637, P638, P639 and P640. The bit order during transmission follows the general CSI-2 rule, i.e. LSB first.

1177

1178

1179

11801181

1182

1183

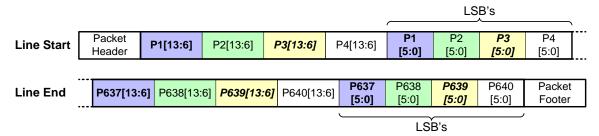


Figure 115 RAW14 Transmission

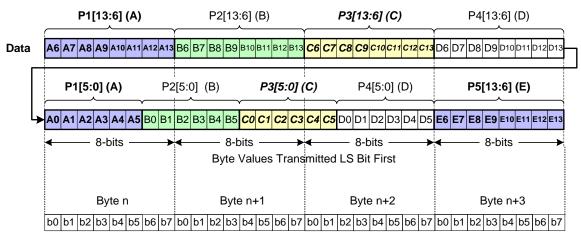


Figure 116 RAW14 Transmission on CSI-2 Bus Bitwise Illustration

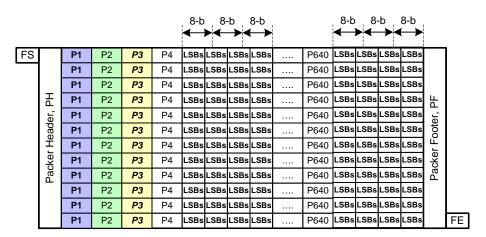


Figure 117 RAW14 Frame Format

11.5 User Defined Data Formats

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

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

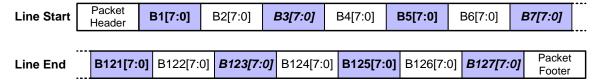


Figure 118 User Defined 8-bit Data (128 Byte Packet)

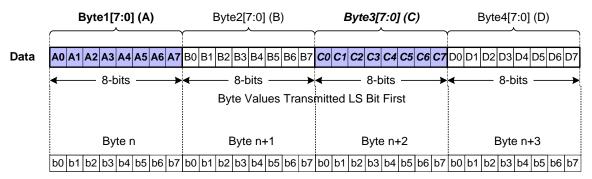


Figure 119 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration

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

1188 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.

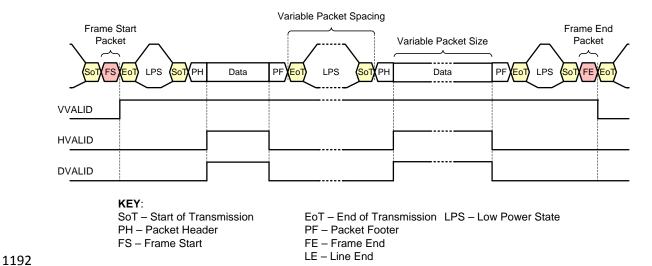


Figure 120 Transmission of User Defined 8-bit Data

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

Table 27 User Defined 8-bit Data Types

Data Type	Description
0x30	User Defined 8-bit Data Type 1
0x31	User Defined 8-bit Data Type 2

1185

1186

1189

1190

1191

1194

Data Type	Description
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

Specification for CSI-2 Version 1.3 29-May-2014

12 Recommended Memory Storage

- 1196 This section is informative.
- The CSI-2 data protocol requires certain behavior from the receiver connected to the CSI transmitter. The
- 1198 following sections describe how different data formats should be stored inside the receiver. While
- 1199 informative, this section is provided to ease application software development by suggesting a common
- data storage format among different receivers.

12.1 General/Arbitrary Data Reception

- 1202 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.

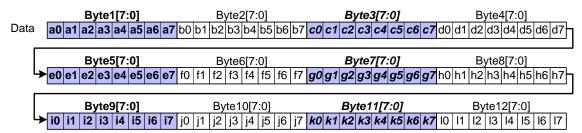
1195

1201

1206

Figure 121 shows the generic CSI-2 byte to 32-bit memory word mapping rule.

Data on CSI-2 bus



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

Figure 121 General/Arbitrary Data Reception

1207 12.2 RGB888 Data Reception

1208 The RGB888 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.

Version 1.3 Specification for CSI-2

1209

1211

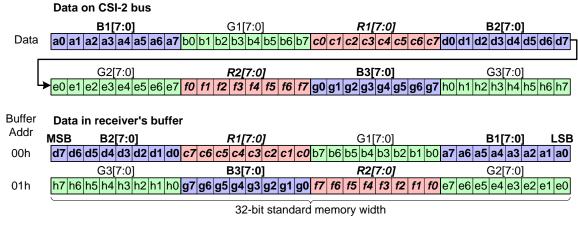


Figure 122 RGB888 Data Format Reception

1210 12.3 RGB666 Data Reception

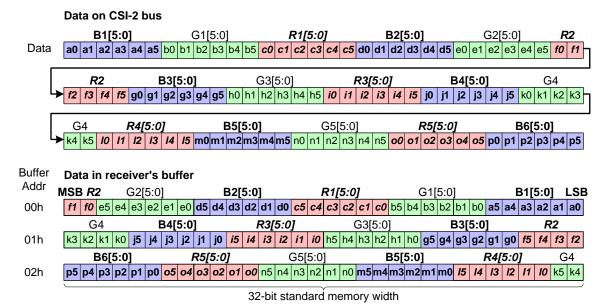


Figure 123 RGB666 Data Format Reception

1212 12.4 RGB565 Data Reception

Data on CSI-2 bus



Buffer	Data in receiv	er's buffer				
Addr	MSB R2[4:0]	G2[5:0]	B2[4:0]	R1[4:0]	G1[5:0]	B1[4:0] LSB
00h	f4 f3 f2 f1 f6	0 e5 e4 e3 e2 e1 e0	d4 d3 d2 d1 d0 c4	4 c3 c2 c1 c0	b5 b4 b3 b2 b1 b0 a	a4 a3 a2 a1 a0
	R4[4:0]	G4[5:0]	B4[4:0]	R3[4:0]	G3[5:0]	B3[4:0]
01h	14 13 12 11 10	0 k5 k4 k3 k2 k1 k0	j4 j3 j2 j1 j0 i4	4 i3 i2 i1 i0	h5 h4 h3 h2 h1 h0	g4 g3 g2 g1 g0
	\ .					,

32-bit standard memory width

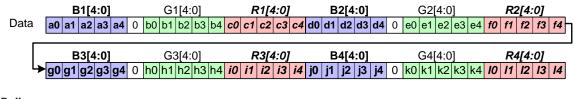
Figure 124 RGB565 Data Format Reception

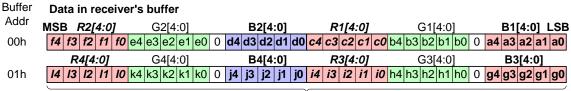
1214 12.5 RGB555 Data Reception

1213

1215

Data on CSI-2 bus





32-bit standard memory width

Figure 125 RGB555 Data Format Reception

1216 12.6 RGB444 Data Reception

The RGB444 data format byte to 32-bit memory word mapping has a special transform as shown in Figure 1218 126.

Version 1.3 Specification for CSI-2

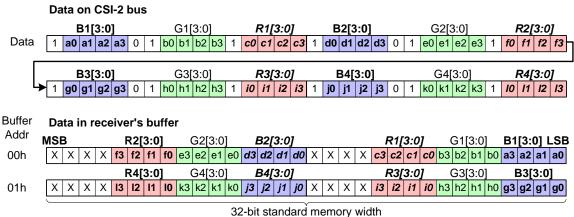


Figure 126 RGB444 Data Format Reception

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

1225 word.

1219

1220

1226

1228

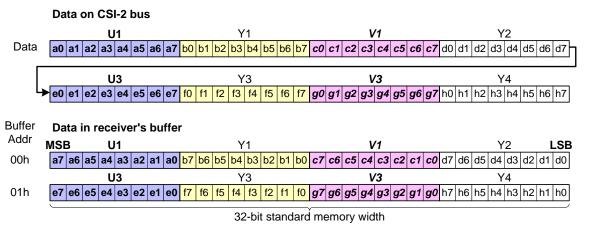


Figure 127 YUV422 8-bit Data Format Reception

1227 12.8 YUV422 10-bit Data Reception

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

Specification for CSI-2 Version 1.3 29-May-2014

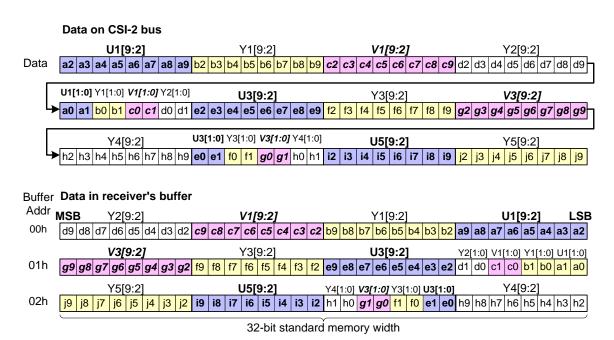


Figure 128 YUV422 10-bit Data Format Reception

12.9 YUV420 8-bit (Legacy) Data Reception

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

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

1229

1230

1231

1232

1233

1234

1237

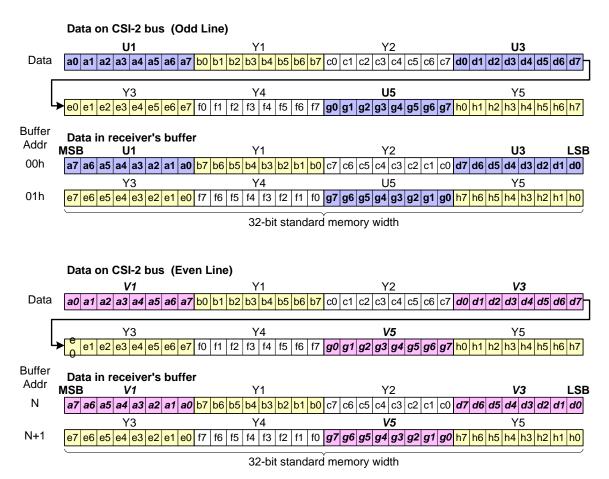


Figure 129 YUV420 8-bit Legacy Data Format Reception

12.10 YUV420 8-bit Data Reception

1238 The YUV420 8-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

Specification for CSI-2 Version 1.3 29-May-2014

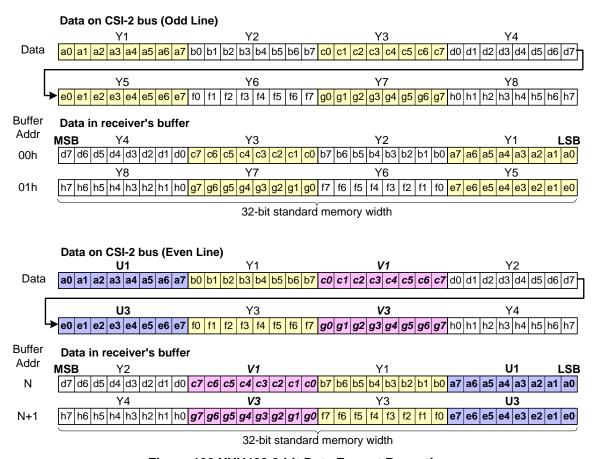


Figure 130 YUV420 8-bit Data Format Reception

1240 12.11 YUV420 10-bit Data Reception

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

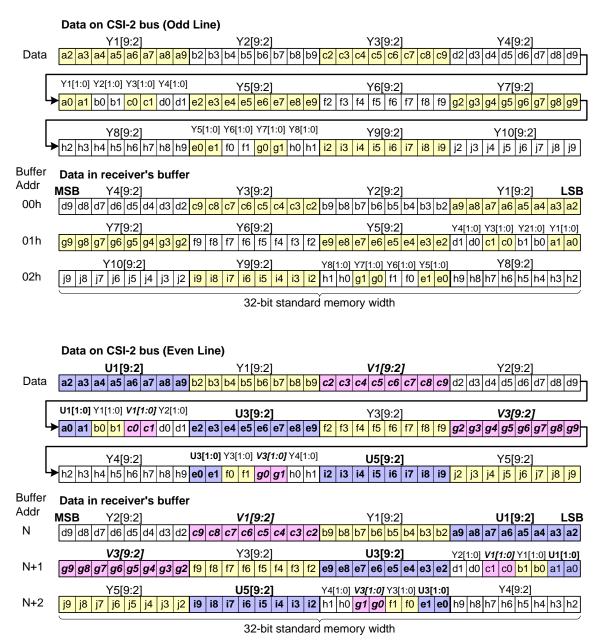


Figure 131 YUV420 10-bit Data Format Reception

Specification for CSI-2 Version 1.3 29-May-2014

12.12 RAW6 Data Reception

1243

1244

1246

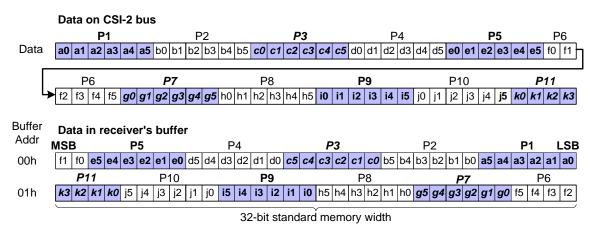


Figure 132 RAW6 Data Format Reception

1245 12.13 RAW7 Data Reception

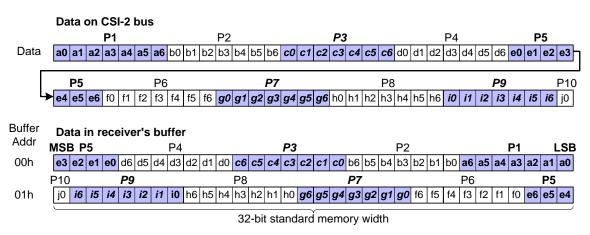


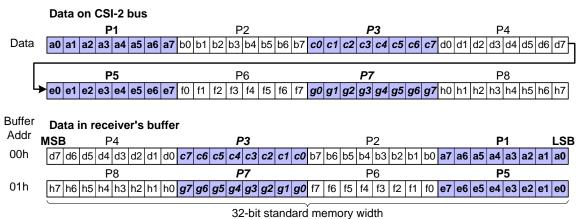
Figure 133 RAW7 Data Format Reception

1247 12.14 RAW8 Data Reception

1248 The RAW8 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

1250

1252



o_ an otaliaala momery main

Figure 134 RAW8 Data Format Reception

12.15 RAW10 Data Reception

1251 The RAW10 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

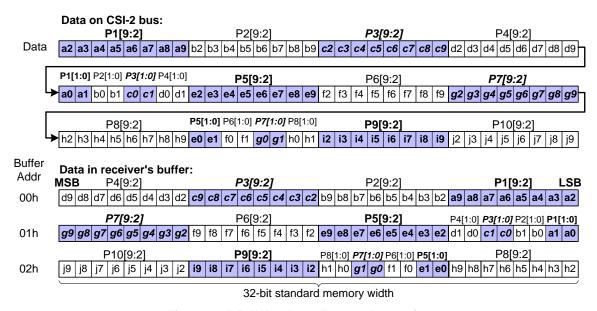


Figure 135 RAW10 Data Format Reception

1253 12.16 RAW12 Data Reception

1254 The RAW12 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

Specification for CSI-2 Version 1.3
29-May-2014

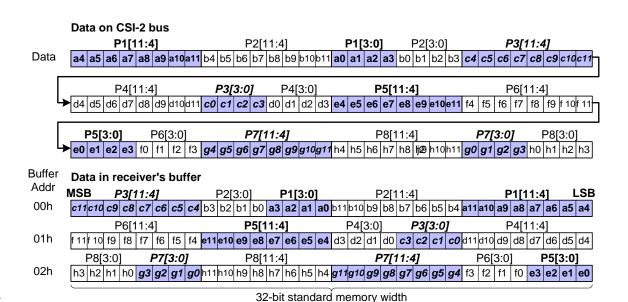


Figure 136 RAW12 Data Format Reception

1256 12.17 RAW14 Data Reception

1255

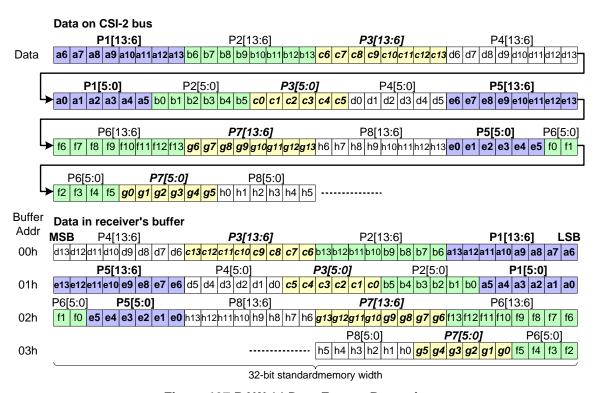


Figure 137 RAW 14 Data Format Reception

Version 1.3 29-May-2014

1266

1267

1268

1269

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.
- 1260 JPEG8 has two non-standard extensions:
- Status information (mandatory)
- Embedded Image information e.g. a thumbnail image (optional)
- Any non-standard or additional data inside the baseline JPEG data structure has to be removed from JPEG8 data before it is compliant with e.g. standard JPEG image viewers in e.g. a personal computer.
- The JPEG8 data flow is illustrated in the Figure 138 and Figure 139.

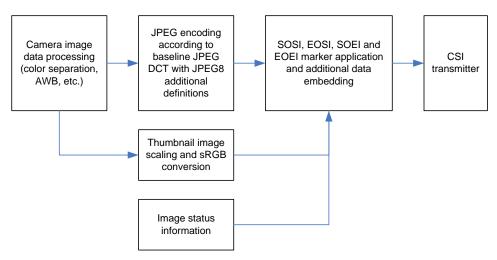


Figure 138 JPEG8 Data Flow in the Encoder

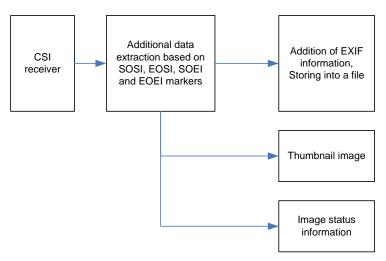


Figure 139 JPEG8 Data Flow in the Decoder

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:

Copyright © 2005-2014 MIPI Alliance, Inc.

Specification for CSI-2 Version 1.3

29-May-2014

• 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 140.
- 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.

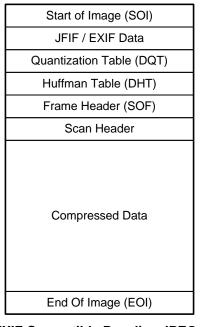


Figure 140 EXIF Compatible Baseline JPEG DCT Format

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 141:
- Image exposure time

1272

1273

1274

1277

1278

- 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
- 1292 Exposure
- Analog and digital gain
- **1294** Gamma

Version 1.3 Specification for CSI-2

29-May-2014

• Color gamut conversion matrix

1296 • Contrast

• Brightness

1298 • Pre-gain

1305

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

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

sequences (0xFF 0x00) are present in the status data.

1304 The SOSI and EOSI markers are defined in Section A.5.

Table 28 Status Data Padding

Data Word	After Padding
D7D6D5D4 D3D2D1D0	1010D7D6D5D4 1010D3D2D1D0

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

1306

1311

Figure 141 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 142.

Specification for CSI-2 Version 1.3

29-May-2014

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.

1321

1322

1323

1324

1325

1326

1330

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.

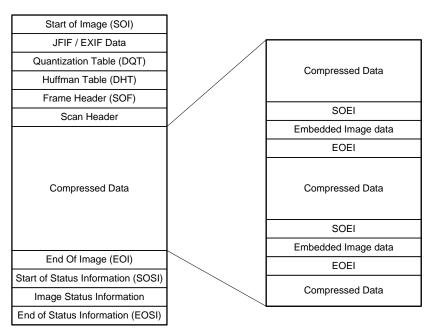


Figure 142 Example of TN Image Embedding Inside the Compressed JPEG Data Block

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.

1338

Table 29 JPEG8 Additional Marker Codes Listing

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

A.6 JPEG8 Data Reception

1337 The compressed data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

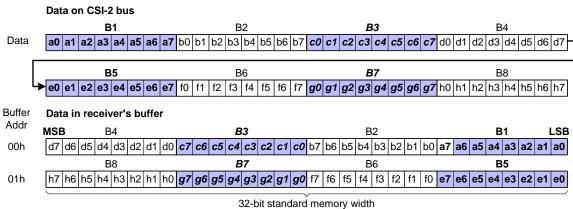


Figure 143 JPEG8 Data Format Reception

29-May-2014

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 144. This implementation example varies from the informative PPI example in [MIPI01].

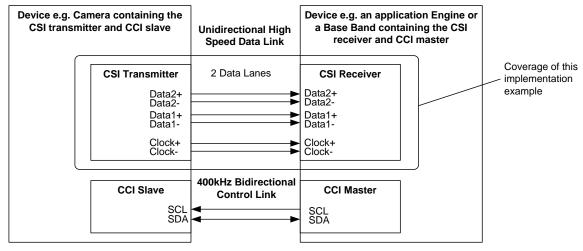


Figure 144 Implementation Example Block Diagram and Coverage

- 1344 For this implementation example a layered structure is described with the following parts:
- D-PHY implementation details
- Multi-lane merger details
- Protocol layer details

1343

- This implementation example refers to a RAW8 data type only; hence no packing/unpacking or byte clock/pixel clock timing will be referenced as for this type of implementation they are not needed.
- No error recovery mechanism or error processing details will be presented, as the intent of the document is to present an implementation from the data flow perspective.

B.2 CSI-2 Transmitter Detailed Block Diagram

Using the layered structure described in the overview the CSI-2 transmitter could have the block diagram in Figure 145.

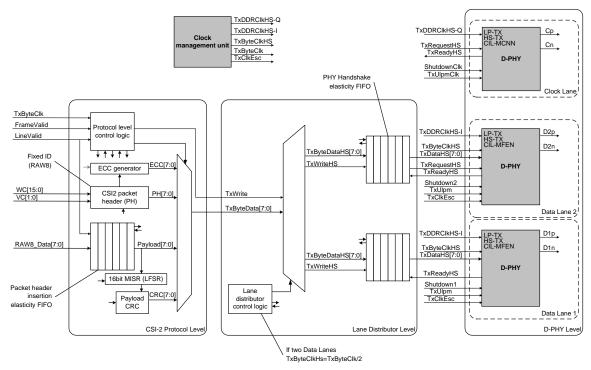


Figure 145 CSI-2 Transmitter Block Diagram

B.3 CSI-2 Receiver Detailed Block Diagram

Using the layered structure described in the overview, the CSI-2 receiver could have the block diagram in Figure 146.

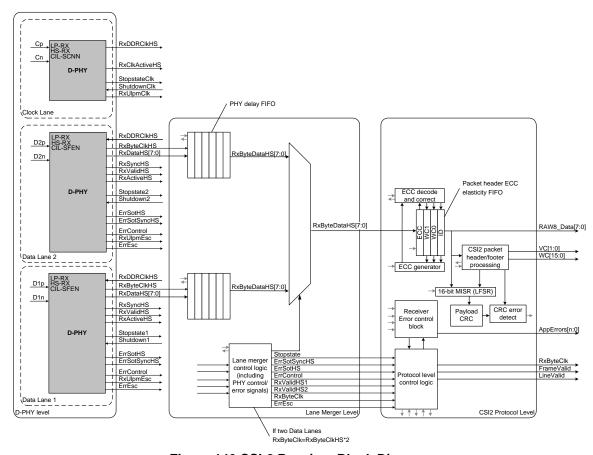


Figure 146 CSI-2 Receiver Block Diagram

B.4 Details on the D-PHY implementation

1358 The PHY level of implementation has the top level structure as seen in Figure 147.

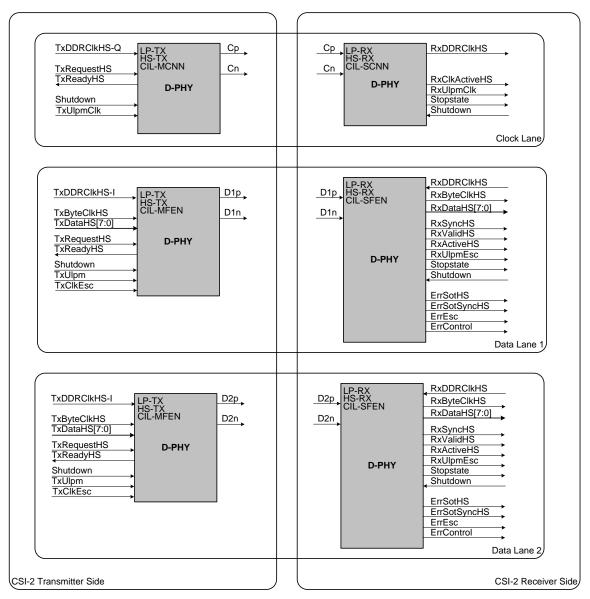


Figure 147 D-PHY Level Block Diagram

- 1360 The components can be categorized as:
- CSI-2 Transmitter side:

1363

1366

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

B.4.1 CSI-2 Clock Lane Transmitter

The suggested implementation can be seen in Figure 148.

Specification for CSI-2 Version 1.3

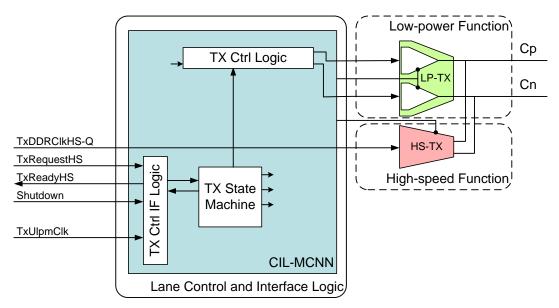


Figure 148 CSI-2 Clock Lane Transmitter

- 1371 The modular D-PHY components used to build a CSI-2 clock lane transmitter are:
- 1372 • **LP-TX** for the Low-power function

1370

1374

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

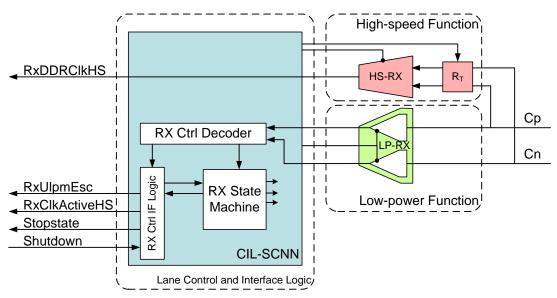
1387

1388

- 1373 • **HS-TX** for the High-speed function
 - CIL-MCNN for the Lane control and interface logic
- 1375 The PPI interface signals to the CSI-2 clock lane transmitter are:
 - TxDDRClkHS-Q (Input): High-Speed Transmit DDR Clock (Quadrature).
 - TxRequestHS (Input): High-Speed Transmit Request. This active high signal causes the lane module to begin transmitting a high-speed clock.
 - TxReadyHS (Output): High-Speed Transmit Ready. This active high signal indicates that the clock lane is transmitting HS clock.
 - Shutdown (Input): Shutdown Lane Module. This active high signal forces the lane module into "shutdown", disabling all activity. All line drivers, including terminators, are turned off when Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on any clock.
 - TxUlpmClk (Input): Transmit Ultra Low-Power mode on Clock Lane This active high signal is asserted to cause a Clock Lane module to enter the Ultra Low-Power mode. The lane module remains in this mode until TxUlpmClk is de-asserted.

B.4.2 CSI-2 Clock Lane Receiver

1389 The suggested implementation can be seen in Figure 149.



1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

Figure 149 CSI-2 Clock Lane Receiver

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

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

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

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

B.4.3 CSI-2 Data Lane Transmitter

1410 The suggested implementation can be seen in Figure 150.

Specification for CSI-2 Version 1.3

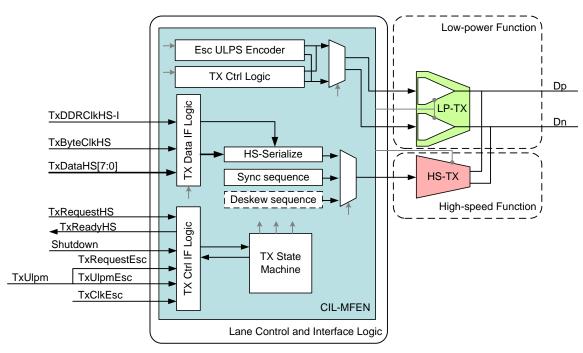


Figure 150 CSI-2 Data Lane Transmitter

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:

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

Version 1.3 29-May-2014

1439

1440

1441

1442

1443

1444

1445

1446 1447

1448

1449

1450

1451

1452

1453

1455

1458

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

B.4.4 CSI-2 Data Lane Receiver

1454 The suggested implementation can be seen in Figure 151.

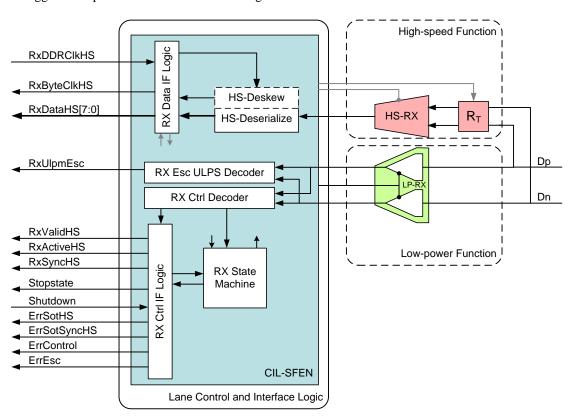


Figure 151 CSI-2 Data Lane Receiver

- The modular D-PHY components used to build a CSI-2 data lane receiver are:
- LP-RX for the Low-power function
 - **HS-RX** for the High-speed function

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

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

C.1 Overview

- This section proposes one approach to handling error conditions at the receiving side of a CSI-2 Link.
 Although the section is informative and therefore does not affect compliance for CSI-2, the approach is
 offered by the MIPI Camera Working Group as a recommended approach. The CSI-2 receiver assumes the
 case of a CSI-2 Link comprised of unidirectional Lanes for D-PHY Clock and Data Lanes with Escape
 Mode functionality on the Data Lanes and a continuously running clock. This Annex does not discuss other
 cases, including those that differ widely in implementation, where the implementer should consider other
 potential error situations.
- Because of the layered structure of a compliant CSI-2 receiver implementation, the error behavior is described in a similar way with several "levels" where errors could occur, each requiring some implementation at the appropriate functional layer of the design:
- **1515** *D-PHY Level errors*

1516

1518

1519

1520 1521

1522

1524

1526

1527

1528

15311532

1533

1534

1535

1536

- 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.
- 1523 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 consists of the following:

1547

1548

1549

1550

1551

15521553

1554

1555

1556

1558

1559

1560

1561

1562

1563

1564

1565

1566 1567

1568

1576

1577

1587

1588

1589

1590

1591

- 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.
- 1557 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 byte and computing the CRC of the data payload.
- 1573 Decoding and applying the ECC byte of the Packet Header should signal the following errors:
- **ErrEccDouble:** Asserted when an ECC syndrome was computed and two bit-errors are detected in the received Packet Header.
 - ErrEccCorrected: Asserted when an ECC syndrome was computed and a single bit-error in the Packet Header was detected and corrected.
- ErrEccNoError: Asserted when an ECC syndrome was computed and the result is zero indicating a Packet Header that is considered to be without errors or has more than two bit-errors.

 CSI-2's ECC mechanism cannot detect this type of error.
- Also, computing the CRC code over the whole payload of the received packet could generate the following errors:
- ErrCrc: Asserted when the computed CRC code is different than the received CRC code.
- ErrID: Asserted when a Packet Header is decoded with an unrecognized or unimplemented data ID.
- 1586 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.

Version 1.3 29-May-2014

1595

1596

1599

1600

1601

1602

1603

1604

1605

1606

1607

1608

1609

1610

1612

1613

1614

1615

1616

1617

1618

1619

1620

1621 1622

1623

1624

1625

1626

- 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.
- 1611 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.

Specification for CSI-2 Version 1.3

29-May-2014

D.1 Overview

- 1627 Since a camera in a mobile terminal spends most of its time in an inactive state, implementers need a way
- 1628 to put the CSI-2 Link into a low power mode that approaches, or may be as low as, the leakage level. This
- 1629 section proposes one approach for putting a CSI-2 Link in a "Sleep Mode" (SLM). Although the section is
- 1630 informative and therefore does not affect compliance for CSI-2, the approach is offered by the MIPI
- 1631 Camera Working Group as a recommended approach.

Annex D CSI-2 Sleep Mode (informative)

- 1632 This approach relies on an aspect of a D-PHY or C-PHY transmitter's behavior that permits regulators to be
- 1633 disabled safely when LP-00 (Space state) is on the Link. Accordingly, this will be the output state for a
- 1634 CSI-2 camera transmitter in SLM.
- 1635 SLM can be thought of as a three-phase process:
- 1636 1. SLM Command Phase. The 'ENTER SLM' command is issued to the TX side only, or to both 1637 sides of the Link.
- 1638 2. SLM Entry Phase. The CSI-2 Link has entered, or is entering, the SLM in a controlled or 1639 synchronized manner. This phase is also part of the power-down process.
- 1640 3. SLM Exit Phase. The CSI-2 Link has exited the SLM and the interface/device is operational. This 1641 phase is also part of the power-up process.
- 1642 In general, when in SLM, both sides of the interface will be in ULPS, as defined in [MIPI01] or [MIPI02].

SLM Command Phase D.2

- 1643 For the first phase, initiation of SLM occurs by a mechanism outside the scope of CSI-2. Of the many 1644 mechanisms available, two examples would be:
- 1645 1. An External SLEEP signal input to the CSI-2 transmitter and optionally also to the CSI-2
- 1646 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. 1647
- 1648 2. A CCI control command, provided on the I2C control Link, is used to trigger ULPS.

D.3 SLM Entry Phase

- 1649 For the second phase, consider one option:
- 1650 Only the TX side enters SLM and propagates the ULPS to the RX side by sending a D-PHY or C-PHY
- 'ULPS' command on each Lane. In Figure 152, only the Data Lane 'ULPS' command is used as an 1651
- 1652 example. The D-PHY Dp, Dn, and C-PHY Data_A, Data_C are logical signal names and do not imply
- specific multiplexing on dual mode (combined D-PHY and C-PHY) implementations. 1653

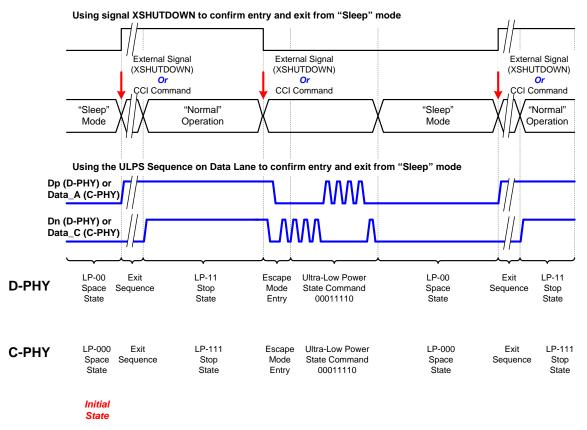


Figure 152 SLM Synchronization

D.4 SLM Exit Phase

1654

1655

1656

1657

1658 1659

1660

1661

1662

1663

1664

1665

1666

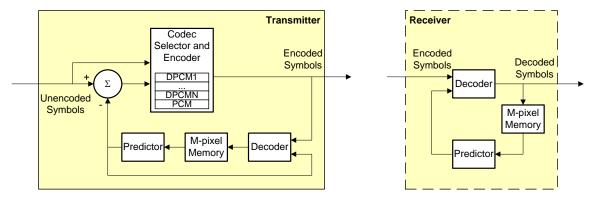
For the third phase, three options are presented and assume the camera peripheral is in ULPS or Sleep mode at power-up:

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

A handshake using the 'ULPS' mechanism as described in [MIPI01] or [MIPI02], as appropriate, should be used for powering up the interface.

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
- $1672 \qquad \text{the original image, Y is the encoded (compressed) bits per pixel and Z is the decoded (uncompressed) bits} \\$
- per pixel.
- 1674 The following data compression schemes are defined:
- **1675** 12–8–12
- **1676** 12–7–12
- **1677** 12–6–12
- **1678** 10–8–10
- **1679** 10–7–10
- **1680** 10–6–10
- To identify the type of data on the CSI-2 interface, packets with compressed data shall have a User Defined
- Data Type value as indicated in Table 27. Note that User Defined data type codes are not reserved for
- 1683 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.
- 1687 The number of bits in a packet shall be a multiple of eight. Therefore, implementations with data
- 1688 compression schemes that result in each pixel having less than eight encoded bits per pixel shall transfer the
- encoded data in a packed pixel format. For example, the 12–7–12 data compression scheme uses a packed
- pixel format as described in Section 11.4.2 except the Data Type value in the Packet Header is a User
- 1691 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
- 1700 The data compression system consists of encoder, decoder and predictor blocks as shown in Figure 153.



1702

1703

1704

1705

1706

1707

1708

1709

1710

1711

Figure 153 Data Compression System Block Diagram

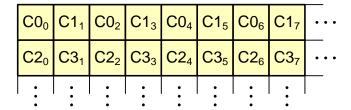
The encoder uses a simple algorithm to encode the pixel values. A fixed number of pixel values at the beginning of each line are encoded without using prediction. These first few values are used to initialize the predictor block. The remaining pixel values on the line are encoded using prediction.

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

E.1 Predictors

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

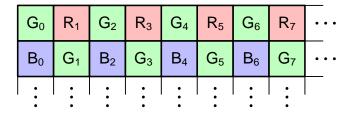
1712 The order of pixels in a raw image is shown in Figure 154.



1713

Figure 154 Pixel Order of the Original Image

1714 Figure 155 shows an example of the pixel order with RGB data.



1715

1717

1718

Figure 155 Example Pixel Order of the Original Image

1716 Two predictors are defined for use in the data compression schemes.

Predictor1 uses a very simple algorithm and is intended to minimize processing power and memory size requirements. Typically, this predictor is used when the compression requirements are modest and the

- original image quality is high. Predictor1 should be used with 10-8-10, 10-7-10 and 12-8-12 data
- 1720 compression schemes.
- 1721 The second predictor, Predictor2, is more complex than Predictor1. This predictor provides slightly better
- 1722 prediction than Predictor1 and therefore the decoded image quality can be improved compared to
- 1723 Predictor1. Predictor2 should be used with 10–6–10, 12–7–12, and 12–6–12 data compression schemes.
- 1724 Both receiver and transmitter shall support Predictor1 for all data compression schemes.

E.1.1 Predictor1

- 1725 Predictor1 uses only the previous same color component value as the prediction value. Therefore, only a
- two-pixel deep memory is required.
- The first two pixels $(C0_0, C1_1 / C2_0, C3_1)$ or as in example $G_0, R_1 / B_0, G_1$ in a line are encoded without
- 1728 prediction.

1743

- 1729 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:
- 1731 Xpred(n) = Xdeco(n-2)

E.1.2 Predictor2

- 1732 Predictor2 uses the four previous pixel values, when the prediction value is evaluated. This means that also
- 1733 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.
- 1735 Predictor2 uses all color components of the four previous pixel values to create the prediction value.
- 1736 Therefore, a four-pixel deep memory is required.
- 1737 The first pixel ($C0_0 / C2_0$, or as in example G_0 / B_0) in a line is coded without prediction.
- 1738 The second pixel (C1₁ / C3₁ or as in example R₁ / G₁) 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:

```
1740 xpred(n) = xdeco(n-1)
```

The third pixel $(C0_2 / C2_2 \text{ or as in example } G_2 / B_2)$ in a line is predicted using the previous decoded same color value as a prediction value. The third pixel is predicted with the following equation:

```
Xpred(n) = Xdeco(n-2)
```

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

Other pixels in all lines are predicted using the equation:

```
1752
          if ((Xdeco(n-1) \leftarrow Xdeco(n-2)) AND Xdeco(n-2) \leftarrow Xdeco(n-3)) OR
1753
             (Xdeco(n-1) >= Xdeco(n-2) AND Xdeco(n-2) >= Xdeco(n-3))) then
1754
                Xpred(n) = Xdeco(n-1)
1755
          else if ((Xdeco(n-1) <= Xdeco(n-3) AND Xdeco(n-2) <= Xdeco(n-4)) OR
1756
             (Xdeco(n-1) >= Xdeco(n-3) AND Xdeco(n-2) >= Xdeco(n-4))) then
1757
                Xpred(n) = Xdeco(n-2)
1758
          else
1759
             Xpred(n) = (Xdeco(n-2) + Xdeco(n-4) + 1) / 2
1760
          endif
```

E.2 Encoders

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

Version 1.3 Specification for CSI-2

29-May-2014

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

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

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

Pixels without prediction are encoded using the following formula:

```
1766 Xenco(n) = Xorig(n) / 4
```

1767 To avoid a full-zero encoded value, the following check is performed:

```
1768 if (Xenco(n) == 0) then
1769 Xenco(n) = 1
1770 endif
```

1771 Pixels with prediction are encoded using the following formula:

```
1772
          if (abs(Xdiff(n)) < 32) then
1773
              use DPCM1
1774
          else if (abs(Xdiff(n)) < 64) then
1775
              use DPCM2
1776
          else if (abs(Xdiff(n)) < 128) then
1777
              use DPCM3
1778
          else
1779
              use PCM
1780
          endif
```

E.2.1.1 DPCM1 for 10-8-10 Coder

```
1781 Xenco(n) has the following format:
```

1782

Xenco(n) = "00 s xxxxx"

1787 The coder equation is described as follows:

```
1788 if (Xdiff( n ) <= 0) then

1789 sign = 1

1790 else

1791 sign = 0

1792 endif

1793 value = abs(Xdiff( n ))
```

1794 Note: Zero code has been avoided (0 is sent as -0).

E.2.1.2 DPCM2 for 10-8-10 Coder

```
1795 Xenco(\mathbf{n}) has the following format:
```

```
1796
           Xenco(n) = "010 s xxxx"
1797
        where.
1798
           "010" is the code word
1799
           "s" is the sign bit
1800
           "xxxx" is the four bit value field
1801
        The coder equation is described as follows:
1802
           if (Xdiff(n) < 0) then
1803
               sign = 1
1804
           else
1805
               sign = 0
```

Specification for CSI-2 Version 1.3
29-May-2014

```
1806
           endif
1807
           value = (abs(Xdiff(n)) - 32) / 2
        E.2.1.3
                   DPCM3 for 10-8-10 Coder
1808
        Xenco(n) has the following format:
1809
           Xenco( n ) = "011 s xxxx"
1810
        where.
1811
            "011" is the code word
1812
            "s" is the sign bit
            "xxxx" is the four bit value field
1813
1814
        The coder equation is described as follows:
1815
           if (Xdiff(n) < 0) then
1816
               sign = 1
1817
           else
1818
               sign = 0
1819
           endif
           value = (abs(Xdiff(n)) - 64) / 4
1820
        E.2.1.4
                   PCM for 10-8-10 Coder
1821
        Xenco(n) has the following format:
1822
           Xenco( n ) = "1 xxxxxxx"
1823
        where,
1824
           "1" is the code word
1825
            the sign bit is not used
1826
            "xxxxxxx" is the seven bit value field
1827
        The coder equation is described as follows:
1828
           value = Xorig( n ) / 8
        E.2.2
                 Coder for 10–7–10 Data Compression
1829
        The 10–7–10 coder offers 30% bit rate reduction with high image quality.
1830
        Pixels without prediction are encoded using the following formula:
1831
           Xenco(n) = Xorig(n) / 8
1832
        To avoid a full-zero encoded value, the following check is performed:
1833
            if (Xenco(n) == 0) then
1834
               Xenco(n) = 1
1835
        Pixels with prediction are encoded using the following formula:
1836
           if (abs(Xdiff(n)) < 8) then
1837
               use DPCM1
1838
           else if (abs(Xdiff(n)) < 16) then
1839
               use DPCM2
1840
           else if (abs(Xdiff(n)) < 32) then
1841
               use DPCM3
1842
           else if (abs(Xdiff(n)) < 160) then
1843
               use DPCM4
1844
           else
1845
               use PCM
1846
           endif
```

E.2.2.1 DPCM1 for 10-7-10 Coder

1847 **Xenco**(\mathbf{n}) has the following format:

Version 1.3 Specification for CSI-2

```
1848
           Xenco( n ) = "000 s xxx"
1849
        where.
1850
           "000" is the code word
1851
           "s" is the sign bit
1852
           "xxx" is the three bit value field
1853
        The coder equation is described as follows:
1854
           if (Xdiff(n) <= 0) then
1855
               sign = 1
1856
           else
1857
               sign = 0
1858
           endif
1859
           value = abs(Xdiff( n ))
1860
        Note: Zero code has been avoided (0 is sent as -0).
        E.2.2.2
                  DPCM2 for 10-7-10 Coder
1861
       Xenco(n) has the following format:
1862
           Xenco(n) = "0010 s xx"
1863
        where.
1864
           "0010" is the code word
1865
           "s" is the sign bit
1866
           "xx" is the two bit value field
1867
        The coder equation is described as follows:
1868
           if (Xdiff(n) < 0) then
1869
               sign = 1
1870
           else
1871
               sign = 0
1872
           endif
1873
           value = (abs(Xdiff(n)) - 8) / 2
        E.2.2.3
                  DPCM3 for 10-7-10 Coder
1874
        Xenco(n) has the following format:
1875
           Xenco(n) = "0011 s xx"
1876
        where,
1877
           "0011" is the code word
1878
           "s" is the sign bit
1879
           "xx" is the two bit value field
1880
        The coder equation is described as follows:
1881
           if (Xdiff(n) < 0) then
1882
               sign = 1
1883
           else
1884
               sign = 0
1885
           endif
1886
           value = (abs(Xdiff(n)) - 16) / 4
        E.2.2.4
                  DPCM4 for 10-7-10 Coder
1887
        Xenco(n) has the following format:
1888
           Xenco(n) = "01 s xxxx"
1889
        where,
```

Specification for CSI-2 Version 1.3

29-May-2014

```
1890
            "01" is the code word
1891
            "s" is the sign bit
1892
            "xxxx" is the four bit value field
1893
        The coder equation is described as follows:
1894
           if (Xdiff(n) < 0) then
1895
               sign = 1
1896
           else
1897
               sign = 0
1898
           endif
1899
           value = (abs(Xdiff(n)) - 32) / 8
        E.2.2.5
                   PCM for 10-7-10 Coder
1900
        Xenco( n ) has the following format:
1901
           Xenco( n ) = "1 xxxxxx"
1902
        where.
1903
           "1" is the code word
1904
            the sign bit is not used
            "xxxxxx" is the six bit value field
1905
1906
        The coder equation is described as follows:
1907
           value = Xorig( n ) / 16
        E.2.3
                 Coder for 10–6–10 Data Compression
1908
        The 10–6–10 coder offers 40% bit rate reduction with acceptable image quality.
1909
        Pixels without prediction are encoded using the following formula:
1910
           Xenco(n) = Xorig(n) / 16
        To avoid a full-zero encoded value, the following check is performed:
1911
1912
            if (Xenco(n) == 0) then
1913
               Xenco(n) = 1
1914
           endif
1915
        Pixels with prediction are encoded using the following formula:
1916
           if (abs(Xdiff(n)) < 1) then
1917
               use DPCM1
1918
           else if (abs(Xdiff(n)) < 3) then
1919
               use DPCM2
1920
           else if (abs(Xdiff(n)) < 11) then
1921
               use DPCM3
1922
           else if (abs(Xdiff(n)) < 43) then
1923
               use DPCM4
           else if (abs(Xdiff(n)) < 171) then
1924
1925
               use DPCM5
1926
           else
1927
               use PCM
1928
           endif
        E.2.3.1
                   DPCM1 for 10-6-10 Coder
1929
        Xenco(n) has the following format:
1930
           Xenco( n ) = "00000 s"
1931
```

where,

Version 1.3 Specification for CSI-2

```
"00000" is the code word
1932
1933
           "s" is the sign bit
1934
           the value field is not used
1935
        The coder equation is described as follows:
1936
           sign = 1
1937
           Note: Zero code has been avoided (0 is sent as -0).
        E.2.3.2
                  DPCM2 for 10-6-10 Coder
1938
        Xenco(n) has the following format:
1939
           Xenco(n) = "00001 s"
1940
        where,
1941
           "00001" is the code word
1942
           "s" is the sign bit
1943
           the value field is not used
1944
        The coder equation is described as follows:
1945
           if (Xdiff(n) < 0) then
1946
               sign = 1
1947
           else
1948
               sign = 0
1949
           endif
        E.2.3.3
                  DPCM3 for 10-6-10 Coder
1950
        Xenco(n) has the following format:
1951
           Xenco(n) = "0001 s x"
1952
        where.
1953
           "0001" is the code word
1954
           "s" is the sign bit
1955
           "x" is the one bit value field
1956
        The coder equation is described as follows:
1957
           if (Xdiff(n) < 0) then
1958
               sign = 1
1959
           else
1960
               sign = 0
1961
           value = (abs(Xdiff(n)) - 3) / 4
1962
           endif
        E.2.3.4
                  DPCM4 for 10-6-10 Coder
1963
        Xenco(n) has the following format:
1964
           Xenco(n) = "001 s xx"
1965
        where.
1966
           "001" is the code word
1967
           "s" is the sign bit
1968
           "xx" is the two bit value field
1969
        The coder equation is described as follows:
1970
           if (Xdiff(n) < 0) then
1971
               sign = 1
1972
           else
1973
               sign = 0
1974
           endif
1975
           value = (abs(Xdiff( n )) - 11) / 8
```

Specification for CSI-2 Version 1.3
29-May-2014

E.2.3.5 DPCM5 for 10-6-10 Coder

```
1976
        Xenco(n) has the following format:
1977
           Xenco(n) = "01 s xxx"
1978
        where.
1979
           "01" is the code word
1980
            "s" is the sign bit
1981
            "xxx" is the three bit value field
1982
        The coder equation is described as follows:
1983
            if (Xdiff(n) < 0) then
1984
               sign = 1
1985
           else
1986
               sign = 0
1987
           endif
1988
           value = (abs(Xdiff( n )) - 43) / 16
        E.2.3.6
                   PCM for 10-6-10 Coder
1989
        Xenco(n) has the following format:
1990
           Xenco(n) = "1 xxxxx"
1991
        where,
1992
           "1" is the code word
1993
            the sign bit is not used
            "xxxxx" is the five bit value field
1994
1995
        The coder equation is described as follows:
1996
           value = Xorig( n ) / 32
        E.2.4
                 Coder for 12-8-12 Data Compression
1997
        The 12–8–12 coder offers 33% bit rate reduction with very high image quality.
1998
        Pixels without prediction are encoded using the following formula:
1999
           Xenco(n) = Xorig(n) / 16
2000
        To avoid a full-zero encoded value, the following check is performed:
2001
            if (Xenco(n) == 0) then
2002
               Xenco(n) = 1
2003
            endif
2004
        Pixels with prediction are encoded using the following formula:
2005
           if (abs(Xdiff(n)) < 8) then
2006
               use DPCM1
2007
           else if (abs(Xdiff(n)) < 40) then
2008
               use DPCM2
2009
           else if (abs(Xdiff(n)) < 104) then
2010
               use DPCM3
2011
           else if (abs(Xdiff(n)) < 232) then
2012
               use DPCM4
2013
           else if (abs(Xdiff(n)) < 360) then
2014
               use DPCM5
2015
           else
2016
               use PCM
        E.2.4.1
                   DPCM1 for 12-8-12 Coder
2017
        Xenco(n) has the following format:
2018
           Xenco( n ) = "0000 s xxx"
```

Copyright © 2005-2014 MIPI Alliance, Inc.

Version 1.3 Specification for CSI-2

```
2019
        where,
2020
           "0000" is the code word
2021
           "s" is the sign bit
2022
           "xxx" is the three bit value field
2023
        The coder equation is described as follows:
2024
           if (Xdiff(n) \le 0) then
2025
               sign = 1
2026
           else
2027
               sign = 0
2028
           endif
2029
           value = abs(Xdiff( n ))
2030
        Note: Zero code has been avoided (0 is sent as -0).
                  DPCM2 for 12-8-12 Coder
        E.2.4.2
2031
        Xenco(n) has the following format:
2032
           Xenco( n ) = "011 s xxxx"
2033
        where,
2034
           "011" is the code word
2035
           "s" is the sign bit
2036
           "xxxx" is the four bit value field
2037
       The coder equation is described as follows:
2038
           if (Xdiff(n) < 0) then
2039
               sign = 1
2040
           else
2041
               sign = 0
2042
           endif
2043
           value = (abs(Xdiff(n)) - 8) / 2
        E.2.4.3
                  DPCM3 for 12-8-12 Coder
2044
       Xenco(n) has the following format:
2045
           Xenco(n) = "010 s xxxx"
2046
        where,
2047
           "010" is the code word
2048
           "s" is the sign bit
2049
           "xxxx" is the four bit value field
2050
        The coder equation is described as follows:
2051
           if (Xdiff(n) < 0) then
2052
               sign = 1
2053
           else
2054
               sign = 0
2055
           endif
2056
           value = (abs(Xdiff(n)) - 40) / 4
        E.2.4.4
                  DPCM4 for 12-8-12 Coder
2057
        Xenco(n) has the following format:
2058
           Xenco( n ) = "001 s xxxx"
2059
        where.
2060
           "001" is the code word
2061
           "s" is the sign bit
2062
           "xxxx" is the four bit value field
```

Specification for CSI-2 Version 1.3

```
2063
        The coder equation is described as follows:
2064
            if (Xdiff(n) < 0) then
2065
               sign = 1
2066
           else
2067
               sign = 0
2068
           endif
2069
           value = (abs(Xdiff(n)) - 104) / 8
        E.2.4.5
                   DPCM5 for 12-8-12 Coder
2070
        Xenco(n) has the following format:
2071
           Xenco( n ) = "0001 s xxx"
2072
        where.
2073
            "0001" is the code word
2074
            "s" is the sign bit
2075
            "xxx" is the three bit value field
2076
        The coder equation is described as follows:
2077
           if (Xdiff(n) < 0) then
2078
               sign = 1
2079
           else
2080
               sign = 0
2081
           endif
2082
           value = (abs(Xdiff( n )) - 232) / 16
        E.2.4.6
                   PCM for 12-8-12 Coder
2083
        Xenco(n) has the following format:
2084
           Xenco( n ) = "1 xxxxxxx"
2085
        where.
2086
           "1" is the code word
2087
            the sign bit is not used
2088
            "xxxxxxx" is the seven bit value field
2089
        The coder equation is described as follows:
2090
           value = Xorig(n) / 32
        E.2.5
                 Coder for 12-7-12 Data Compression
2091
        The 12–7–12 coder offers 42% bit rate reduction with high image quality.
2092
        Pixels without prediction are encoded using the following formula:
2093
           Xenco(n) = Xorig(n) / 32
2094
        To avoid a full-zero encoded value, the following check is performed:
2095
           if (Xenco(n) == 0) then
2096
               Xenco(n) = 1
2097
           endif
2098
        Pixels with prediction are encoded using the following formula:
2099
           if (abs(Xdiff(n)) < 4) then
2100
               use DPCM1
2101
           else if (abs(Xdiff(n)) < 12) then
2102
               use DPCM2
2103
           else if (abs(Xdiff(n)) < 28) then
2104
               use DPCM3
2105
           else if (abs(Xdiff(n)) < 92) then
2106
               use DPCM4
```

Version 1.3 Specification for CSI-2

29-May-2014

```
2107
           else if (abs(Xdiff(n)) < 220) then
2108
               use DPCM5
2109
           else if (abs(Xdiff(n)) < 348) then
2110
              use DPCM6
2111
           else
2112
              use PCM
2113
           endif
        E.2.5.1
                  DPCM1 for 12-7-12 Coder
2114
       Xenco(n) has the following format:
2115
           Xenco(n) = "0000 s xx"
2116
       where,
2117
           "0000" is the code word
2118
           "s" is the sign bit
2119
           "xx" is the two bit value field
2120
       The coder equation is described as follows:
2121
           if (Xdiff(n) \le 0) then
2122
               sign = 1
2123
           else
2124
               sign = 0
2125
           endif
2126
           value = abs(Xdiff( n ))
2127
        Note: Zero code has been avoided (0 is sent as -0).
        E.2.5.2
                  DPCM2 for 12-7-12 Coder
2128
       Xenco(n) has the following format:
2129
           Xenco(n) = "0001 s xx"
2130
       where.
2131
           "0001" is the code word
2132
           "s" is the sign bit
2133
           "xx" is the two bit value field
2134
        The coder equation is described as follows:
2135
           if (Xdiff(n) < 0) then
2136
               sign = 1
2137
           else
2138
               sign = 0
2139
           endif
2140
           value = (abs(Xdiff(n)) - 4) / 2
        E.2.5.3
                  DPCM3 for 12-7-12 Coder
2141
        Xenco(n) has the following format:
2142
           Xenco(n) = "0010 s xx"
2143
        where.
2144
           "0010" is the code word
2145
           "s" is the sign bit
2146
           "xx" is the two bit value field
2147
        The coder equation is described as follows:
2148
           if (Xdiff(n) < 0) then
2149
```

sign = 1

else

2150

Specification for CSI-2 Version 1.3

```
2151
               sign = 0
2152
           endif
2153
           value = (abs(Xdiff(n)) - 12) / 4
        E.2.5.4
                  DPCM4 for 12-7-12 Coder
2154
       Xenco(n) has the following format:
2155
           Xenco( n ) = "010 s xxx"
2156
        where,
2157
           "010" is the code word
2158
           "s" is the sign bit
2159
           "xxx" is the three bit value field
2160
        The coder equation is described as follows:
2161
           if (Xdiff(n) < 0) then
2162
               sign = 1
2163
           else
2164
               sign = 0
2165
           endif
2166
           value = (abs(Xdiff(n)) - 28) / 8
        E.2.5.5
                  DPCM5 for 12-7-12 Coder
2167
       Xenco(n) has the following format:
2168
           Xenco( n ) = "011 s xxx"
2169
        where,
2170
           "011" is the code word
2171
           "s" is the sign bit
2172
           "xxx" is the three bit value field
2173
        The coder equation is described as follows:
2174
           if (Xdiff(n) < 0) then
2175
               sign = 1
2176
           else
2177
               sign = 0
2178
           endif
2179
           value = (abs(Xdiff(n)) - 92) / 16
        E.2.5.6
                  DPCM6 for 12-7-12 Coder
2180
        Xenco(n) has the following format:
2181
           Xenco(n) = "0011 s xx"
2182
2183
           "0011" is the code word
2184
           "s" is the sign bit
2185
           "xx" is the two bit value field
2186
        The coder equation is described as follows:
2187
           if (Xdiff(n) < 0) then
2188
               sign = 1
2189
           else
2190
               sign = 0
2191
           endif
2192
           value = (abs(Xdiff(n)) - 220) / 32
```

29-May-2014

2237

value = abs(Xdiff(n))

```
E.2.5.7 PCM for 12-7-12 Coder
```

```
2193
        Xenco(n) has the following format:
2194
           Xenco( n ) = "1 xxxxxx"
2195
        where.
2196
           "1" is the code word
2197
           the sign bit is not used
2198
           "xxxxxx" is the six bit value field
2199
        The coder equation is described as follows:
2200
           value = Xorig( n ) / 64
        E.2.6
                 Coder for 12-6-12 Data Compression
2201
        The 12–6–12 coder offers 50% bit rate reduction with acceptable image quality.
2202
        Pixels without prediction are encoded using the following formula:
2203
           Xenco(n) = Xorig(n) / 64
2204
        To avoid a full-zero encoded value, the following check is performed:
2205
           if (Xenco(n) == 0) then
2206
               Xenco(n) = 1
2207
           endif
2208
        Pixels with prediction are encoded using the following formula:
2209
           if (abs(Xdiff(n)) < 2) then
2210
               use DPCM1
2211
           else if (abs(Xdiff(n)) < 10) then
2212
               use DPCM3
2213
           else if (abs(Xdiff(n)) < 42) then
2214
               use DPCM4
2215
           else if (abs(Xdiff(n)) < 74) then
2216
               use DPCM5
2217
           else if (abs(Xdiff(n)) < 202) then
2218
               use DPCM6
           else if (abs(Xdiff(n)) < 330) then
2219
2220
               use DPCM7
2221
           else
2222
               use PCM
2223
           endif
2224
        Note: DPCM2 is not used.
        E.2.6.1
                  DPCM1 for 12-6-12 Coder
2225
        Xenco(n) has the following format:
2226
           Xenco(n) = "0000 s x"
2227
        where,
2228
           "0000" is the code word
2229
            "s" is the sign bit
2230
           "x" is the one bit value field
2231
        The coder equation is described as follows:
2232
           if (Xdiff(n) <= 0) then
2233
               sign = 1
2234
           else
2235
               sign = 0
2236
           endif
```

Specification for CSI-2 Version 1.3
29-May-2014

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

```
E.2.6.2 DPCM3 for 12-6-12 Coder
```

```
2239
       Xenco(n) has the following format:
2240
           Xenco(n) = "0001 s x"
2241
        where,
2242
           "0001" is the code word
2243
           "s" is the sign bit
2244
           "x" is the one bit value field
2245
        The coder equation is described as follows:
2246
           if (Xdiff(n) < 0) then
2247
               sign = 1
2248
           else
2249
               sign = 0
2250
           endif
2251
           value = (abs(Xdiff(n)) - 2) / 4
```

E.2.6.3 DPCM4 for 12–6–12 Coder

```
2252 Xenco(\mathbf{n}) has the following format:
```

```
2253
           Xenco(n) = "010 s xx"
2254
       where,
2255
           "010" is the code word
2256
           "s" is the sign bit
2257
           "xx" is the two bit value field
2258
       The coder equation is described as follows:
2259
           if (Xdiff(n) < 0) then
2260
              sign = 1
2261
           else
2262
               sign = 0
2263
           endif
2264
           value = (abs(Xdiff(n)) - 10) / 8
```

E.2.6.4 DPCM5 for 12-6-12 Coder

```
    2265 Xenco(n) has the following format:
    2266 xenco(n) = "0010 s x"
    2267 where,
```

2268

2269 "s" is the sign bit
2270 "x" is the one bit value field

"0010" is the code word

The coder equation is described as follows:

```
2272     if (Xdiff( n ) < 0) then
2273         sign = 1
2274     else
2275         sign = 0
2276     endif
2277     value = (abs(Xdiff( n )) - 42) / 16</pre>
```

E.2.6.5 DPCM6 for 12-6-12 Coder

```
Xenco(\mathbf{n}) has the following format:
```

```
2279 Xenco(n) = "011 s xx"
```

140

Copyright © 2005-2014 MIPI Alliance, Inc.

Version 1.3 Specification for CSI-2

29-May-2014

2319

2320

use DPCM1

else if (Xenco(n) & 0xe0 == 0x40) then

```
2280
        where,
2281
            "011" is the code word
2282
            "s" is the sign bit
2283
            "xx" is the two bit value field
2284
        The coder equation is described as follows:
2285
            if (Xdiff(n) < 0) then
2286
               sign = 1
2287
            else
2288
               sign = 0
2289
            endif
2290
            value = (abs(Xdiff(n)) - 74) / 32
                   DPCM7 for 12-6-12 Coder
        E.2.6.6
2291
        Xenco(n) has the following format:
2292
            Xenco(n) = "0011 s x"
2293
        where,
2294
            "0011" is the code word
2295
            "s" is the sign bit
2296
            "x" is the one bit value field
2297
        The coder equation is described as follows:
2298
            if (Xdiff(n) < 0) then
2299
               sign = 1
2300
            else
2301
               sign = 0
2302
            endif
2303
            value = (abs(Xdiff(n)) - 202) / 64
        E.2.6.7
                   PCM for 12-6-12 Coder
2304
        Xenco(n) has the following format:
2305
            Xenco(n) = "1 xxxxx"
2306
        where.
2307
            "1" is the code word
2308
            the sign bit is not used
2309
            "xxxxx" is the five bit value field
2310
        The coder equation is described as follows:
2311
            value = Xorig( n ) / 128
        E.3
                Decoders
2312
        There are six different decoders available, one for each data compression scheme.
2313
        For all decoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
2314
        for predicted pixels.
                 Decoder for 10-8-10 Data Compression
2315
        Pixels without prediction are decoded using the following formula:
2316
            Xdeco(n) = 4 * Xenco(n) + 2
2317
        Pixels with prediction are decoded using the following formula:
2318
            if (Xenco( n ) & 0xc0 == 0x00) then
```

Specification for CSI-2 Version 1.3

```
2321
              use DPCM2
2322
           else if (Xenco(n) & 0xe0 == 0x60) then
2323
              use DPCM3
2324
           else
2325
              use PCM
2326
           endif
        E.3.1.1
                  DPCM1 for 10-8-10 Decoder
2327
       Xenco(n) has the following format:
2328
           Xenco(n) = "00 s xxxxx"
2329
       where.
2330
           "00" is the code word
2331
           "s" is the sign bit
2332
           "xxxxx" is the five bit value field
2333
       The decoder equation is described as follows:
2334
           sign = Xenco(n) & 0x20
2335
           value = Xenco(n) & 0x1f
2336
           if (sign > 0) then
2337
              Xdeco( n ) = Xpred( n ) - value
2338
           else
2339
              Xdeco( n ) = Xpred( n ) + value
2340
           endif
       E.3.1.2
                  DPCM2 for 10-8-10 Decoder
2341
       Xenco(n) has the following format:
2342
           Xenco( n ) = "010 s xxxx"
2343
       where.
2344
           "010" is the code word
2345
           "s" is the sign bit
2346
           "xxxx" is the four bit value field
2347
        The decoder equation is described as follows:
2348
           sign = Xenco(n) & 0x10
2349
           value = 2 * (Xenco(n) & 0xf) + 32
2350
           if (sign > 0) then
2351
              Xdeco( n ) = Xpred( n ) - value
2352
           else
2353
              Xdeco( n ) = Xpred( n ) + value
2354
           endif
                  DPCM3 for 10-8-10 Decoder
        E.3.1.3
2355
       Xenco(n) has the following format:
2356
           Xenco(n) = "011 s xxxx"
2357
       where,
2358
           "011" is the code word
2359
           "s" is the sign bit
2360
           "xxxx" is the four bit value field
2361
        The decoder equation is described as follows:
2362
           sign = Xenco(n) & 0x10
2363
           value = 4 * (Xenco( n ) & 0xf) + 64 + 1
2364
           if (sign > 0) then
2365
              Xdeco( n ) = Xpred( n ) - value
        142
                                 Copyright © 2005-2014 MIPI Alliance, Inc.
```

Version 1.3 Specification for CSI-2

29-May-2014

```
2366
              if (Xdeco(n) < 0) then
2367
                 Xdeco(n) = 0
2368
              endif
2369
          else
2370
              Xdeco( n ) = Xpred( n ) + value
2371
              if (Xdeco(n) > 1023) then
2372
                 Xdeco(n) = 1023
2373
              endif
           endif
2374
       E.3.1.4
                 PCM for 10-8-10 Decoder
2375
       Xenco(n) has the following format:
2376
           Xenco( n ) = "1 xxxxxxx"
```

2377 where,

2390

```
2378 "1" is the code word
2379 the sign bit is not used
"xxxxxxxx" is the seven bit value field
```

The codec equation is described as follows:

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

2389 Pixels without prediction are decoded using the following formula:

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

Pixels with prediction are decoded using the following formula:

```
2392
           if (Xenco( \mathbf{n} ) & 0x70 == 0x00) then
2393
              use DPCM1
2394
           else if (Xenco( n ) & 0x78 == 0x10) then
2395
              use DPCM2
2396
           else if (Xenco(n) & 0x78 == 0x18) then
2397
              use DPCM3
2398
           else if (Xenco( n ) & 0x60 == 0x20) then
2399
              use DPCM4
2400
           else
2401
              use PCM
2402
           endif
```

E.3.2.1 DPCM1 for 10-7-10 Decoder

```
Xenco(\mathbf{n}) has the following format:
```

Specification for CSI-2 Version 1.3

```
2411
           value = Xenco(n) & 0x7
2412
           if (sign > 0) then
2413
              Xdeco( n ) = Xpred( n ) - value
2414
2415
              Xdeco( n ) = Xpred( n ) + value
2416
           endif
       E.3.2.2
                  DPCM2 for 10-7-10 Decoder
2417
       Xenco(n) has the following format:
2418
           Xenco(n) = "0010 s xx"
2419
       where.
2420
           "0010" is the code word
2421
           "s" is the sign bit
           "xx" is the two bit value field
2422
2423
       The codec equation is described as follows:
2424
           sign = Xenco(n) & 0x4
2425
           value = 2 * (Xenco(n) & 0x3) + 8
2426
           if (sign > 0) then
2427
              Xdeco( n ) = Xpred( n ) - value
2428
           else
2429
              Xdeco( n ) = Xpred( n ) + value
2430
           endif
       E.3.2.3
                  DPCM3 for 10-7-10 Decoder
2431
       Xenco(n) has the following format:
2432
           Xenco(n) = "0011 s xx"
2433
       where.
2434
           "0011" is the code word
2435
           "s" is the sign bit
           "xx" is the two bit value field
2436
2437
       The codec equation is described as follows:
2438
           sign = Xenco(n) & 0x4
2439
           value = 4 * (Xenco(n) & 0x3) + 16 + 1
2440
           if (sign > 0) then
2441
              Xdeco( n ) = Xpred( n ) - value
2442
              if (Xdeco(n) < 0) then
2443
                 Xdeco(n) = 0
2444
              endif
2445
           else
2446
              Xdeco( n ) = Xpred( n ) + value
2447
              if (Xdeco(n) > 1023) then
2448
                  Xdeco(n) = 1023
2449
              endif
2450
           endif
       E.3.2.4
                  DPCM4 for 10-7-10 Decoder
2451
       Xenco( n ) has the following format:
2452
           Xenco(n) = "01 s xxxx"
2453
       where,
```

```
2454
           "01" is the code word
2455
           "s" is the sign bit
2456
           "xxxx" is the four bit value field
2457
       The codec equation is described as follows:
2458
           sign = Xenco(n) & 0x10
2459
           value = 8 * (Xenco(n) & 0xf) + 32 + 3
2460
           if (sign > 0) then
2461
              Xdeco( n ) = Xpred( n ) - value
2462
              if (Xdeco(n) < 0) then
2463
                  Xdeco(n) = 0
2464
              endif
2465
           else
2466
              Xdeco( n ) = Xpred( n ) + value
2467
              if (Xdeco(n) > 1023) then
2468
                  Xdeco(n) = 1023
2469
              endif
2470
           endif
       E.3.2.5
                  PCM for 10-7-10 Decoder
2471
       Xenco( n ) has the following format:
2472
           Xenco( n ) = "1 xxxxxx"
2473
       where,
2474
           "1" is the code word
2475
           the sign bit is not used
2476
           "xxxxxx" is the six bit value field
2477
       The codec equation is described as follows:
2478
           value = 16 * (Xenco( n ) & 0x3f)
2479
           if (value > Xpred( n )) then
2480
              Xdeco(n) = value + 7
2481
           else
2482
              Xdeco(n) = value + 8
2483
           endif
        E.3.3
                Decoder for 10-6-10 Data Compression
2484
       Pixels without prediction are decoded using the following formula:
2485
           Xdeco(n) = 16 * Xenco(n) + 8
2486
        Pixels with prediction are decoded using the following formula:
2487
           if (Xenco(\mathbf{n}) & 0x3e == 0x00) then
2488
              use DPCM1
2489
           else if (Xenco(n) & 0x3e == 0x02) then
2490
              use DPCM2
2491
           else if (Xenco(n) & 0x3c == 0x04) then
2492
              use DPCM3
2493
           else if (Xenco(n) & 0x38 == 0x08) then
2494
              use DPCM4
2495
           else if (Xenco(n) & 0x30 == 0x10) then
2496
              use DPCM5
2497
           else
2498
              use PCM
2499
           endif
```

Specification for CSI-2 Version 1.3
29-May-2014

E.3.3.1 DPCM1 for 10-6-10 Decoder

```
2500
       Xenco(n) has the following format:
2501
           Xenco(n) = "00000 s"
2502
       where.
2503
           "00000" is the code word
2504
           "s" is the sign bit
2505
           the value field is not used
2506
        The codec equation is described as follows:
2507
           Xdeco(n) = Xpred(n)
        E.3.3.2
                  DPCM2 for 10-6-10 Decoder
2508
       Xenco(n) has the following format:
2509
           Xenco(n) = "00001 s"
2510
       where.
2511
           "00001" is the code word
2512
           "s" is the sign bit
2513
           the value field is not used
2514
       The codec equation is described as follows:
2515
           sign = Xenco(n) & 0x1
2516
           value = 1
2517
           if (sign > 0) then
2518
              Xdeco( n ) = Xpred( n ) - value
2519
2520
              Xdeco( n ) = Xpred( n ) + value
2521
           endif
       E.3.3.3
                  DPCM3 for 10-6-10 Decoder
2522
       Xenco(n) has the following format:
2523
           Xenco(n) = "0001 s x"
2524
       where.
2525
           "0001" is the code word
2526
           "s" is the sign bit
2527
           "x" is the one bit value field
2528
       The codec equation is described as follows:
2529
           sign = Xenco(n) & 0x2
2530
           value = 4 * (Xenco(n) & 0x1) + 3 + 1
2531
           if (sign > 0) then
2532
              Xdeco( n ) = Xpred( n ) - value
2533
              if (Xdeco(n) < 0) then
2534
                  Xdeco(n) = 0
2535
              endif
2536
           else
2537
              Xdeco( n ) = Xpred( n ) + value
2538
              if (Xdeco(n) > 1023) then
2539
                  Xdeco(n) = 1023
2540
              endif
2541
           endif
```

E.3.3.4 DPCM4 for 10-6-10 Decoder

Xenco(\mathbf{n}) has the following format:

146

```
2543
           Xenco(n) = "001 s xx"
2544
       where.
2545
           "001" is the code word
2546
           "s" is the sign bit
2547
           "xx" is the two bit value field
2548
       The codec equation is described as follows:
2549
           sign = Xenco(n) & 0x4
2550
           value = 8 * (Xenco(n) & 0x3) + 11 + 3
2551
           if (sign > 0) then
2552
              Xdeco( n ) = Xpred( n ) - value
2553
              if (Xdeco(n) < 0) then
2554
                  Xdeco(n) = 0
2555
              endif
2556
           else
2557
              Xdeco( n ) = Xpred( n ) + value
2558
              if (Xdeco(n) > 1023) then
2559
                  Xdeco(n) = 1023
2560
              endif
2561
           endif
       E.3.3.5
                  DPCM5 for 10-6-10 Decoder
2562
       Xenco(n) has the following format:
2563
           Xenco(n) = "01 s xxx"
2564
       where.
2565
           "01" is the code word
2566
           "s" is the sign bit
2567
           "xxx" is the three bit value field
2568
       The codec equation is described as follows:
2569
           sign = Xenco(n) \& 0x8
           value = 16 * (Xenco(n) & 0x7) + 43 + 7
2570
2571
           if (sign > 0) then
2572
              Xdeco( n ) = Xpred( n ) - value
2573
              if (Xdeco(n) < 0) then
2574
                  Xdeco(n) = 0
2575
              endif
2576
           else
2577
              Xdeco( n ) = Xpred( n ) + value
2578
              if (Xdeco(n) > 1023) then
2579
                  Xdeco(n) = 1023
2580
              endif
2581
           endif
       E.3.3.6
                  PCM for 10-6-10 Decoder
2582
       Xenco(n) has the following format:
2583
           Xenco( n ) = "1 xxxxx"
2584
       where.
2585
           "1" is the code word
2586
           the sign bit is not used
2587
           "xxxxx" is the five bit value field
2588
       The codec equation is described as follows:
2589
           value = 32 * (Xenco(n) & 0x1f)
```

```
29-May-2014
```

```
2590
           if (value > Xpred( n )) then
2591
              Xdeco(n) = value + 15
2592
           else
2593
              Xdeco(n) = value + 16
2594
           endif
       E.3.4
                Decoder for 12-8-12 Data Compression
2595
       Pixels without prediction are decoded using the following formula:
2596
           Xdeco(n) = 16 * Xenco(n) + 8
2597
       Pixels with prediction are decoded using the following formula:
2598
           if (Xenco( \mathbf{n} ) & 0xf0 == 0x00) then
2599
              use DPCM1
2600
           else if (Xenco(n) & 0xe0 == 0x60) then
2601
              use DPCM2
2602
           else if (Xenco(n) & 0xe0 == 0x40) then
2603
              use DPCM3
2604
           else if (Xenco(n) & 0xe0 == 0x20) then
2605
              use DPCM4
2606
           else if (Xenco(n) & 0xf0 == 0x10) then
2607
              use DPCM5
2608
           else
2609
              use PCM
2610
           endif
       E.3.4.1
                  DPCM1 for 12-8-12 Decoder
2611
       Xenco(n) has the following format:
2612
           Xenco(n) = "0000 s xxx"
2613
2614
           "0000" is the code word
2615
           "s" is the sign bit
2616
           "xxx" is the three bit value field
2617
       The codec equation is described as follows:
2618
           sign = Xenco(n) & 0x8
2619
           value = Xenco(n) & 0x7
2620
           if (sign > 0) then
2621
              Xdeco( n ) = Xpred( n ) - value
2622
2623
              Xdeco( n ) = Xpred( n ) + value
2624
           endif
        E.3.4.2
                  DPCM2 for 12-8-12 Decoder
2625
       Xenco(n) has the following format:
2626
           Xenco( n ) = "011 s xxxx"
2627
       where.
2628
           "011" is the code word
2629
           "s" is the sign bit
2630
           "xxxx" is the four bit value field
2631
       The codec equation is described as follows:
2632
           sign = Xenco(n) & 0x10
2633
           value = 2 * (Xenco(n) & 0xf) + 8
2634
           if (sign > 0) then
```

Version 1.3 Specification for CSI-2

```
2635
              Xdeco( n ) = Xpred( n ) - value
2636
           else
2637
              Xdeco( n ) = Xpred( n ) + value
2638
           endif
       E.3.4.3
                  DPCM3 for 12-8-12 Decoder
2639
       Xenco(n) has the following format:
2640
           Xenco( n ) = "010 s xxxx"
2641
       where,
2642
           "010" is the code word
2643
           "s" is the sign bit
2644
           "xxxx" is the four bit value field
2645
       The codec equation is described as follows:
2646
           sign = Xenco(n) & 0x10
2647
           value = 4 * (Xenco(n) & 0xf) + 40 + 1
2648
           if (sign > 0) then
2649
              Xdeco( n ) = Xpred( n ) - value
2650
              if (Xdeco(n) < 0) then
2651
                  Xdeco(n) = 0
2652
              endif
2653
           else
2654
              Xdeco( n ) = Xpred( n ) + value
2655
              if (Xdeco(n) > 4095) then
2656
                  Xdeco(n) = 4095
2657
              endif
2658
           endif
       E.3.4.4
                  DPCM4 for 12-8-12 Decoder
       Xenco(n) has the following format:
2659
2660
           Xenco( n ) = "001 s xxxx"
2661
       where.
2662
           "001" is the code word
2663
           "s" is the sign bit
2664
           "xxxx" is the four bit value field
2665
       The codec equation is described as follows:
2666
           sign = Xenco(n) & 0x10
2667
           value = 8 * (Xenco(n) & 0xf) + 104 + 3
2668
           if (sign > 0) then
2669
              Xdeco( n ) = Xpred( n ) - value
2670
              if (Xdeco(n) < 0) then
2671
                  Xdeco(n) = 0
2672
              endif
2673
           else
2674
              Xdeco( n ) = Xpred( n ) + value
2675
              if (Xdeco(n) > 4095)
2676
                  Xdeco(n) = 4095
2677
              endif
           endif
2678
       E.3.4.5
                  DPCM5 for 12-8-12 Decoder
2679
       Xenco(n) has the following format:
```

```
2680
           Xenco( n ) = "0001 s xxx"
```

Specification for CSI-2 Version 1.3

```
2681
       where.
2682
           "0001" is the code word
2683
           "s" is the sign bit
2684
           "xxx" is the three bit value field
2685
        The codec equation is described as follows:
2686
           sign = Xenco(n) \& 0x8
2687
           value = 16 * (Xenco(n) & 0x7) + 232 + 7
2688
           if (sign > 0) then
2689
              Xdeco( n ) = Xpred( n ) - value
2690
               if (Xdeco(n) < 0) then
2691
                  Xdeco(n) = 0
2692
               endif
2693
           else
2694
              Xdeco( n ) = Xpred( n ) + value
2695
               if (Xdeco(n) > 4095) then
2696
                  Xdeco(n) = 4095
2697
               endif
2698
           endif
        E.3.4.6
                  PCM for 12-8-12 Decoder
2699
        Xenco(n) has the following format:
2700
           Xenco( n ) = "1 xxxxxxx"
2701
       where.
2702
           "1" is the code word
2703
           the sign bit is not used
2704
           "xxxxxxx" is the seven bit value field
2705
       The codec equation is described as follows:
2706
           value = 32 * (Xenco(n) & 0x7f)
2707
           if (value > Xpred( n )) then
2708
              Xdeco(n) = value + 15
2709
           else
2710
              Xdeco(n) = value + 16
2711
           endif
        E.3.5
                Decoder for 12-7-12 Data Compression
2712
        Pixels without prediction are decoded using the following formula:
           Xdeco(n) = 32 * Xenco(n) + 16
2713
2714
       Pixels with prediction are decoded using the following formula:
2715
           if (Xenco( n ) & 0x78 == 0x00) then
2716
              use DPCM1
2717
           else if (Xenco( n ) & 0x78 == 0x08) then
2718
              use DPCM2
2719
           else if (Xenco( n ) & 0x78 == 0x10) then
2720
              use DPCM3
2721
           else if (Xenco(\mathbf{n}) & 0x70 == 0x20) then
2722
              use DPCM4
2723
           else if (Xenco(n) & 0x70 == 0x30) then
2724
              use DPCM5
2725
           else if (Xenco(n) & 0x78 == 0x18) then
2726
              use DPCM6
2727
           else
2728
              use PCM
2729
           endif
```

E.3.5.1 DPCM1 for 12-7-12 Decoder

```
2730
       Xenco(n) has the following format:
2731
           Xenco(n) = "0000 s xx"
2732
       where.
2733
           "0000" is the code word
2734
           "s" is the sign bit
2735
           "xx" is the two bit value field
2736
       The codec equation is described as follows:
2737
           sign = Xenco(n) \& 0x4
2738
           value = Xenco(n) & 0x3
2739
           if (sign > 0) then
2740
              Xdeco( n ) = Xpred( n ) - value
2741
2742
              Xdeco( n ) = Xpred( n ) + value
2743
           endif
       E.3.5.2
                  DPCM2 for 12-7-12 Decoder
2744
        Xenco(n) has the following format:
2745
           Xenco( n ) = "0001 s xx"
2746
       where.
2747
           "0001" is the code word
2748
           "s" is the sign bit
2749
           "xx" is the two bit value field
       The codec equation is described as follows:
2750
2751
           sign = Xenco(n) \& 0x4
           value = 2 * (Xenco(n) & 0x3) + 4
2752
2753
           if (sign > 0) then
2754
              Xdeco( n ) = Xpred( n ) - value
2755
           else
2756
              Xdeco( n ) = Xpred( n ) + value
2757
           endif
       E.3.5.3
                  DPCM3 for 12-7-12 Decoder
2758
       Xenco(n) has the following format:
2759
           Xenco(n) = "0010 s xx"
2760
       where.
2761
           "0010" is the code word
2762
           "s" is the sign bit
2763
           "xx" is the two bit value field
2764
       The codec equation is described as follows:
2765
           sign = Xenco(n) & 0x4
2766
           value = 4 * (Xenco(n) & 0x3) + 12 + 1
2767
           if (sign > 0) then
              Xdeco( n ) = Xpred( n ) - value
2768
2769
               if (Xdeco(n) < 0) then
2770
                  Xdeco(n) = 0
2771
               endif
2772
           else
```

Xdeco(n) = Xpred(n) + value

if (Xdeco(n) > 4095) then

Xdeco(n) = 4095

2773

2774

2775

Specification for CSI-2 Version 1.3

```
2776
              endif
2777
           endif
       E.3.5.4
                  DPCM4 for 12-7-12 Decoder
2778
       Xenco(n) has the following format:
2779
           Xenco( n ) = "010 s xxx"
2780
       where.
2781
           "010" is the code word
2782
           "s" is the sign bit
2783
           "xxx" is the three bit value field
2784
       The codec equation is described as follows:
2785
           sign = Xenco(n) \& 0x8
2786
           value = 8 * (Xenco(n) & 0x7) + 28 + 3
2787
           if (sign > 0) then
2788
              Xdeco( n ) = Xpred( n ) - value
2789
              if (Xdeco(n) < 0) then
2790
                  Xdeco(n) = 0
2791
              endif
2792
           else
2793
              Xdeco( n ) = Xpred( n ) + value
2794
              if (Xdeco(n) > 4095) then
2795
                  Xdeco(n) = 4095
2796
              endif
2797
           endif
       E.3.5.5
                  DPCM5 for 12-7-12 Decoder
2798
       Xenco(n) has the following format:
2799
           Xenco(n) = "011 s xxx"
2800
       where,
2801
           "011" is the code word
2802
           "s" is the sign bit
2803
           "xxx" is the three bit value field
2804
       The codec equation is described as follows:
2805
           sign = Xenco(n) \& 0x8
2806
           value = 16 * (Xenco(n) & 0x7) + 92 + 7
2807
           if (sign > 0) then
2808
              Xdeco( n ) = Xpred( n ) - value
2809
              if (Xdeco(n) < 0) then
2810
                  Xdeco(n) = 0
2811
              endif
2812
           else
2813
              Xdeco( n ) = Xpred( n ) + value
2814
              if (Xdeco(n) > 4095) then
2815
                  Xdeco(n) = 4095
2816
              endif
2817
           endif
       E.3.5.6
                  DPCM6 for 12-7-12 Decoder
2818
       Xenco(n) has the following format:
2819
           Xenco(n) = "0011 s xx"
2820
       where.
```

```
2821
           "0011" is the code word
2822
           "s" is the sign bit
2823
           "xx" is the two bit value field
2824
       The codec equation is described as follows:
2825
           sign = Xenco(n) \& 0x4
2826
           value = 32 * (Xenco(n) & 0x3) + 220 + 15
2827
           if (sign > 0) then
2828
              Xdeco( n ) = Xpred( n ) - value
2829
              if (Xdeco(n) < 0) then
2830
                  Xdeco(n) = 0
2831
              endif
2832
           else
              Xdeco( n ) = Xpred( n ) + value
2833
2834
              if (Xdeco(n) > 4095) then
2835
                  Xdeco(n) = 4095
2836
              endif
2837
           endif
       E.3.5.7
                  PCM for 12-7-12 Decoder
2838
       Xenco( n ) has the following format:
2839
           Xenco( n ) = "1 xxxxxx"
2840
       where,
2841
           "1" is the code word
2842
           the sign bit is not used
2843
           "xxxxxx" is the six bit value field
2844
       The codec equation is described as follows:
2845
           value = 64 * (Xenco(n) & 0x3f)
2846
           if (value > Xpred( n )) then
2847
              Xdeco(n) = value + 31
2848
           else
2849
              Xdeco(n) = value + 32
2850
           endif
        E.3.6
                Decoder for 12–6–12 Data Compression
2851
       Pixels without prediction are decoded using the following formula:
2852
           Xdeco(n) = 64 * Xenco(n) + 32
2853
        Pixels with prediction are decoded using the following formula:
2854
           if (Xenco(\mathbf{n}) & 0x3c == 0x00) then
2855
              use DPCM1
2856
           else if (Xenco(n) & 0x3c == 0x04) then
2857
              use DPCM3
2858
           else if (Xenco(n) & 0x38 == 0x10) then
2859
              use DPCM4
2860
           else if (Xenco(n) & 0x3c == 0x08) then
2861
              use DPCM5
2862
           else if (Xenco(n) & 0x38 == 0x18) then
2863
              use DPCM6
2864
           else if (Xenco( n ) & 0x3c == 0x0c) then
2865
              use DPCM7
2866
           else
2867
              use PCM
2868
           endif
2869
       Note: DPCM2 is not used.
```

Specification for CSI-2 Version 1.3
29-May-2014

E.3.6.1 DPCM1 for 12-6-12 Decoder

```
2870
       Xenco(n) has the following format:
2871
           Xenco(n) = "0000 s x"
2872
       where.
2873
           "0000" is the code word
2874
           "s" is the sign bit
2875
           "x" is the one bit value field
2876
        The codec equation is described as follows:
2877
           sign = Xenco(n) \& 0x2
2878
           value = Xenco(n) & 0x1
2879
           if (sign > 0) then
2880
              Xdeco( n ) = Xpred( n ) - value
2881
2882
              Xdeco( n ) = Xpred( n ) + value
2883
           endif
        E.3.6.2
                  DPCM3 for 12-6-12 Decoder
2884
        Xenco(n) has the following format:
2885
           Xenco(n) = "0001 s x"
2886
       where.
2887
           "0001" is the code word
2888
           "s" is the sign bit
2889
           "x" is the one bit value field
       The codec equation is described as follows:
2890
2891
           sign = Xenco(n) \& 0x2
           value = 4 * (Xenco( n ) & 0x1) + 2 + 1
2892
2893
           if (sign > 0) then
2894
              Xdeco( n ) = Xpred( n ) - value
2895
              if (Xdeco(n) < 0) then
2896
                  Xdeco(n) = 0
2897
              endif
2898
           else
2899
              Xdeco( n ) = Xpred( n ) + value
2900
               if (Xdeco(n) > 4095) then
2901
                  Xdeco(n) = 4095
2902
              endif
2903
           endif
       E.3.6.3
                  DPCM4 for 12-6-12 Decoder
2904
       Xenco(n) has the following format:
2905
           Xenco(n) = "010 s xx"
2906
       where,
2907
           "010" is the code word
2908
           "s" is the sign bit
2909
           "xx" is the two bit value field
2910
       The codec equation is described as follows:
2911
           sign = Xenco(n) & 0x4
2912
           value = 8 * (Xenco(n) & 0x3) + 10 + 3
2913
           if (sign > 0) then
2914
              Xdeco( n ) = Xpred( n ) - value
2915
              if (Xdeco(n) < 0) then
```

```
2916
                 Xdeco(n) = 0
2917
              endif
2918
           else
2919
              Xdeco( n ) = Xpred( n ) + value
2920
              if (Xdeco(n) > 4095) then
2921
                 Xdeco(n) = 4095
2922
              endif
2923
           endif
       E.3.6.4
                  DPCM5 for 12-6-12 Decoder
2924
       Xenco(n) has the following format:
2925
           Xenco(n) = "0010 s x"
2926
       where.
2927
           "0010" is the code word
2928
           "s" is the sign bit
2929
           "x" is the one bit value field
2930
       The codec equation is described as follows:
2931
           sign = Xenco(n) \& 0x2
2932
           value = 16 * (Xenco(n) & 0x1) + 42 + 7
2933
           if (sign > 0) then
2934
              Xdeco( n ) = Xpred( n ) - value
2935
              if (Xdeco(n) < 0) then
2936
                 Xdeco(n) = 0
2937
              endif
2938
          else
2939
              Xdeco( n ) = Xpred( n ) + value
2940
              if (Xdeco(n) > 4095) then
2941
                 Xdeco(n) = 4095
              endif
2942
2943
           endif
       E.3.6.5
                 DPCM6 for 12-6-12 Decoder
2944
       Xenco(n) has the following format:
2945
           Xenco(n) = "011 s xx"
2946
       where,
2947
           "011" is the code word
2948
           "s" is the sign bit
2949
           "xx" is the two bit value field
2950
       The codec equation is described as follows:
2951
           sign = Xenco(n) \& 0x4
2952
           value = 32 * (Xenco(n) & 0x3) + 74 + 15
2953
           if (sign > 0) then
2954
              Xdeco( n ) = Xpred( n ) - value
2955
              if (Xdeco(n) < 0) then
2956
                 Xdeco(n) = 0
2957
              endif
2958
           else
2959
              Xdeco( n ) = Xpred( n ) + value
2960
              if (Xdeco(n) > 4095) then
2961
                 Xdeco(n) = 4095
2962
              endif
2963
           endif
```

E.3.6.6 DPCM7 for 12-6-12 Decoder

```
2964
       Xenco(n) has the following format:
2965
           Xenco(n) = "0011 s x"
2966
       where.
2967
           "0011" is the code word
2968
           "s" is the sign bit
2969
           "x" is the one bit value field
2970
       The codec equation is described as follows:
2971
           sign = Xenco(n) \& 0x2
2972
           value = 64 * (Xenco(n) & 0x1) + 202 + 31
2973
           if (sign > 0) then
2974
              Xdeco( n ) = Xpred( n ) - value
2975
              if (Xdeco(n) < 0) then
2976
                  Xdeco(n) = 0
2977
              endif
2978
           else
2979
              Xdeco( n ) = Xpred( n ) + value
2980
              if (Xdeco(n) > 4095) then
2981
                  Xdeco(n) = 4095
2982
              endif
2983
           endif
       E.3.6.7
                  PCM for 12-6-12 Decoder
2984
       Xenco(n) has the following format:
2985
           Xenco( n ) = "1 xxxxx"
2986
       where.
2987
           "1" is the code word
2988
           the sign bit is not used
2989
           "xxxxx" is the five bit value field
2990
       The codec equation is described as follows:
2991
           value = 128 * (Xenco(n) & 0x1f)
2992
           if (value > Xpred( n )) then
2993
              Xdeco(n) = value + 63
2994
```

Xdeco(n) = value + 64

2995

2996

endif

3016

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 156 illustrates interleaving JPEG image data with YUV422 image data using Data Type values.

Figure 157 illustrates interleaving JPEG image data with YUV422 image data using both Data Type values and Virtual Channel Identifiers.

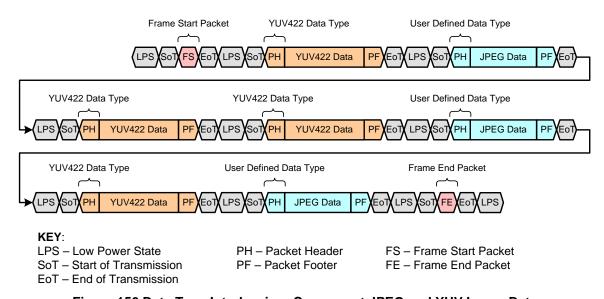


Figure 156 Data Type Interleaving: Concurrent JPEG and YUV Image Data

Specification for CSI-2 Version 1.3

29-May-2014

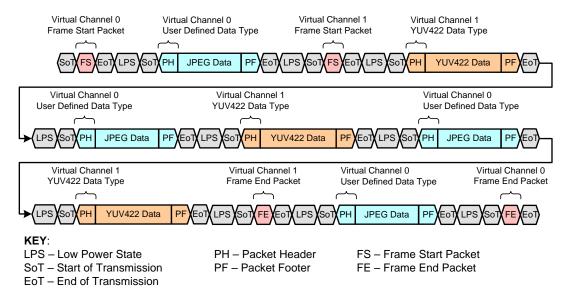


Figure 157 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data

3018 Both Figure 156 and Figure 157 can be similarly extended to the interleaving of JPEG image data with any 3019 other type of image data, e.g. RGB565.

Figure 158 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
- 3023 • Streaming YUV22 with occasional JPEG for still capture
- 3024 Again, these examples could also represent interleaving JPEG data with any other image data type.

3017

3020

3021

3022

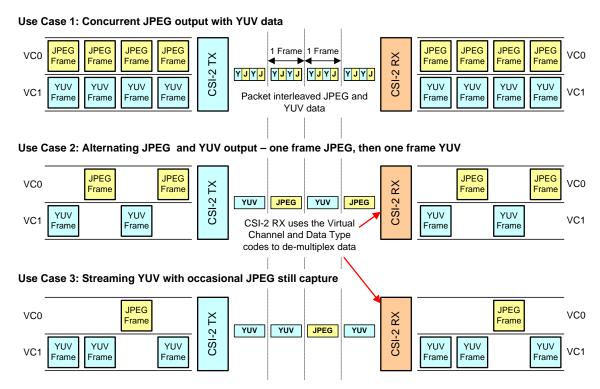


Figure 158 Example JPEG and YUV Interleaving Use Cases