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

<u>Draft</u> Version 1.01.00 Revision 0.04 – 2 April 2009

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

1 **NOTICE OF DISCLAIMER**

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

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

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

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

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

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

41

Contents

182 **Figures**

329

330

Tables

361

362

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

1   Overview

1.1   Scope

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

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

1.2   Purpose

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

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

2   Terminology

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

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

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

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

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

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

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

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

2.1   Definitions

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

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

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

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

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

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

420   **2.2   Abbreviations**

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

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

423   **2.3   Acronyms**

424   BER   Bit Error Rate

425   CIL   Control and Interface Logic

426   CRC   Cyclic Redundancy Check

427   CSI   Camera Serial Interface

428   CSPS   Chroma Sample Pixel Shifted

429   DDR   Dual Data Rate

430   DI   Data Identifier

431   DT   Data Type

432   ECC   Error Correction Code

433   EoT   End of Transmission

434   EXIF   Exchangeable Image File Format

435   FE   Frame End

436   FS   Frame Start

437   HS   High Speed; identifier for operation mode

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

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

440   I2C   Inter-Integrated Circuit

441   JFIF   JPEG File Interchange Format

442   JPEG   Joint Photographic Expert Group

443   LE   Line End

444   LLP   Low Level Protocol

445   LS   Line Start

446   LSB   Least Significant Bit

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

469 **3　References**

470　　[PHIL01]　　　　*THE I 2C-BUS SPECIFICATION*, version 2.1, Philips Semiconductors, January 2000

471　　[MIPI01]　　　　*MIPI Alliance Specification for D-PHY*, version 0.90.00, MIPI Alliance, 8 October 2007
472

473   ## 4   Overview of CSI-2

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

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

480
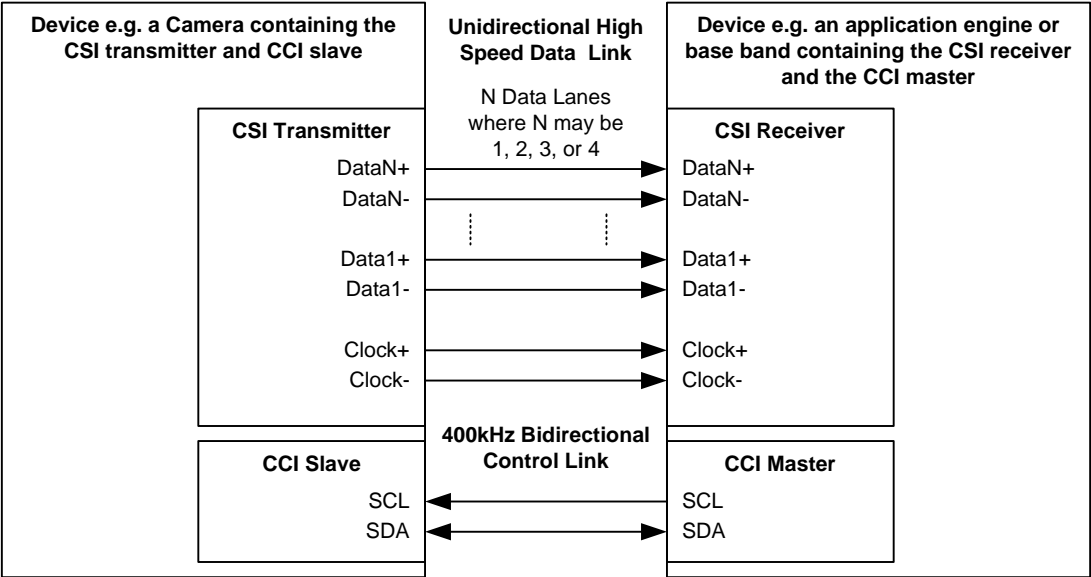
481   **Figure 1 CSI-2 and CCI Transmitter and Receiver Interface**

482

483 # 5   CSI-2 Layer Definitions

Transmitter   **Receiver**

Application	Data Formats	Byte to Pixel Unpacking Formats

Pixel   Control

6-, 7-, 8-, 10-, 12-, 14-, 15-, 16-, 18- or 24-bits

Pixel to Byte Packing Formats — Data Formats

Low Level Protocol — Packet Based Protocol / Arbitrary Data Support

Lane Management Layer — Lane Distribution/Lane Merging

PHY Layer — Generation / Detection of packet start and stop signaling / Serializer / Deserializer / Clock Generation / Recovery (DDR) / Electrical Layer

High Speed Unidirectional Clock

Lane 1 - High Speed Unidirectional Data

484   Lane N - High Speed Unidirectional Data

485   **Figure 2 CSI-2 Layer Definitions**

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

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

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

496 The PHY layer is described in [MIPI01].

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

501   • **Pixel/Byte Packing/Unpacking Layer.** The CSI-2 supports image applications with varying
502   pixel formats from six to twenty-four bits per pixels. In the transmitter this layer packs pixels
503   from the Application layer into bytes before sending the data to the Low Level Protocol layer.
504   In the receiver this layer unpacks bytes from the Low Level Protocol layer into pixels before
505   sending the data to the Application layer. Eight bits per pixel data is transferred unchanged by
506   this layer.

507   • **Low Level Protocol.** The Low Level Protocol (LLP) includes the means of establishing bit-
508   level and byte-level synchronization for serial data transferred between SoT (Start of
509   Transmission) and EoT (End of Transmission) events and for passing data to the next layer.
510   The minimum data granularity of the LLP is one byte. The LLP also includes assignment of
511   bit-value interpretation within the byte, i.e. the "Endian" assignment.

512   • **Lane Management.** CSI-2 is Lane-scalable for increased performance. The number of data
513   Lanes may be one, two, three or four depending on the bandwidth requirements of the
514   application. The transmitting side of the interface distributes ("distributor" function) the
515   outgoing data stream to one or more Lanes. On the receiving side, the interface collects bytes
516   from the Lanes and merges ("merger" function) them together into a recombined data stream
517   that restores the original stream sequence.

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

523 • **Application Layer.** This layer describes higher-level encoding and interpretation of data
524   contained in the data stream. The CSI-2 specification describes the mapping of pixel values to
525   bytes.

526 The normative sections of the specification only relate to the external part of the Link, e.g. the data and bit
527 patterns that are transferred across the Link. All internal interfaces and layers are purely informative.
528

6   Camera Control Interface (CCI)

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

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

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

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

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

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

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

6.1   Data Transfer Protocol

The data transfer protocol is according to I2C standard. The START, REPEATED START and STOP conditions as well as data transfer protocol are specified in *The I^{2}C Specification* [PHIL01].

6.1.1   Message Type

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

The slave address in the CCI is 7-bit.

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

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

| S | SLAVE ADDRESS | R/W | A | SUB ADDRESS | A | DATA | A/Ā | P |

INDEX[7:0]
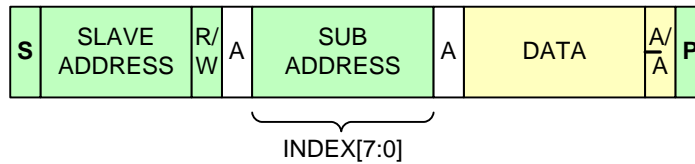
Message type with 16-bit index and 8-bit data (7-bit address)

| S | SLAVE ADDRESS | R/W | A | SUB ADDRESS | A | SUB ADDRESS | A | DATA | A/Ā | P |

INDEX[15:8]   INDEX[7:0]

	From slave to master	S = START condition	A = Acknowledge
	From master to slave	P = STOP condition	Ā = Negative acknowledge
	Direction dependent on operation		

563
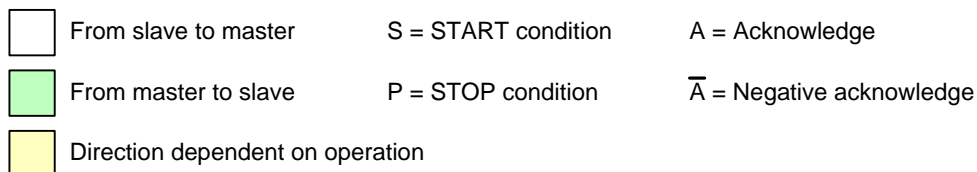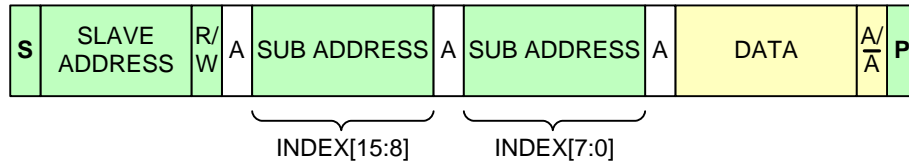
564   **Figure 3 CCI Message Types**

565   **6.1.2   Read/Write Operations**

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

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

572   **6.1.2.1   Single Read from Random Location**

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

| Previous Index value, K | Index M | Index M +1 |

| S | SLAVE ADDRESS | 0 | A | SUB ADDRESS | A | Sr | SLAVE ADDRESS | 1 | A | DATA | Ā | P |

INDEX, value M

| | From slave to master | S = START condition | A = Acknowledge | Sr = REPEATED START condition |
| | From master to slave | P = STOP condition | Ā = Negative acknowledge | |

577
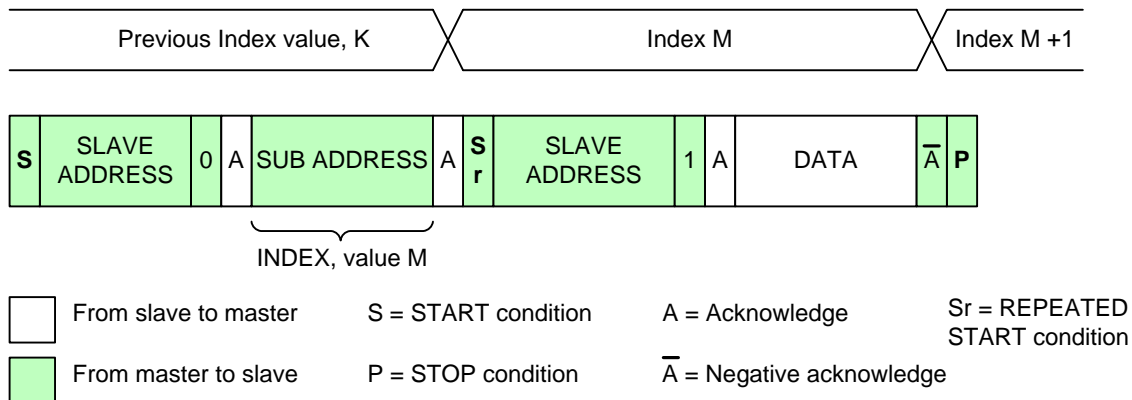
578   **Figure 4 CCI Single Read from Random Location**

579 ### 6.1.2.2   Single Read from the Current Location

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

583
584

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

586 ### 6.1.2.3   Sequential Read Starting from a Random Location

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

592
593
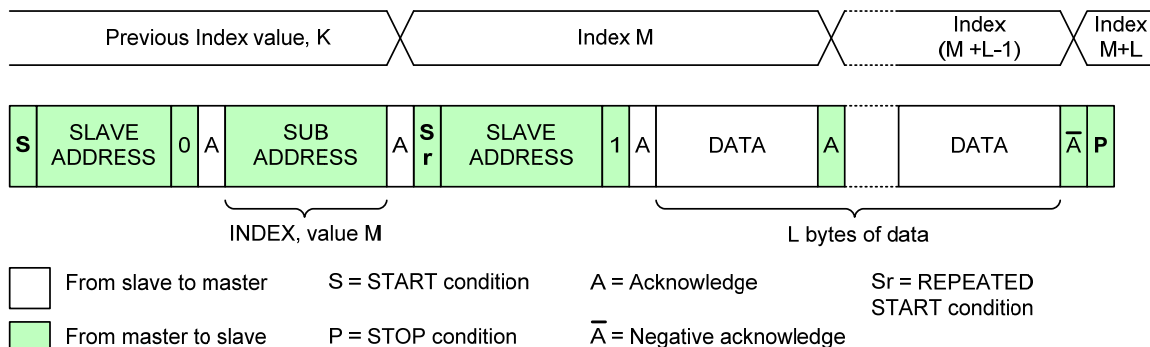
594   **Figure 6 CCI Sequential Read Starting from a Random Location**

595 ### 6.1.2.4   Sequential Read Starting from the Current Location

596 A sequential read starting from the current location is similar to a sequential read from a random location.
597 The only exception is there is no dummy write operation. The command sequence is illustrated in Figure 7.
598 The master terminates the read operation by issuing a negative acknowledge and stop condition.
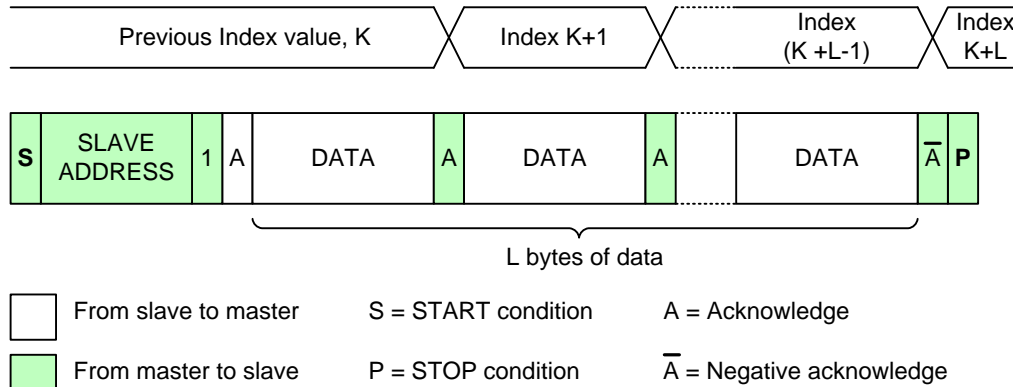
599
600   **Figure 7 CCI Sequential Read Starting from the Current Location**

601 **6.1.2.5   Single Write to a Random Location**

602 A write operation to a random location is illustrated in Figure 8. The master issues a write operation to the
603 slave then issues the index and data after the slave has acknowledged the write operation. The write
604 operation is terminated with a stop condition from the master.
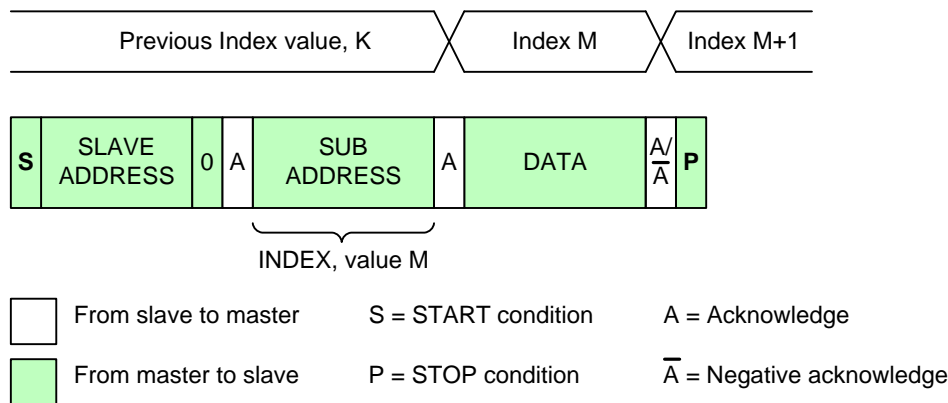
605
606   **Figure 8 CCI Single Write to a Random Location**

607 **6.1.2.6   Sequential Write**

608 The sequential write operation is illustrated in Figure 9. The slave auto-increments the index after each data
609 byte is received. The sequential write operation is terminated with a stop condition from the master.
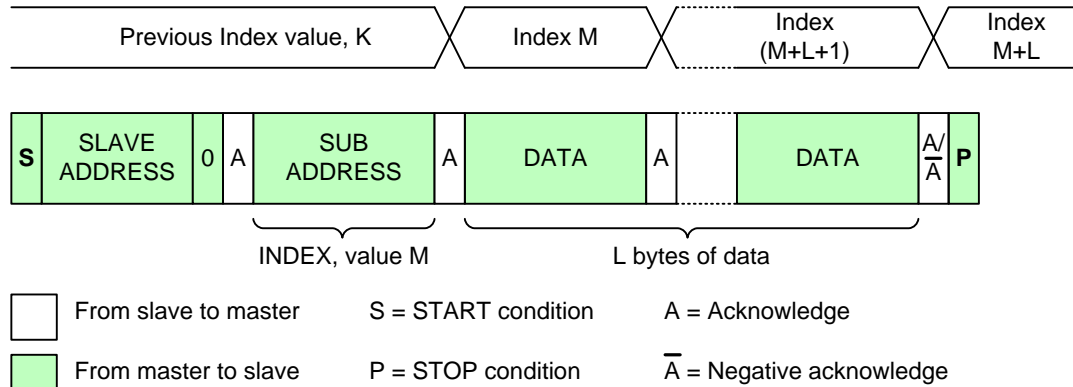
Figure 9 CCI Sequential Write Starting from a Random Location

6.2 CCI Slave Addresses

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

6.3 CCI Multi-Byte Registers

6.3.1 Overview

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

- 8-bit – generic setup registers

- 16-bit – parameters like line-length, frame-length and exposure values

- 32-bit – high precision setup values

- 64-bit – for needs of future sensors

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

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

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

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

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

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

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

647 • For a Write operation, the updating of the register shall be deferred to a time when the last bit of
648   the last byte has been received

649 • For a Read operation, the value read shall reflect the status of all bytes at the time that the first bit
650   of the first byte has been read

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

652 Without re-timing data may be corrupted as illustrated in Figure 10 and Figure 11 below.
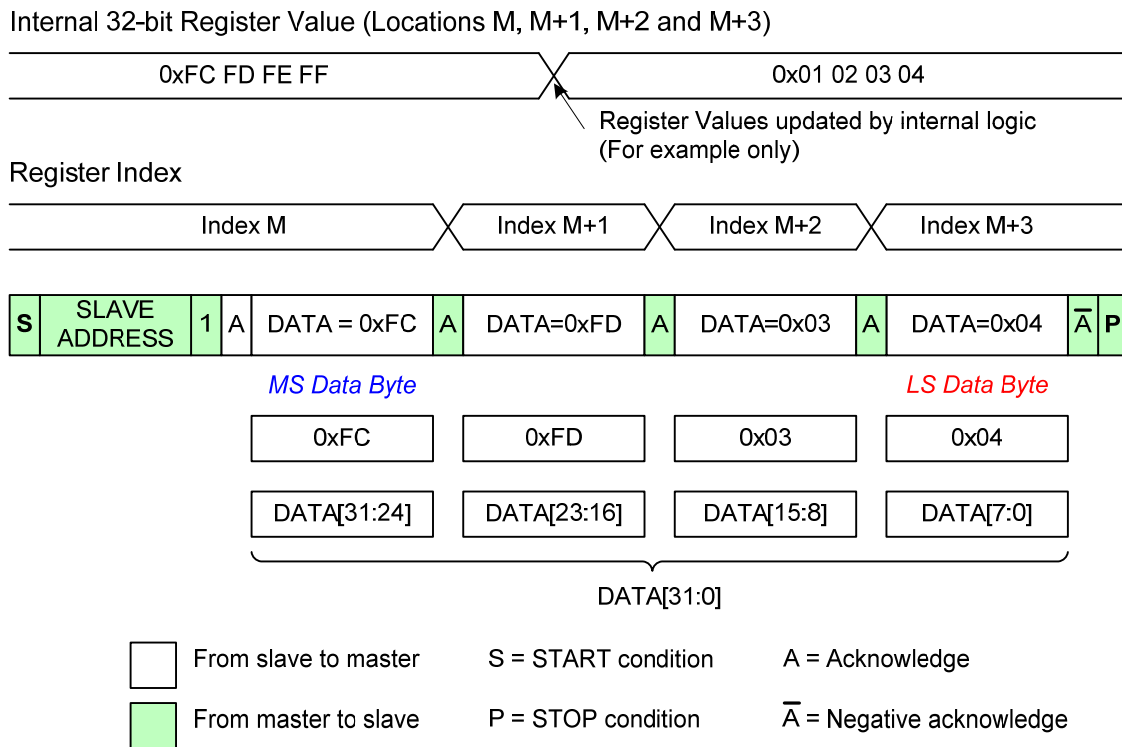
653
654 **Figure 10 Corruption of a 32-bit Wide Register during a Read Message**

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

| 0xFC FD FE FF | 0x01 FD FE FF | 0x01 02 FE FF | 0x01 02 03 FF | 0x01 02 03 04 |

Internal logic reads register value (For example only)

Register Index

| Index M | Index M+1 | Index M+2 | Index M+3 |

| S | SLAVE ADDRESS | 0 | A | // | DATA=0x01 | A | DATA=0x02 | A | DATA=0x03 | A | DATA=0x04 | A̅ | P |

MS Data Byte   *LS Data Byte*

| 0x01 | 0x02 | 0x03 | 0x04 |

| DATA[31:24] | DATA[23:16] | DATA[15:8] | DATA[7:0] |

DATA[31:0]

| ☐ | From slave to master | S = START condition | A = Acknowledge |
| ☐ | From master to slave | P = STOP condition | A̅ = Negative acknowledge |

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

6.3.2 The Transmission Byte Order for Multi-byte Register Values

This is a normative section.

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

DATA[15:0]

| S | SLAVE ADDRESS | 0 | A | SUB ADDRESS | A | DATA[15:8] | A | DATA[7:0] | A̅ | P |

Index Value, M   *MS Data Byte*   *LS Data Byte*

Index M   *Index M+1*
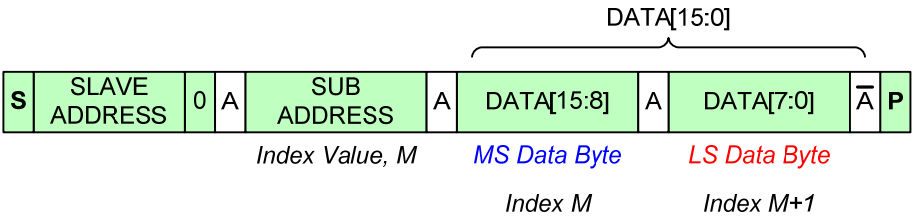
Figure 12 Example 16-bit Register Write

Register Index

664
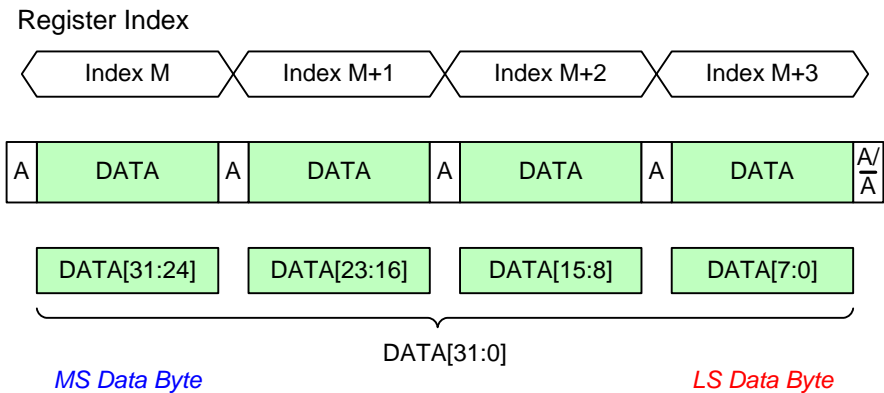
665   **Figure 13 Example 32-bit Register Write (address not shown)**

666
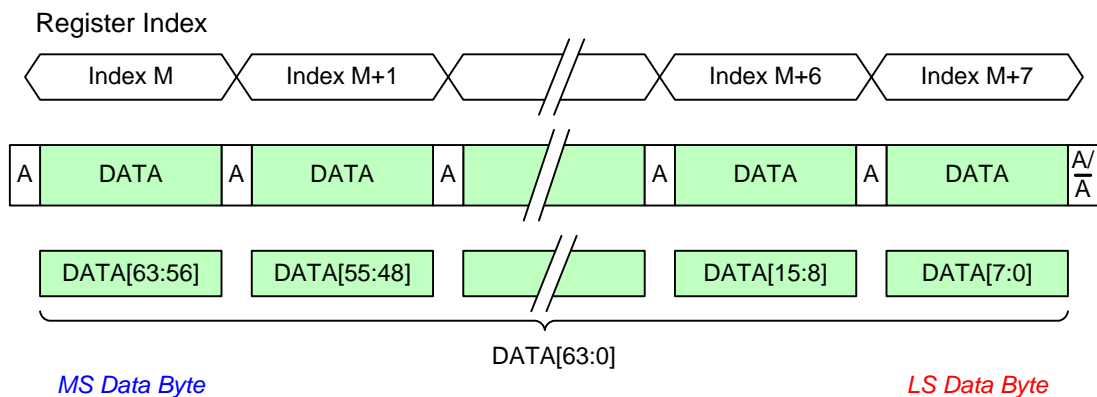
667   **Figure 14 Example 64-bit Register Write (address not shown)**

668   **6.3.3   Multi-Byte Register Protocol**

669   This is an informative section.

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

672   **6.3.3.1   Reading Multi-byte Registers**

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

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

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

| 0xFC FD | 0x01 02 |

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

| 0xFE FF | 0x03 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 | 0xFC FD | 0x03 04 |

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

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

| S | SLAVE ADDRESS | 1 | A | DATA = 0xFC | A | DATA=0xFD | A | DATA=0x03 | A | DATA=0x04 | $\overline{A}$ | P |

MS Data Byte	LS Data Byte	MS Data Byte	LS Data Byte
0xFC	0xFD	0x03	0x04
DATA[15:8]	DATA[7:0]	DATA[15:8]	DATA[7:0]

DATA[15:0]   DATA[15:0]

From slave to master   S = START condition   A = Acknowledge

From master to slave   P = STOP condition $\overline{A}$ = Negative acknowledge

Figure 15 Example 16-bit Register Read

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

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

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

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

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

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 causes the whole register value to be transferred into a temporary buffer

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

S	SLAVE ADDRESS	1	A	DATA = 0xFC	A	DATA=0xFD	A	DATA=0xFE	A	DATA=0xFF	A̅	P

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

🟩 From master to slave   P = STOP condition   A̅ = Negative acknowledge

Figure 16 Example 32-bit Register Read

6.3.3.2   Writing Multi-byte Registers

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

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

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

| 0xFC FD | 0x01 02 |

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

| 0xFE FF | 0x03 04 |

Register Index

| Index M | Index M+1 | Index M+2 | Index M+3 |

Temporary Buffer

| 0x00 00 | 0x01 00 | 0x00 00 | 0x03 00 | 0x00 00 |

A write to the LS byte of the register causes the contents of the temporary buffer to be transferred onto the register location

| S | SLAVE ADDRESS | 0 | A | | DATA=0x01 | A | DATA=0x02 | A | DATA=0x03 | A | DATA=0x04 | A̅ | P |

MS Data Byte	*LS Data Byte*	*MS Data Byte*	*LS Data Byte*
0x01	0x02	0x03	0x04
DATA[15:8]	DATA[7:0]	DATA[15:8]	DATA[7:0]

DATA[15:0]   DATA[15:0]

From slave to master   S = START condition   A = Acknowledge

From master to slave   P = STOP condition   A̅ = Negative acknowledge

700
701

702

Figure 17 Example 16-bit Register Write

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

703
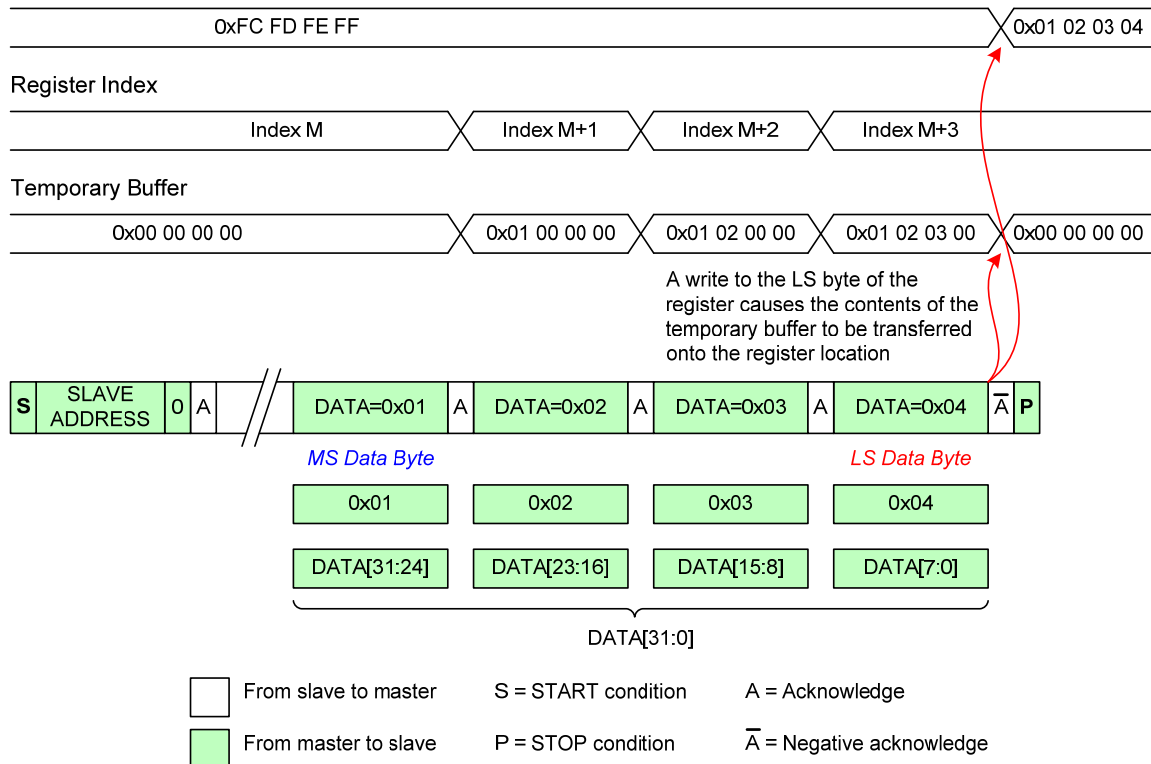
704 **Figure 18 Example 32-bit Register Write**

705 ## 6.4　Electrical Specifications and Timing for I/O Stages

706 The electrical specification and timing for I/O stages conform to I^2C Standard- and Fast-mode devices.
707 Information presented in Table 1 is from [PHIL01].

708 **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_{HYS}	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 $V_{DD} > 2V$ $V_{DD} < 2V$	V_{OL1} V_{OL3}	0 N/A	0.4 N/A	0 0	0.4 $0.2V_{DD}$	V

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

709   Notes:

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

711   2. C_B = capacitance of one bus line in pF

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

713   **Table 2 CCI Timing Specification**

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

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

714   Notes:

715   1. All values referred to $V_{IHmin} = 0.7V_{DD}$ and $V_{ILmax} = 0.3V_{DD}$

716   2. A device shall internally provide a hold time of at least 300 ns for the SDA signal (referred to the V_{IHmin} of
717   the SCL signal) to bridge the undefined region of the falling edge of SCL

718   3. The maximum $t_{HD;DAT}$ has only to be met if the device does not the LOW period (t_{LOW}) of the SCL signal

719   4. A Fast-mode I2C-bus device can be used in a Standard-mode I2C-bus system, but the requirement $t_{SU;DAT} \geq$
720   250 ns shall be then met. This will be automatically the case if the device does not stretch the LOW period of
721   the SCL signal. If such device does stretch the low period of SCL signal, it shall output the next data bit to
722   the SDA line $t_{rMAX} + t_{SU;DAT} = 1000 + 250 = 1250$ ns (according to the Standard-mode I2C bus specification)
723   before the SCL line is released.

724   5. CB = total capacitance of one bus line in pF.

725   The CCI timing is illustrated in Figure 19.
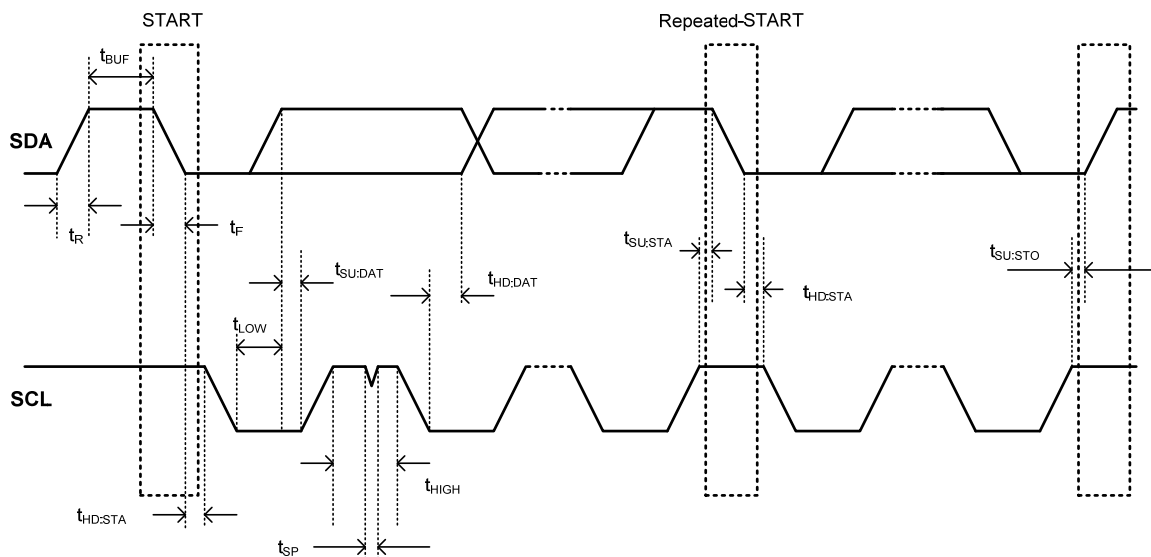
Figure 19 CCI Timing

7   Physical Layer

729

730   The CSI-2 uses the [MIPI01] physical layer.

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

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

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

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

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

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

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

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

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

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

745

8   Multi-Lane Distribution and Merging

746

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

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

757
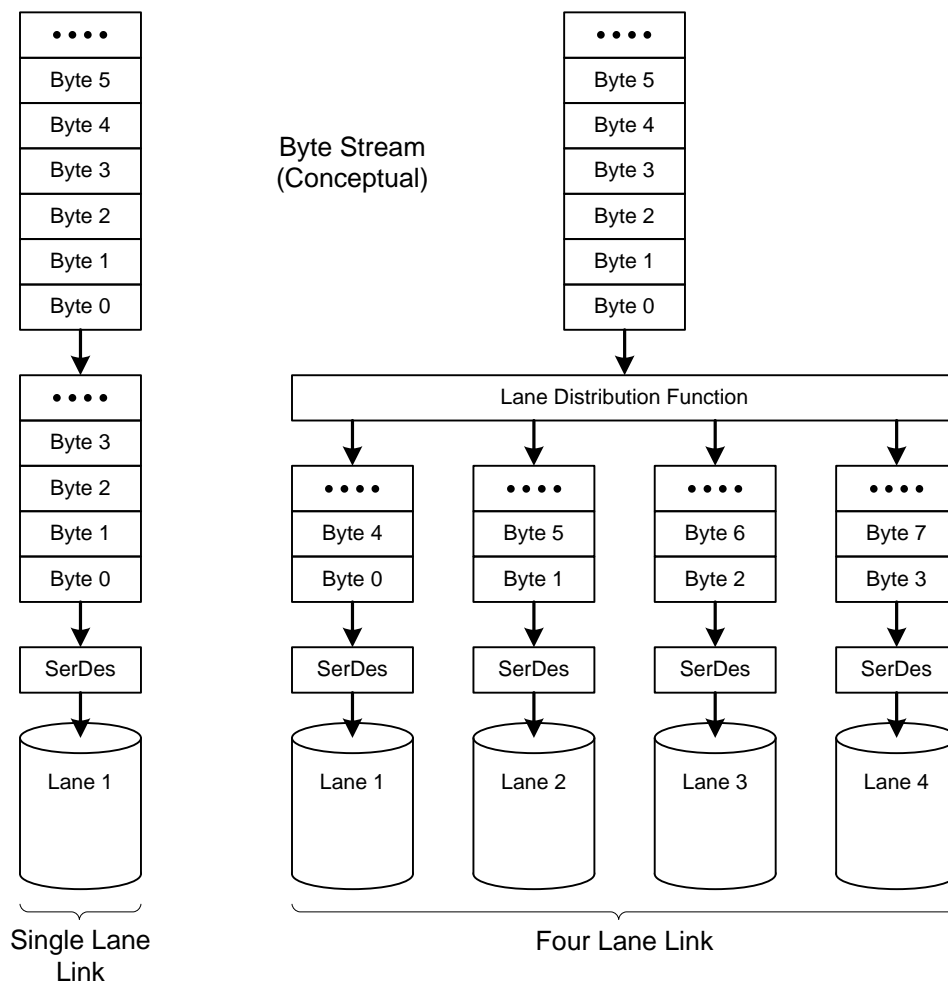
758   **Figure 20 Conceptual Overview of the Lane Distributor Function**

759
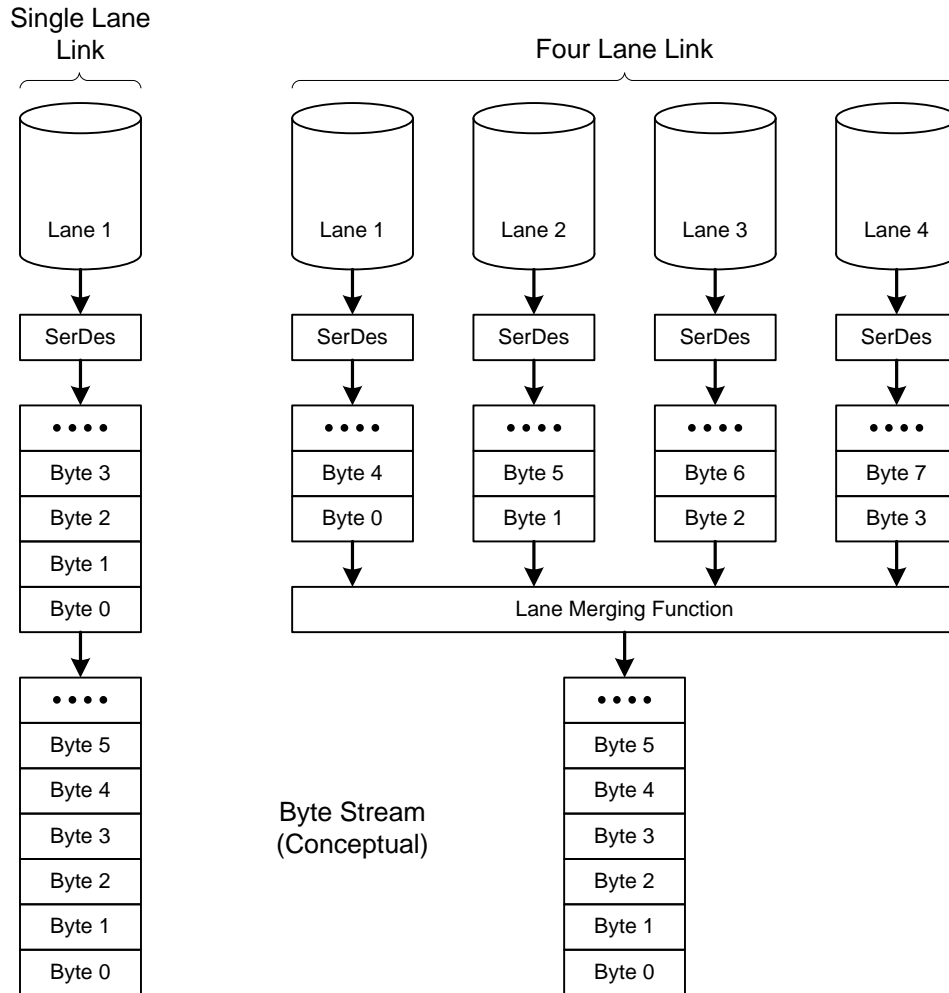
760   **Figure 21 Conceptual Overview of the Lane Merging Function**

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

766   Examples:

767   • 2-Lane system (Figure 22): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to
768   Lane 1, byte 3 goes to Lane 2, byte 4 goes to Lane 1 and so on.

769   • 3-Lane system (Figure 23): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to
770   Lane 3, byte 3 goes to Lane 1, byte 4 goes to Lane 2 and so on.

771   • 4-Lane system (Figure 24):byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to
772   Lane 3, byte 3 goes to Lane 4, byte 4 goes to Lane 1 and so on

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

777   Each D-PHY data Lane operates autonomously.

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

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

Number of Bytes, N, transmitted is an integer multiple of the number of lanes:
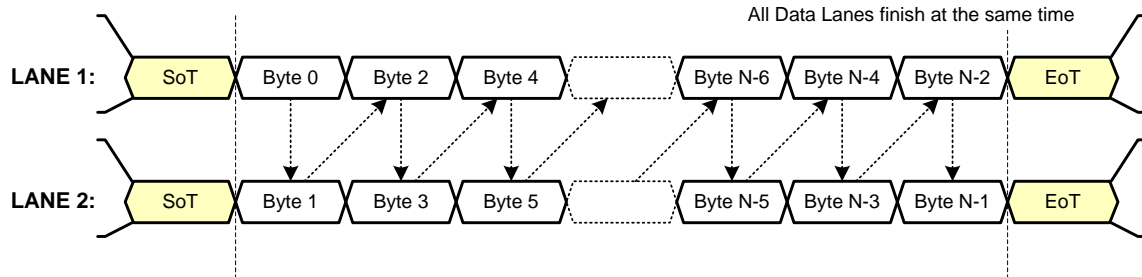
All Data Lanes finish at the same time

LANE 1: | SoT | Byte 0 | Byte 2 | Byte 4 | | Byte N-6 | Byte N-4 | Byte N-2 | EoT |

LANE 2: | SoT | Byte 1 | Byte 3 | Byte 5 | | Byte N-5 | Byte N-3 | Byte N-1 | EoT |

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

Data Lane 2 finishes 1 byte earlier than Data Lane 1

LANE 1: | SoT | Byte 0 | Byte 2 | Byte 4 | | Byte N-5 | Byte N-3 | Byte N-1 | EoT |

LANE 2: | SoT | Byte 1 | Byte 3 | Byte 5 | | Byte N-4 | Byte N-2 | EoT | LPS |

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

783
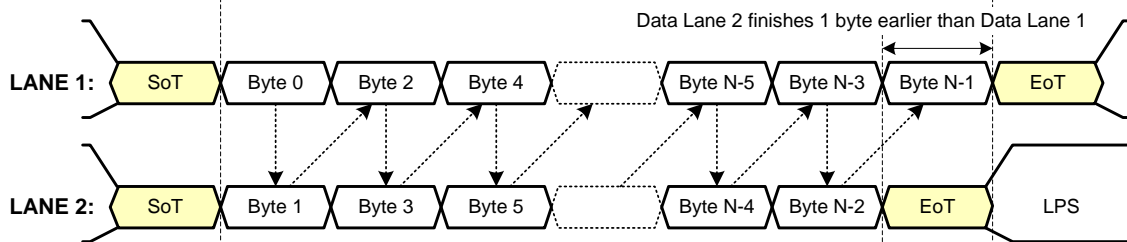
784   **Figure 22 Two Lane Multi-Lane Example**

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

All Data Lanes finish at the same time

LANE 1:	SoT	Byte 0	Byte 3	Byte 6		Byte N-9	Byte N-6	Byte N-3	EoT
LANE 2:	SoT	Byte 1	Byte 4	Byte 7		Byte N-8	Byte N-5	Byte N-2	EoT
LANE 3:	SoT	Byte 2	Byte 5	Byte 8		Byte N-7	Byte N-4	Byte N-1	EoT

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

Data Lanes 2 & 3 finish 1 byte earlier than Data Lane 1

LANE 1:	SoT	Byte 0	Byte 3	Byte 6		Byte N-7	Byte N-4	Byte N-1	EoT
LANE 2:	SoT	Byte 1	Byte 4	Byte 7		Byte N-6	Byte N-3	EoT	LPS
LANE 3:	SoT	Byte 2	Byte 5	Byte 8		Byte N-5	Byte N-2	EoT	LPS

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

Data Lane 3 finishes 1 byte earlier than Data Lanes 1 & 2

LANE 1:	SoT	Byte 0	Byte 3	Byte 6		Byte N-8	Byte N-5	Byte N-2	EoT
LANE 2:	SoT	Byte 1	Byte 4	Byte 7		Byte N-7	Byte N-4	Byte N-1	EoT
LANE 3:	SoT	Byte 2	Byte 5	Byte 8		Byte N-6	Byte N-3	EoT	LPS

KEY:
LPS – Low Power State   SoT – Start of Transmission   EoT – End of Transmission
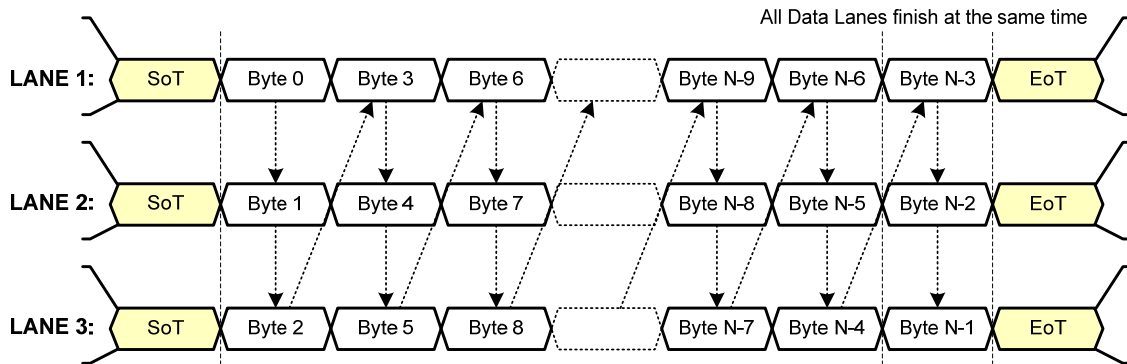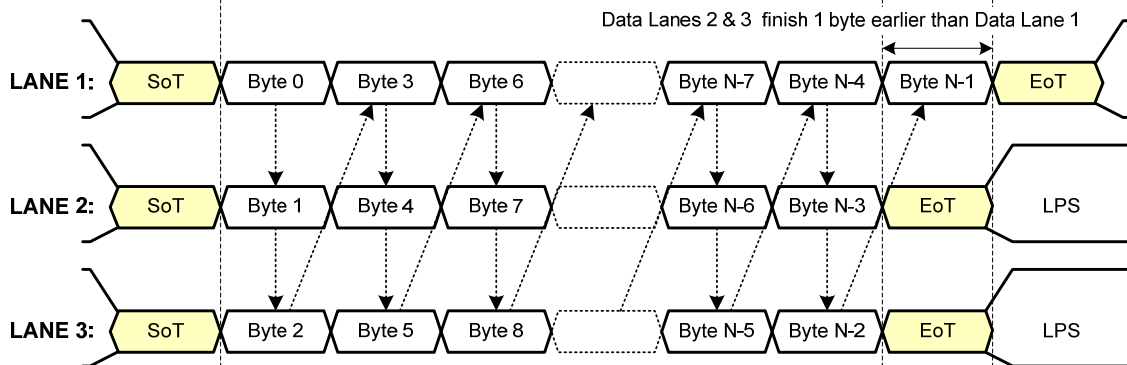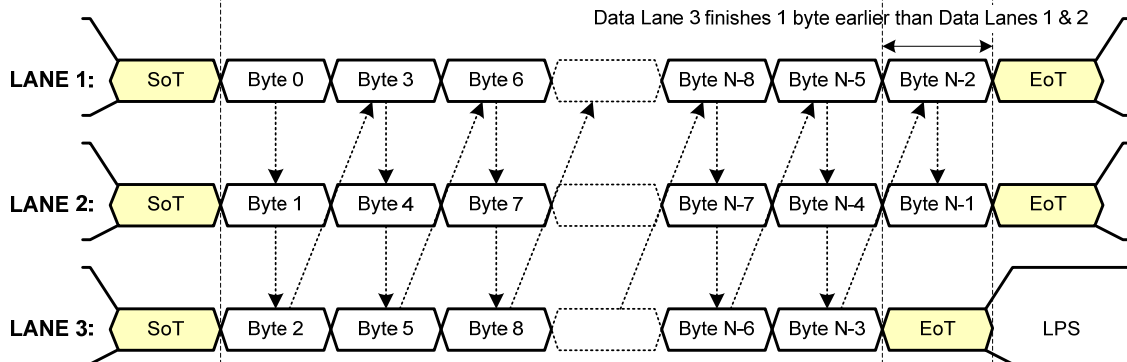
785
786   **Figure 23 Three Lane Multi-Lane Example**

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

All Data Lanes finish at the same time

LANE 1: SoT | Byte 0 | Byte 4 | Byte 8 | | Byte N-12 | Byte N-8 | Byte N-4 | EoT

LANE 2: SoT | Byte 1 | Byte 5 | Byte 9 | | Byte N-11 | Byte N-7 | Byte N-3 | EoT

LANE 3: SoT | Byte 2 | Byte 6 | Byte 10 | | Byte N-10 | Byte N-6 | Byte N-2 | EoT

LANE 4: SoT | Byte 3 | Byte 7 | Byte 11 | | Byte N-9 | Byte N-5 | Byte N-1 | EoT

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

Data Lanes 2, 3 & 4 finish 1 byte earlier than Data Lane 1

LANE 1: SoT | Byte 0 | Byte 4 | Byte 8 | | Byte N-9 | Byte N-5 | Byte N-1 | EoT

LANE 2: SoT | Byte 1 | Byte 5 | Byte 9 | | Byte N-8 | Byte N-4 | EoT | LPS

LANE 3: SoT | Byte 2 | Byte 6 | Byte 10 | | Byte N-7 | Byte N-3 | EoT | LPS

LANE 4: SoT | Byte 3 | Byte 7 | Byte 11 | | Byte N-6 | Byte N-2 | EoT | LPS
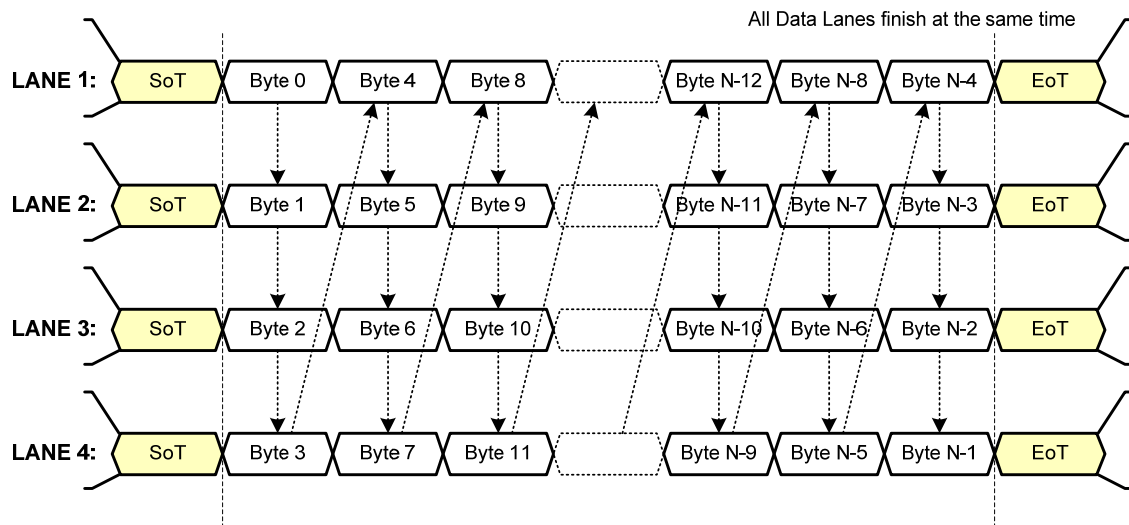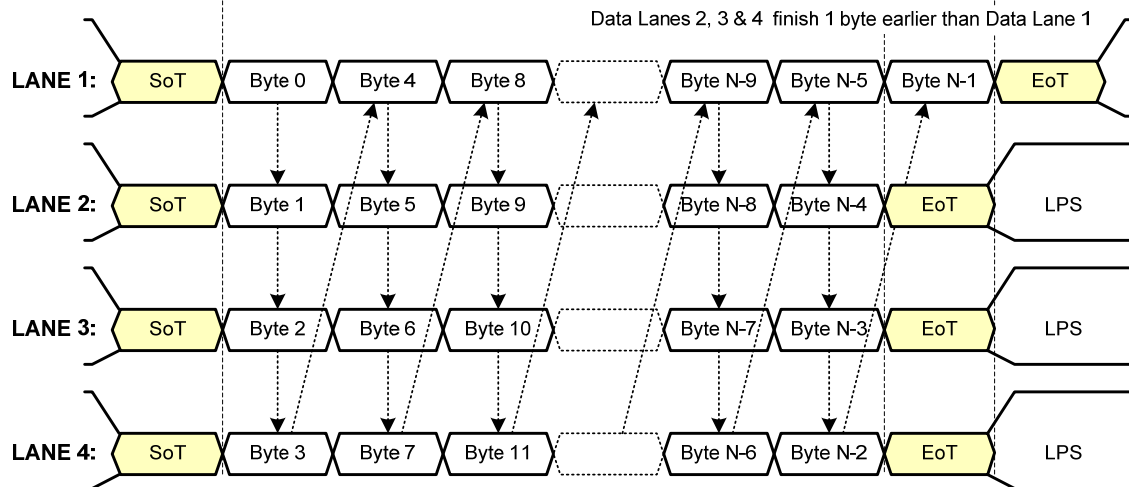
KEY:
LPS – Low Power State   SoT – Start of Transmission   EoT – End of Transmission

Figure 24 Four Lane Multi-Lane Example

8.1   Multi-Lane Interoperability

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

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

Two cases:

798   • If M<=N then there is no loss of performance – the receiver has sufficient data Lanes to match the
799   transmitter (Figure 25 and Figure 26).

800   • If M> N then there may be a loss of performance (e.g. frame rate) as the receiver has fewer data
801   Lanes than the transmitter (Figure 27 and Figure 28).

802
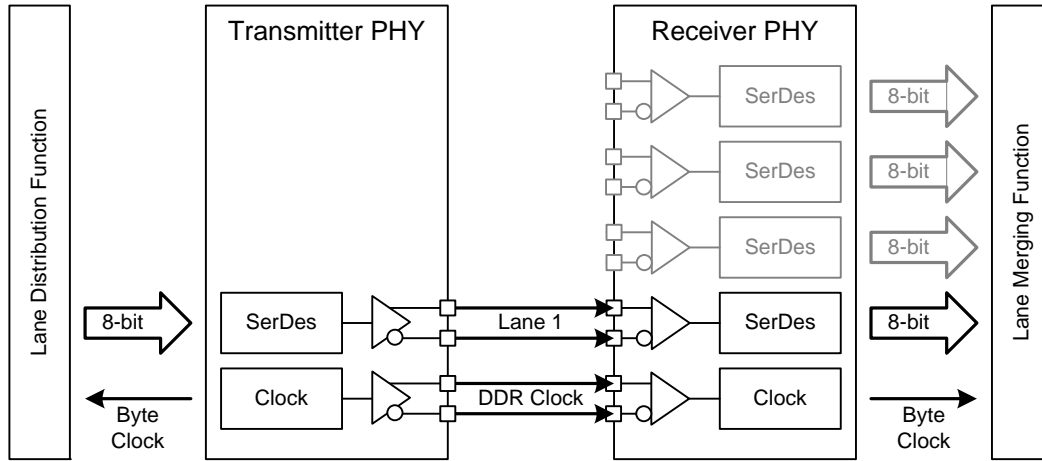
803   **Figure 25 One Lane Transmitter and Four Lane Receiver Example**

804
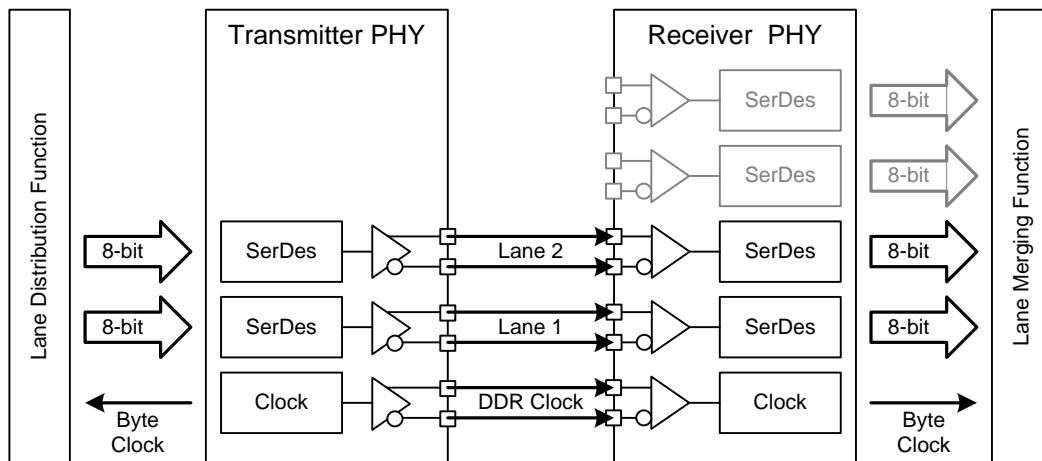
805   **Figure 26 Two Lane Transmitter and Four Lane Receiver Example**

806
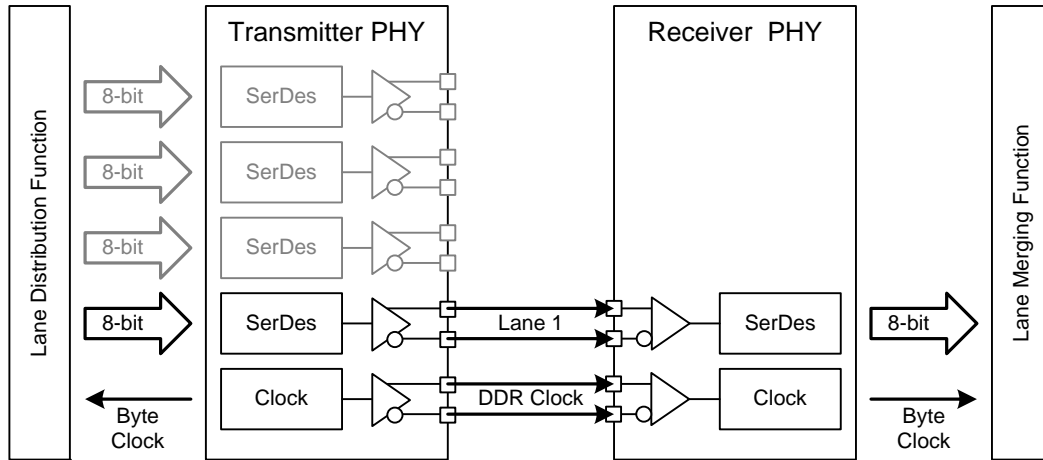
807   **Figure 27 Four Lane Transmitter and One Lane Receiver Example**

808
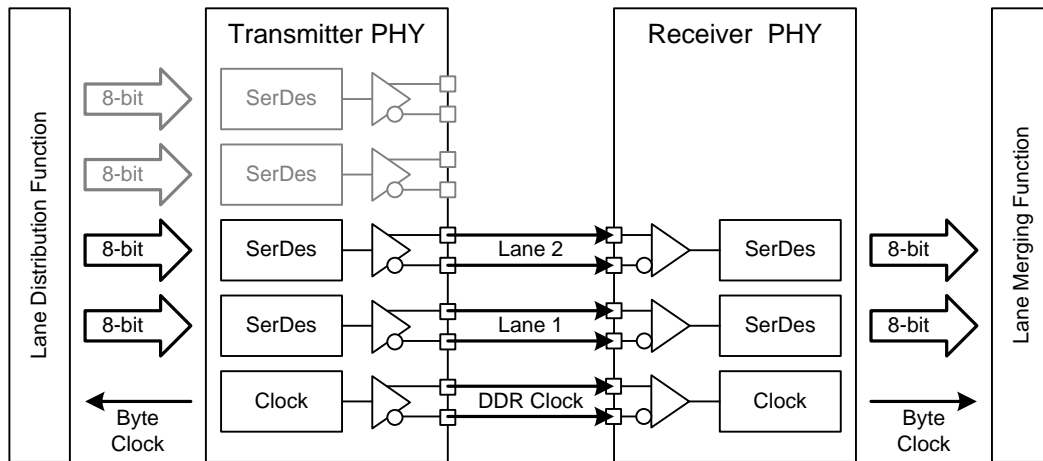
809   **Figure 28 Four Lane Transmitter and Two Lane Receiver Example**

810

811 ## 9   Low Level Protocol

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

815 Low Level Protocol Features:

816 •   Transport of arbitrary data (Payload independent)

817 •   8-bit word size

818 •   Support for up to four interleaved virtual channels on the same link

819 •   Special packets for frame start, frame end, line start and line end information

820 •   Descriptor for the type, pixel depth and format of the Application Specific Payload data

821 •   16-bit Checksum Code for error detection.

822
823
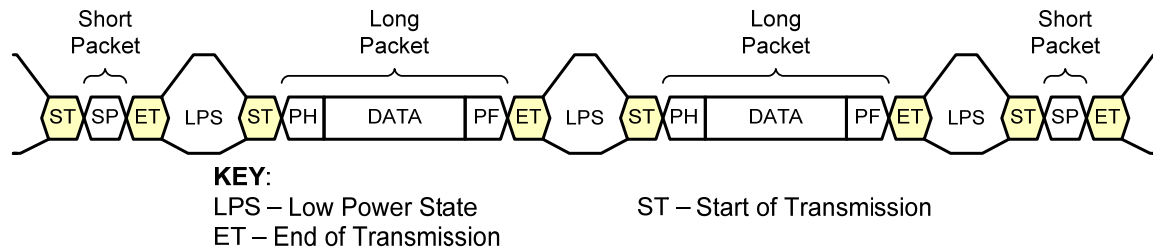
824   **Figure 29 Low Level Protocol Packet Overview**

825 ## 9.1   Low Level Protocol Packet Format

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

830 ### 9.1.1   Low Level Protocol Long Packet Format

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

DATA IDENTIFIER (DI):
Contains the Virtual Channel Identifier and the Data Type Information
Data Type denotes the format/content of the Application Specific Payload Data.
Used by the application specific layer.

16-bit WORD COUNT (WC):
The receiver reads the next WC data words independent of their values.
The receiver is NOT looking for any embedded sync sequences within the
payload data. The receiver uses the WC value to determine the end End of
the Packet

8-bit Error Correction Code (ECC) for the Packet Header:
8-bit ECC code for the Packet Header. Allows 1-bit errors with the
packet header to be corrected and 2-bit errors to be detected

APPLICATION SPECIFIC PAYLOAD   CHECKSUM (CS)

| LPS | SoT | Data ID | Word Count (WC) | ECC | Data 0 | Data 1 | Data 2 | Data 3 | ... | Data WC-4 | Data WC-3 | Data WC-2 | Data WC-1 | 16-bit Checksum | EoT | LPS |

32-bit PACKET HEADER (PH)

PACKET DATA:
Length = Word Count (WC) * Data Word
Width (8-bits). There are NO restrictions
on the values of the data words

16-bit PACKET FOOTER (PF)

Figure 30 Long Packet Structure

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

The Word Count defines the number of 8-bit data words in the Data Payload between the end of the Packet Header and the start of the Packet Footer. Neither the Packet Header nor the Packet Footer shall be included in the Word Count.

The Error Correction Code (ECC) byte allows single-bit errors to be corrected and 2-bit errors to be detected in the packet header. This includes both the data identifier value and the word count value.

After the end of the Packet Header the receiver reads the next Word Count * 8-bit data words of the Data Payload. While reading the Data Payload the receiver shall not look for any embedded sync codes. Therefore, there are no limitations on the value of a data word.

Once the receiver has read the Data Payload it reads the checksum in the Packet Footer. In the generic case, the length of the Data Payload shall be a multiple of 8-bit data words. In addition, each data format may impose additional restrictions on the length of the payload data, e.g. multiple of four bytes.

Each byte shall be transmitted least significant bit first. Payload data may be transmitted in any byte order restricted only by data format requirements. Multi-byte elements such as Word Count, Checksum and the Short packet 16-bit Data Field shall be transmitted least significant byte first.

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

856   ## 9.1.2   Low Level Protocol Short Packet Format

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

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

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

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

867
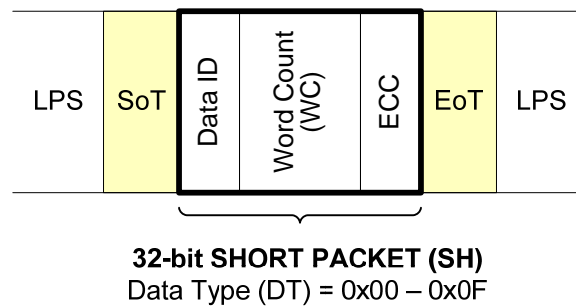
868   **Figure 31 Short Packet Structure**

869   ## 9.2   Data Identifier (DI)

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

873
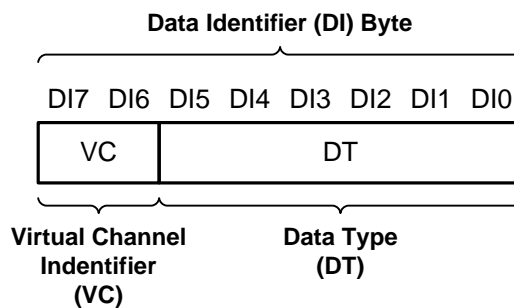
874   **Figure 32 Data Identifier Byte**

875   ## 9.3   Virtual Channel Identifier

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

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

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

883
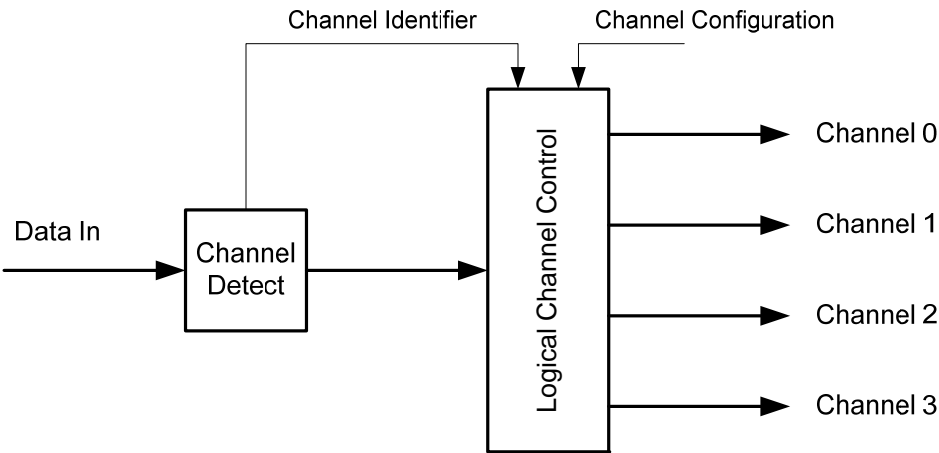
884   **Figure 33 Logical Channel Block Diagram (Receiver)**

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

KEY:
LPS – Low Power State   PH – Packet Header
SoT – Start of Transmission   PF – Packet Footer
EoT – End of Transmission
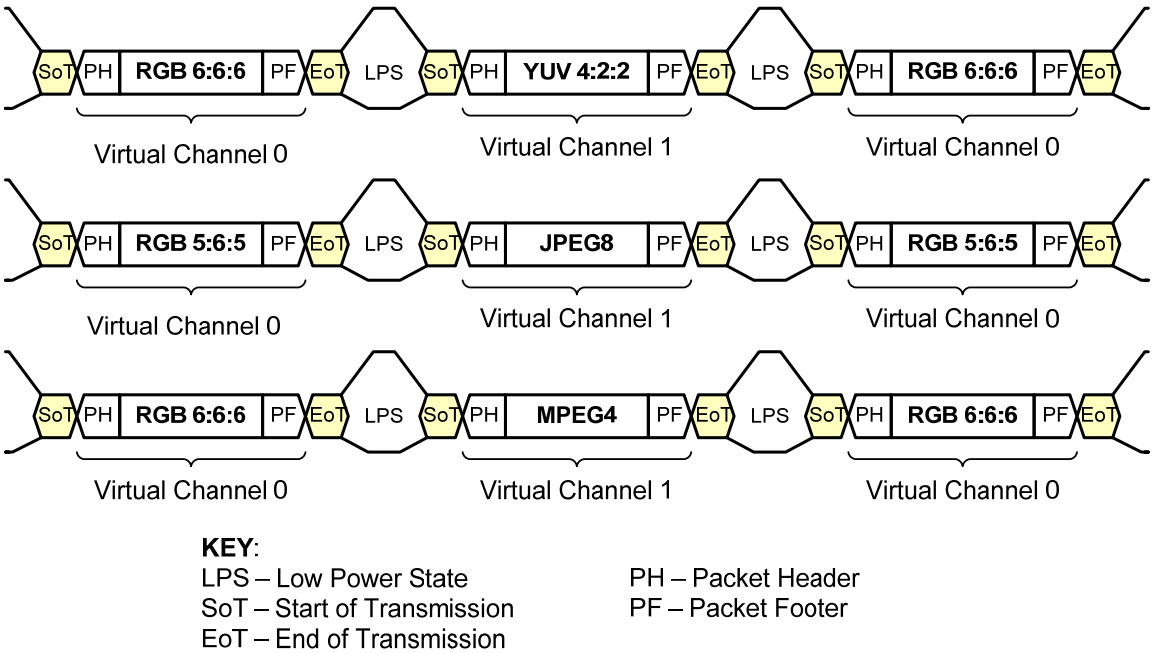
886
887   **Figure 34 Interleaved Video Data Streams Examples**

888 ## 9.4   Data Type (DT)

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

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

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

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

896   **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

897   ## 9.5   Packet Header Error Correction Code

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

903   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])
904   to D[15:8] and the Word Count MS Byte (WC[15:8]) to D[23:16]. This mapping is shown in Figure 35,
905   which also serves as an ECC calculation example.
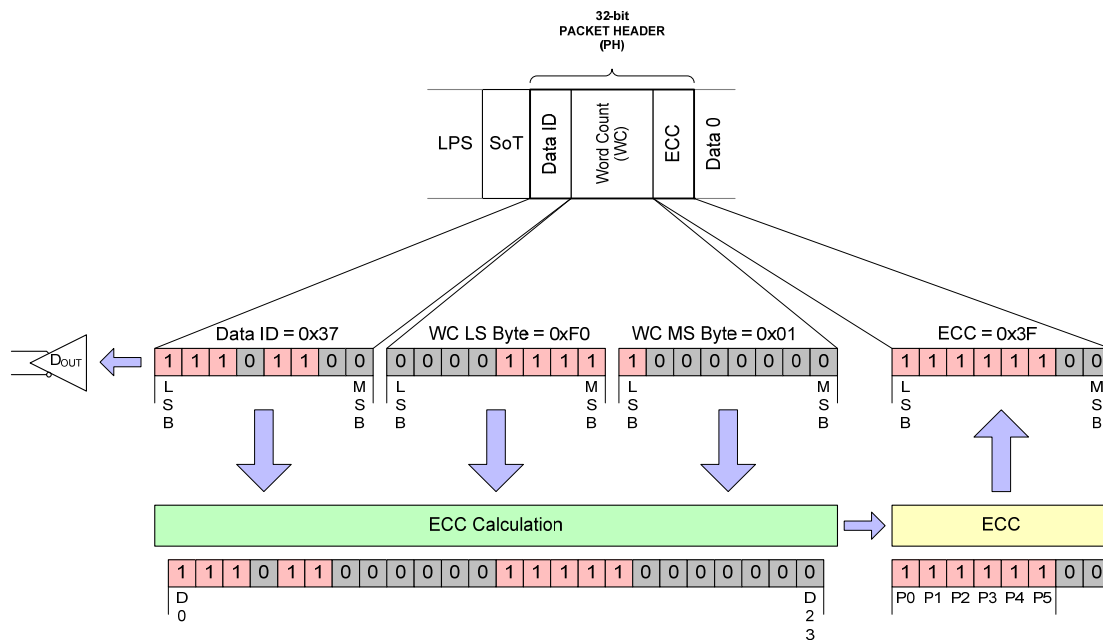
906
907   **Figure 35 24-bit ECC Generation Example**

9.5.1   General Hamming Code Applied to Packet Header

908

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

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

912   The result of appending the computed parity bits to the data bits is called the Hamming code word. The size
913   of the code word c is obviously $d + p$, and a Hamming code word is described by the ordered set (c,d). A
914   Hamming code word is generated by multiplying the data bits by a generator matrix **G**. This
915   multiplication's result is called the code word vector (c1, c2, c3,…cn), consisting of the original data bits
916   and the calculated parity bits. The generator matrix **G** used in constructing Hamming codes consists of **I**
917   (the identity matrix) and a parity generation matrix **A**:

918   **G** = [**I** | **A**]

919   The packet header plus the ECC code can be obtained as: PH = p***G** where p represents the header (24 or
920   64 bits) and **G** is the corresponding generator matrix.

921   Validating the received code word r, involves multiplying it by a parity check to form s, the syndrome or
922   parity check vector: s = **H***PH where PH is the received packet header and **H** is the parity check matrix:

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

924   If all elements of s are zero, the code word was received correctly. If s contains non-zero elements, then at
925   least one error is present. If a single bit error is encountered then the syndrome s is one of the elements of **H**
926   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
927   syndrome will be one of the elements on **I**, else it will be the data bit identified by the position of the
928   syndrome in **A**$^\mathbf{T}$.

9.5.2   Hamming-modified Code

929

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

935

936   **Table 4 ECC Syndrome Association Matrix**

d5d4d3	d2d1d0							
	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

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

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

939 e.g. 0x07=0b0000_0111=P7P6P5P4P3P2P1P0

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

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

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

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

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

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

950 $S = P_{SEND}\char`^P_{RECEIVED}$ where P_{SEND} is the 8/6-bit ECC field in the header and $P_{RECEIVED}$ is the
951 calculated parity of the received header.

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

955 **Table 5 ECC Parity Generation Rules**

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

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
11	0	0	1	0	0	1	0	1	0x25
12	0	0	1	0	0	1	1	0	0x26
13	0	0	1	0	1	0	0	1	0x29
14	0	0	1	0	1	0	1	0	0x2A
15	0	0	1	0	1	1	0	0	0x2C
16	0	0	1	1	0	0	0	1	0x31
17	0	0	1	1	0	0	1	0	0x32
18	0	0	1	1	0	1	0	0	0x34
19	0	0	1	1	1	0	0	0	0x38
20	0	0	0	1	1	1	1	1	0x1F
21	0	0	1	0	1	1	1	1	0x2F
22	0	0	1	1	0	1	1	1	0x37
23	0	0	1	1	1	0	1	1	0x3B
24	0	1	0	0	0	0	1	1	0x43
25	0	1	0	0	0	1	0	1	0x45
26	0	1	0	0	0	1	1	0	0x46
27	0	1	0	0	1	0	0	1	0x49
28	0	1	0	0	1	0	1	0	0x4A
29	0	1	0	0	1	1	0	0	0x4C
30	0	1	0	1	0	0	0	1	0x51
31	0	1	0	1	0	0	1	0	0x52
32	0	1	0	1	0	1	0	0	0x54
33	0	1	0	1	1	0	0	0	0x58
34	0	1	1	0	0	0	0	1	0x61
35	0	1	1	0	0	0	1	0	0x62
36	0	1	1	0	0	1	0	0	0x64
37	0	1	1	0	1	0	0	0	0x68
38	0	1	1	1	0	0	0	0	0x70
39	1	0	0	0	0	0	1	1	0x83
40	1	0	0	0	0	1	0	1	0x85
41	1	0	0	0	0	1	1	0	0x86
42	1	0	0	0	1	0	0	1	0x89
43	1	0	0	0	1	0	1	0	0x8A
44	0	0	1	1	1	1	0	1	0x3D

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
45	0	0	1	1	1	1	1	0	0x3E
46	0	1	0	0	1	1	1	1	0x4F
47	0	1	0	1	0	1	1	1	0x57
48	1	0	0	0	1	1	0	0	0x8C
49	1	0	0	1	0	0	0	1	0x91
50	1	0	0	1	0	0	1	0	0x92
51	1	0	0	1	0	1	0	0	0x94
52	1	0	0	1	1	0	0	0	0x98
53	1	0	1	0	0	0	0	1	0xA1
54	1	0	1	0	0	0	1	0	0xA2
55	1	0	1	0	0	1	0	0	0xA4
56	1	0	1	0	1	0	0	0	0xA8
57	1	0	1	1	0	0	0	0	0xB0
58	1	1	0	0	0	0	0	1	0xC1
59	1	1	0	0	0	0	1	0	0xC2
60	1	1	0	0	0	1	0	0	0xC4
61	1	1	0	0	1	0	0	0	0xC8
62	1	1	0	1	0	0	0	0	0xD0
63	1	1	1	0	0	0	0	0	0xE0

956   ### 9.5.3   ECC Generation on TX Side

957   This is an informative section.

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

959
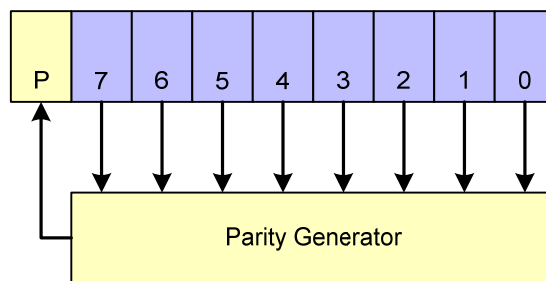
960   **Figure 36 64-bit ECC Generation on TX Side**

961   And Figure 37 for a 24-bit header:

962
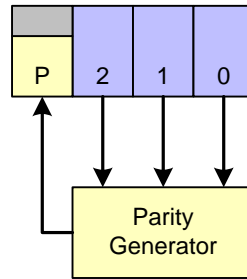
963   **Figure 37 24-bit ECC Generation on TX Side**

964   The parity generators are based on Table 5.

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

966   **9.5.4   Applying ECC on RX Side**

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

970
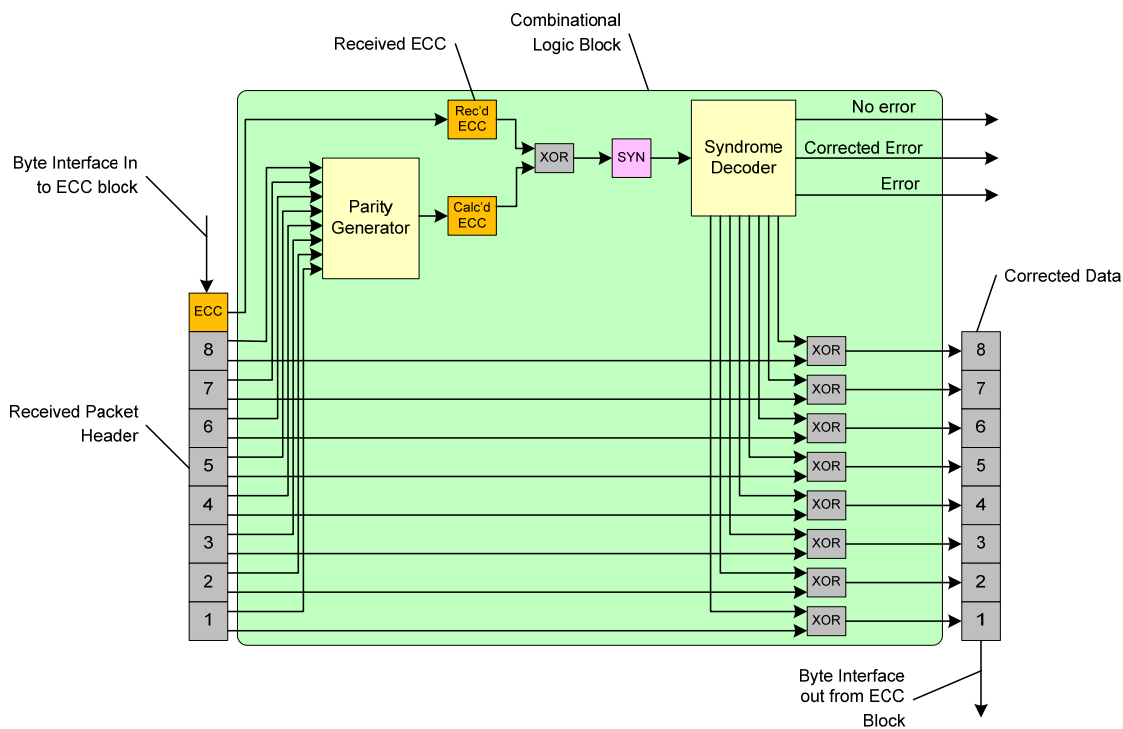
971   **Figure 38 64-bit ECC on RX Side Including Error Correction**

972   Decoding the syndrome has three aspects:

973   • Finding if the packet has any errors (if syndrome is 0, no errors are present)

974   • Checking if a single error has occurred by searching Table 5, if the syndrome is one of the entries
975   in the table, then a single bit error has occurred and the corresponding bit is affected, thus this
976   position in the data stream needs to be complemented. Also, if the syndrome is one of the rows of
977   the identity matrix I, then one of the parity bits are in error. If the syndrome cannot be identified,

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

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

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

983
984
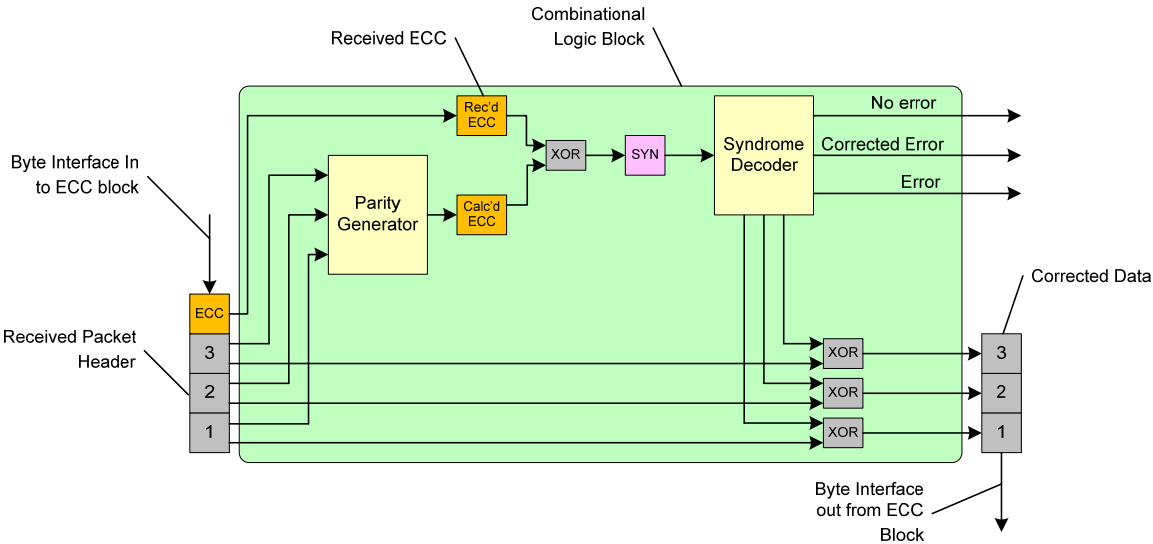
Figure 39 24-bit ECC on RX side Including Error Correction

986   ## 9.6   Checksum Generation

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

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

990
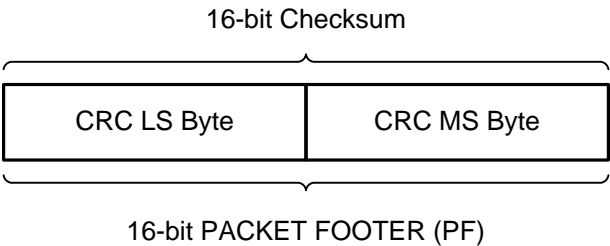
991   **Figure 40 Checksum Transmission**

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

Included in the Checksum　　　　　Checksum

DI	WC	ECC	Payload Data – Packet 1	CS
DI	WC	ECC	Payload Data – Packet 2	CS
DI	WC	ECC	Payload Data – Packet N	CS

32-bit PACKET
HEADER (PH)

16-bit PACKET
FOOTER (PF)

994
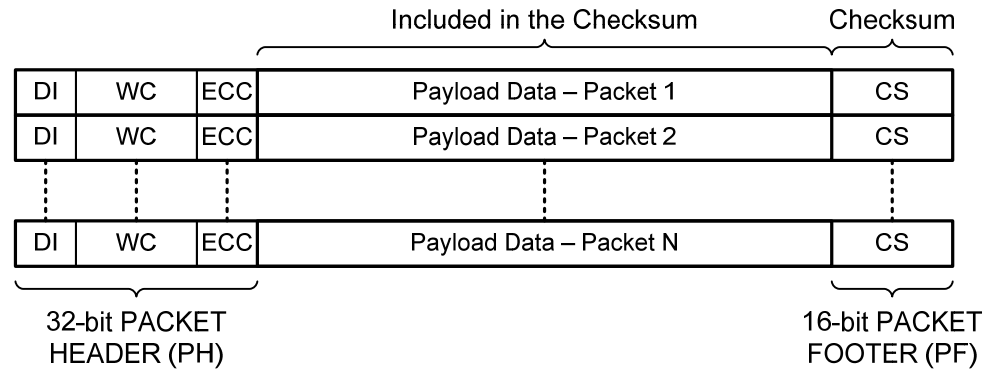
995　　　　　　　　**Figure 41 Checksum Generation for Packet Data**

996　The definition of a serial CRC implementation is presented in Figure 42. The CRC implementation shall be
997　functionally equivalent with the C code presented in Figure 43. The CRC shift register is initialized to
998　0xFFFF at the beginning of each packet. After all payload data has passed through the CRC circuitry, the
999　CRC circuitry contains the checksum. The 16-bit checksum produced by the C code in Figure 43 equals the
1000　final contents of the C[15:0] shift register shown in Figure 42. The checksum is then sent over CSI-2 bus to
1001　the receiver to verify that no errors have occurred in the transmission.

Polynomial: $x^{16} + x^{12} + x^5 + x^0$

Note: C15 represents x^0, C0 represents x^{15}

1002
1003
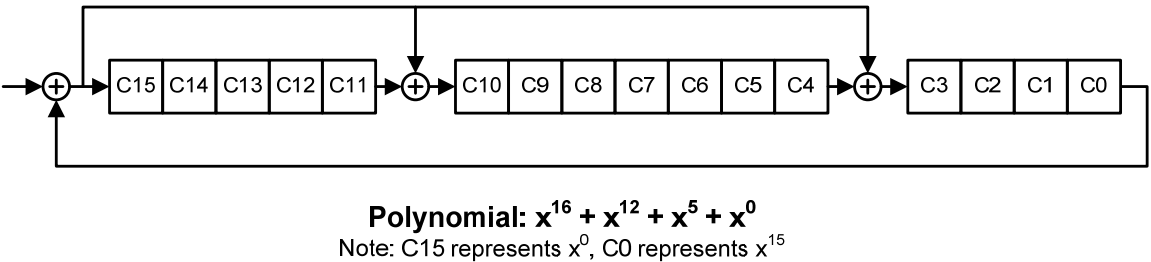
1004　　　　　　　　**Figure 42 Definition of 16-bit CRC Shift Register**

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

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

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

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

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

1005

1006   **Figure 43 16-bit CRC Software Implementation Example**

1007   The data and checksum are transmitted least significant byte first. Each bit within a byte is transmitted least
1008   significant bit first.
1009

1010   Data:
1011   FF 00 00 02 B9 DC F3 72 BB D4 B8 5A C8 75 C2 7C 81 F8 05 DF FF 00 00 01
1012   Checksum LS byte and MS byte:
1013   F0 00
1014
1015   Data:
1016   FF 00 00 00 1E F0 1E C7 4F 82 78 C5 82 E0 8C 70 D2 3C 78 E9 FF 00 00 01
1017   Checksum LS byte and MS byte:
1018   69 E5

1019   **9.7   Packet Spacing**

1020   Between Low Level Protocol packets there must always be a transition into and out of the Low Power State
1021   (LPS). Figure 44 illustrates the packet spacing with the LPS.

1022   The packet spacing does not have to be a multiple of 8-bit data words as the receiver will resynchronize to
1023   the correct byte boundary during the SoT sequence prior to the Packet Header of the next packet.

SHORT / LONG PACKET SPACING:
Variable - always a LPS between packets
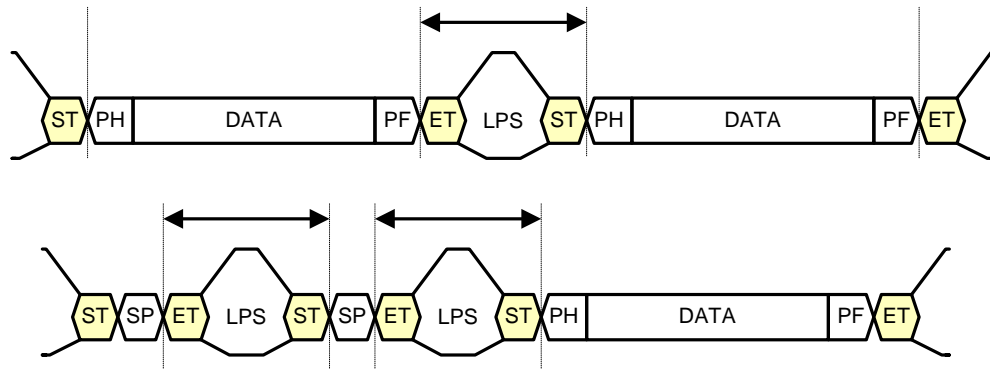
KEY:
LPS – Low Power State   PH – Packet Header
ST – Start of Transmission   PF – Packet Footer
ET – End of Transmission   SP – Short Packet

1024

1025   **Figure 44 Packet Spacing**

1026   **9.8   Synchronization Short Packet Data Type Codes**

1027   Short Packet Data Types shall be transmitted using only the Short Packet format. See section 9.1.2 for a
1028   format description.

1029   **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

1030   **9.8.1   Frame Synchronization Packets**

1031   Each image frame shall begin with a Frame Start (FS) Packet containing the Frame Start Code. The FS
1032   Packet shall be followed by one or more long packets containing image data and zero or more short packets
1033   containing synchronization codes. Each image frame shall end with a Frame End (FE) Packet containing
1034   the Frame End Code. See Table 6 for a description of the synchronization code data types.

1035   For FS and FE synchronization packets the Short Packet Data Field shall contain a 16-bit frame number.
1036   This frame number shall be the same for the FS and FE synchronization packets corresponding to a given
1037   frame.

1038   The 16-bit frame number, when used, shall be non-zero to distinguish it from the use-case where frame
1039   number is inoperative and remains set to zero.

1040   The behavior of the 16-bit frame number shall be as one of the following

1041   •   Frame number is always zero – frame number is inoperative.

1042   •   Frame number increments by 1 for every FS packet with the same Virtual Channel and is
1043   periodically reset to one e.g. 1, 2, 1, 2, 1, 2, 1, 2 or 1, 2, 3, 4, 1, 2, 3, 4

1044   The frame number must be a non-zero value.

9.8.2   Line Synchronization Packets

1045

1046   Line synchronization packets are optional.

1047   For Line Start (LS) and Line End (LE) synchronization packets the Short Packet Data Field shall contain a
1048   16-bit line number. This line number shall be the same for the LS and LE packets corresponding to a given
1049   line. Line numbers are logical line numbers and are not necessarily equal to the physical line numbers

1050   The 16-bit line number, when used, shall be non-zero to distinguish it from the case where line number is
1051   inoperative and remains set to zero.

1052   The behavior of the 16-bit line number shall be as one of the following:

1053   •   Line number is always zero – line number is inoperative.

1054   •   Line number increments by one for every LS packet within the same Virtual Channel and the same
1055   Data Type. The line number is periodically reset to one for the first LS packet after a FS packet.
1056   The intended usage is for progressive scan (non- interlaced) video data streams. The line number
1057   must be a non-zero value.

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

9.9   Generic Short Packet Data Type Codes

1063

1064   Table 7 lists the Generic Short Packet Data Types.

1065   **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

1066   The intention of the Generic Short Packet Data Types is to provide a mechanism for including timing
1067   information for the opening/closing of shutters, triggering of flashes, etc within the data stream. The intent
1068   of the 16-bit User defined data field in the generic short packets is to pass a data type value and a 16-bit

1069 data value from the transmitter to application layer in the receiver. The CSI-2 receiver shall pass the data
1070 type value and the associated 16-bit data value to the application layer.

1071 9.10   Packet Spacing Examples

1072 Packets are separated by an EoT, LPS, SoT sequence as defined in [MIPI01].

1073 Figure 45 and Figure 46 contain examples of data frames composed of multiple packets and a single
1074 packet, respectively.

1075 Note that the VVALID, HVALID and DVALID signals in the figures in this section are only concepts to
1076 help illustrate the behavior of the frame start/end and line start/end packets. The VVALID, HVALID and
1077 DVALID signals do not form part of the specification.
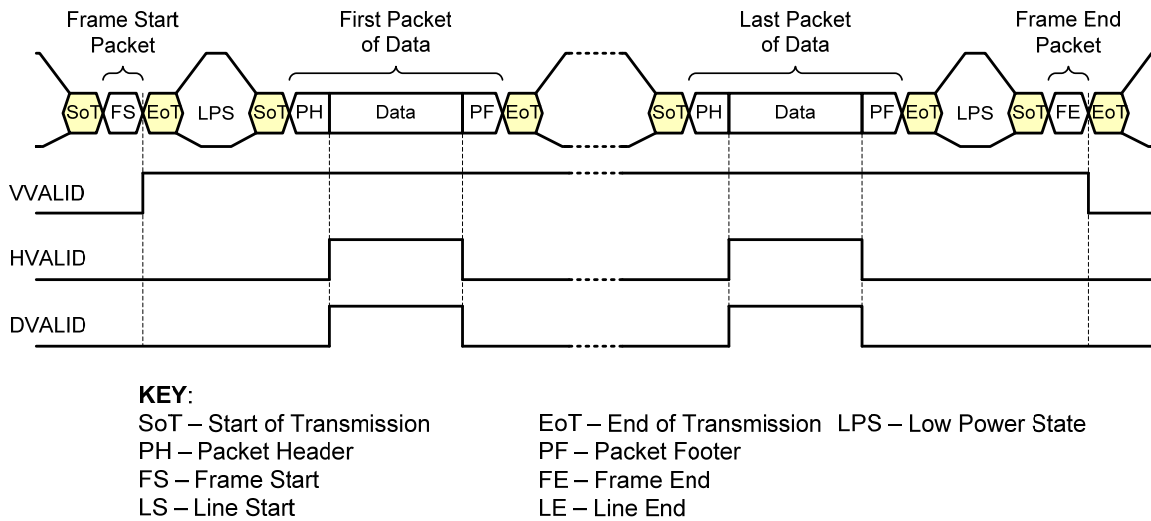
1078
1079

1080   **Figure 45 Multiple Packet Example**
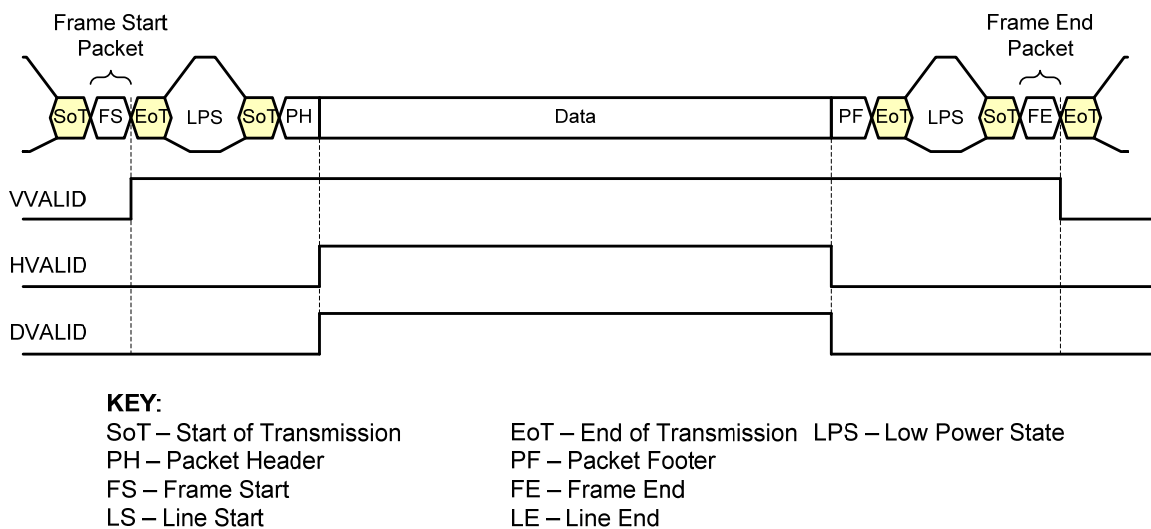
1081

1082   **Figure 46 Single Packet Example**
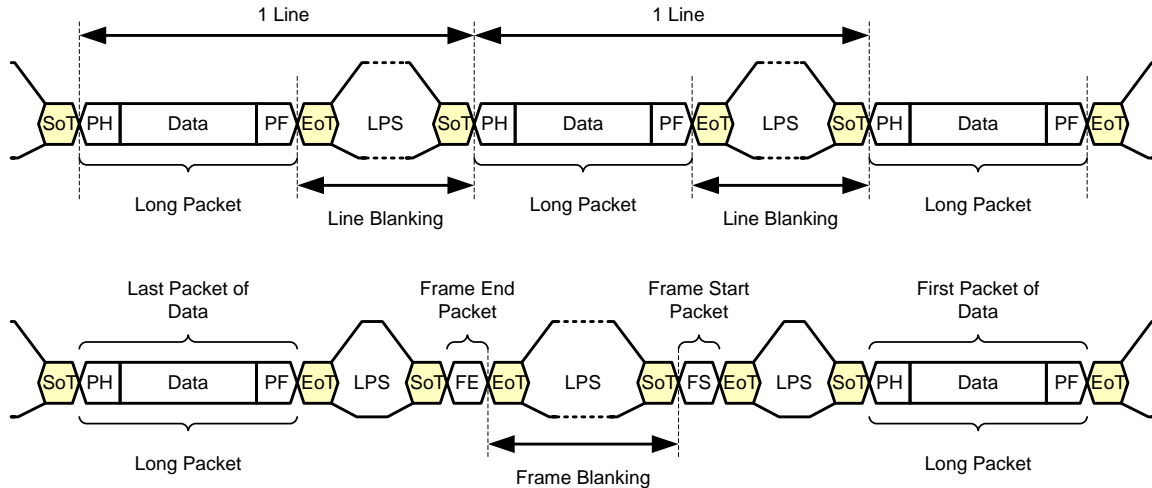
KEY:
SoT – Start of Transmission   EoT – End of Transmission   LPS – Low Power State
PH – Packet Header   PF – Packet Footer
FS – Frame Start   FE – Frame End
LS – Line Start   LE – Line End

1083
1084

1085   **Figure 47 Line and Frame Blanking Definitions**

1086   The period between the Packet Footer of one long packet and the Packet Header of the next long packet is
1087   called the Line Blanking Period.

1088   The period between the Frame End packet in frame N and the Frame Start packet in frame N+1 is called the
1089   Frame Blanking Period (Figure 47).

1090   The Line Blanking Period is not fixed and may vary in length. The receiver should be able to cope with a
1091   near zero Line Blanking Period as defined in [MIPI01]. The transmitter defines the minimum time for the
1092   Frame Blanking Period. The Frame Blanking Period duration should be programmable in the transmitter.

1093   Frame Start and Frame End packets shall be used.

1094   Recommendations (informative) for frame start and end packet spacing:

1095   • The Frame Start packet to first data packet spacing should be as close as possible to the minimum
1096   packet spacing

1097   • The last data packet to Frame End packet spacing should be as close as possible to the minimum
1098   packet spacing

1099   The intention is to ensure that the Frame Start and Frame End packets accurately denote the start and end of
1100   a frame of image data. A valid exception is when the positions of the Frame Start and Frame End packets
1101   are being used to convey pixel level accurate vertical synchronization timing information.

1102   The positions of the Frame Start and Frame End packets can be varied within the Frame Blanking Period in
1103   order to provide pixel level accurate vertical synchronization timing information. See Figure 48.

1104   Line Start and Line End packets shall be used for pixel level accurate horizontal synchronization timing
1105   information.

1106 The positions of the Line Start and Line End packets, if present, can be varied within the Line Blanking
1107 Period in order to provide pixel accurate horizontal synchronization timing information. See Figure 49.

1108
1109
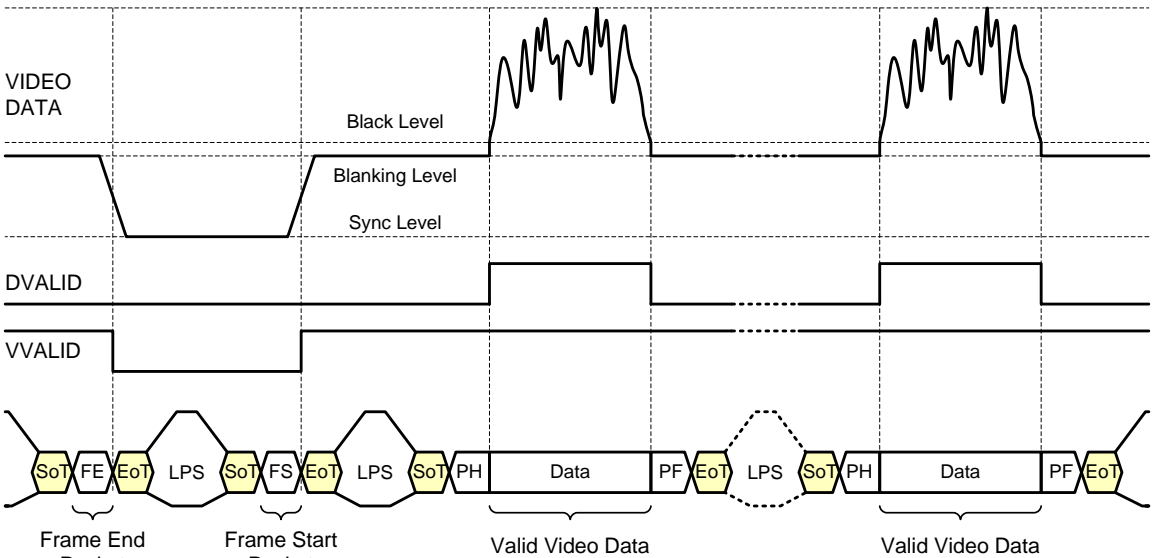
1110 **Figure 48 Vertical Sync Example**

1111
1112
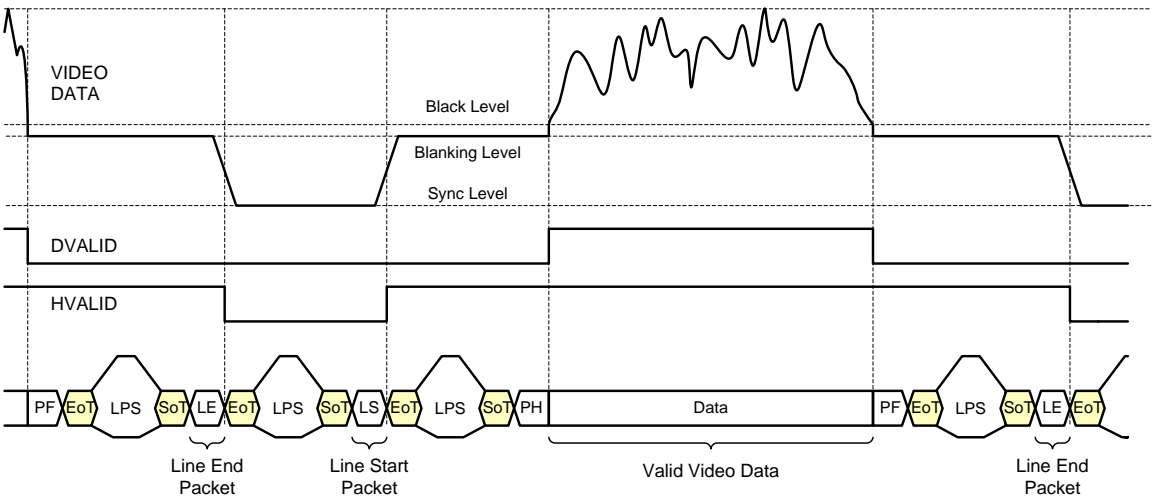
1113 **Figure 49 Horizontal Sync Example**

1114 **9.11   Packet Data Payload Size Rules**

1115 For YUV, RGB or RAW data types, one long packet shall contain one line of image data. Each long packet
1116 of the same Data Type shall have equal length when packets are within the same Virtual Channel and when
1117 packets are within the same frame. An exception to this rule is the YUV420 data type which is defined in
1118 section 11.2.2.

1119 For User Defined Byte-based Data Types, long packets can have arbitrary length. The spacing between
1120 packets can also vary.

1121 The total size of data within a long packet for all data types shall be a multiple of eight bits. However, it is
1122 also possible that a data type's payload data transmission format, as defined elsewhere in this specification,
1123 imposes additional constraints on payload size. In order to meet these constraints it may sometimes be
1124 necessary to add some number of "padding" pixels to the end of a payload e.g., when a packet with the
1125 RAW10 data type contains an image line whose length is not a multiple of four pixels as required by the
1126 RAW10 transmission format as described in Section 11.4.4. The values of such padding pixels are not
1127 specified.

1128 ## 9.12   Frame Format Examples

1129 This in an informative section.

1130 This section contains three examples to illustrate how the CSI-2 features can be used.

1131 • General Frame Format Example, Figure 50

1132 • Digital Interlaced Video Example, Figure 51

1133 • Digital Interlaced Video with accurate synchronization timing information, Figure 52

1134 **KEY**:
PH – Packet Header   PF – Packet Footer
FS – Frame Start   FE – Frame End
LS – Line Start   LE – Line End
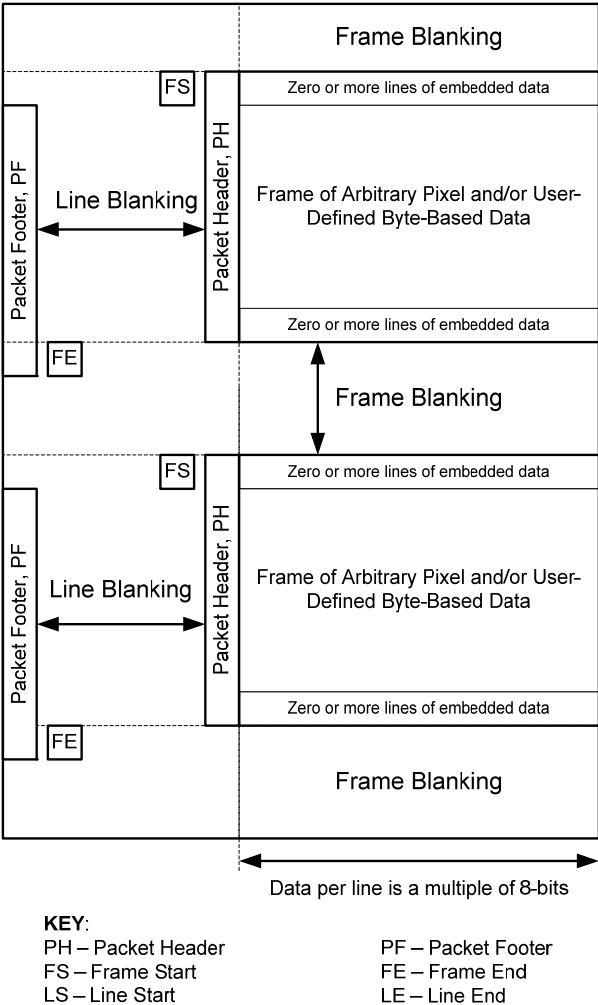
1135 **Figure 50 General Frame Format Example**
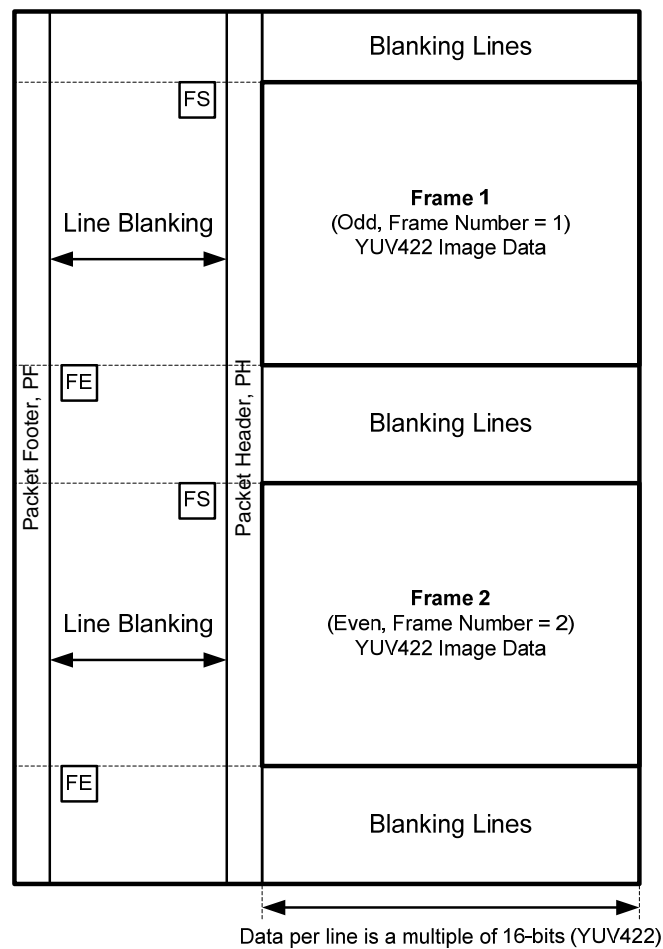
KEY:
PH – Packet Header   PF – Packet Footer
FS – Frame Start   FE – Frame End
LS – Line Start   LE – Line End

1136

1137   **Figure 51 Digital Interlaced Video Example**

KEY:
PH – Packet Header   PF – Packet Footer
FS – Frame Start   FE – Frame End
LS – Line Start   LE – Line End

1138
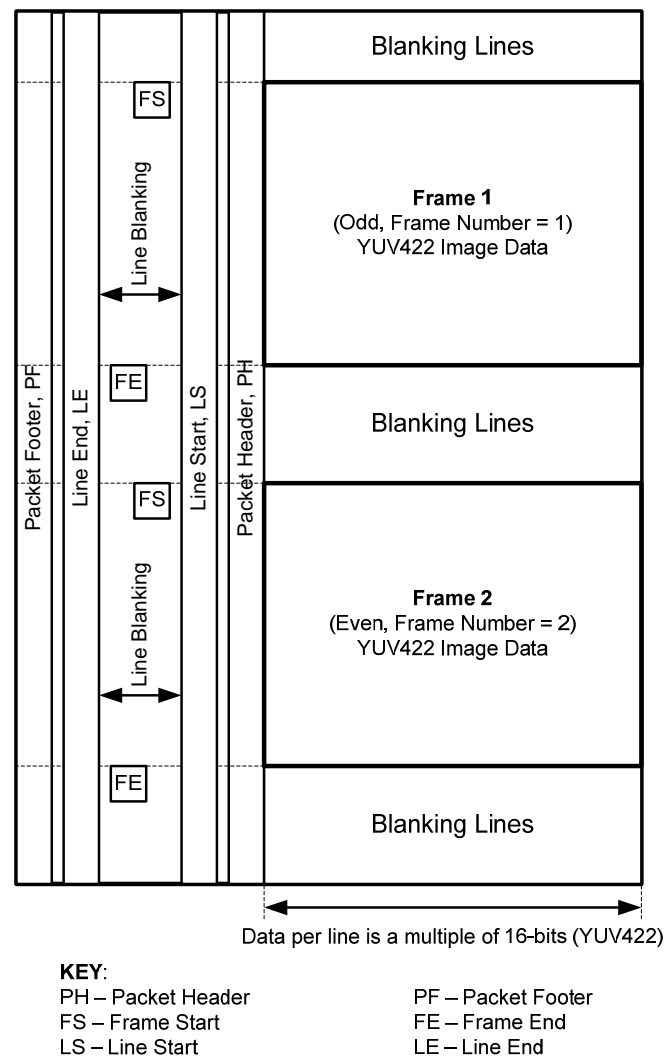
1139   **Figure 52 Digital Interlaced Video with Accurate Synchronization Timing Information**

1140   **9.13   Data Interleaving**

1141   The CSI-2 supports the interleaved transmission of different image data formats within the same video data
1142   stream.

1143   There are two methods to interleave the transmission of different image data formats:

1144   • Data Type

1145   • Virtual Channel Identifier

1146   The above methods of interleaved data transmission can be combined in any manner.

1147   **9.13.1   Data Type Interleaving**

1148   The Data Type value uniquely defines the data format for that packet of data. The receiver uses the Data
1149   Type value in the packet header to de-multiplex data packets containing different data formats as illustrated
1150   in Figure 53. Note, in the figure the Virtual Channel Identifier is the same in each Packet Header.

1151   The packet payload data format shall agree with the Data Type code in the Packet Header as follows:

1152   • For defined image data types – any non-reserved codes in the range 0x18 to 0x3F – only the single
1153   corresponding MIPI-defined packet payload data format shall be considered correct

1154   • Reserved image data types – any reserved codes in the range 0x18 to 0x3F – shall not be used. No
1155   packet payload data format shall be considered correct for reserved image data types

1156   • For generic long packet data types (codes 0x10 thru 0x17) and user-defined, byte-based (codes
1157   0x30 – 0x37), any packet payload data format shall be considered correct

1158   • Generic long packet data types (codes 0x10 thru 0x17) and user-defined, byte-based (codes 0x30 –
1159   0x37), should not be used with packet payloads that meet any MIPI image data format definition

1160   • Synchronization short packet data types (codes 0x00 thru 0x07) shall consist of only the header
1161   and shall not include payload data bytes

1162   • Generic short packet data types (codes 0x08 thru 0x0F) shall consist of only the header and shall
1163   not include payload data bytes

1164   Data formats are defined further in section 11.

1165
1166
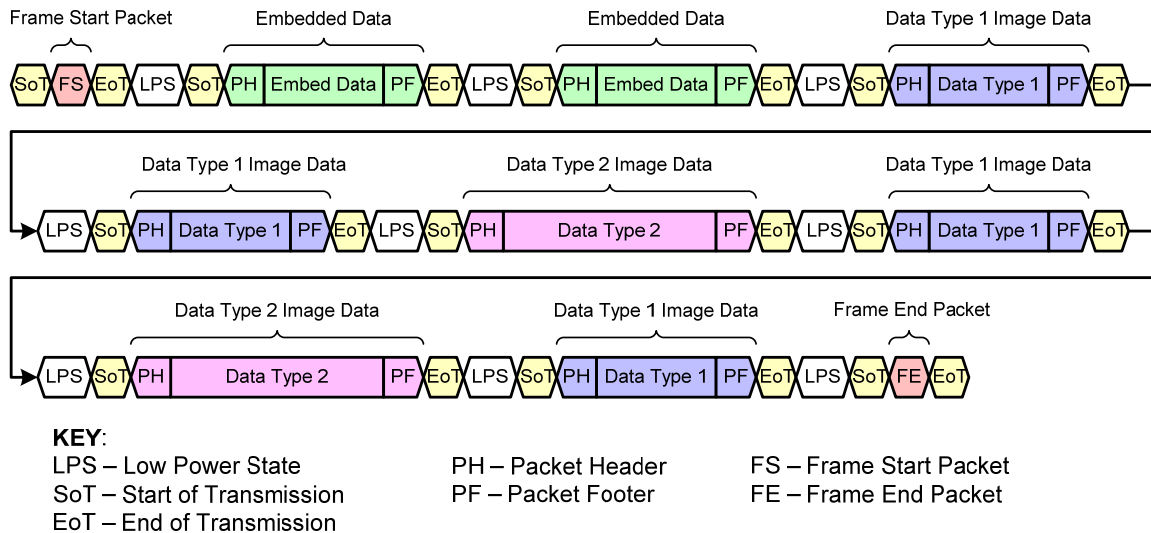
1167   **Figure 53 Interleaved Data Transmission using Data Type Value**

1168   All of the packets within the same virtual channel, independent of the Data Type value, share the same
1169   frame start/end and line start/end synchronization information. By definition, all of the packets,
1170   independent of data type, between a Frame Start and a Frame End packet within the same virtual channel
1171   belong to the same frame.

1172   Packets of different data types may be interleaved at either the packet level as illustrated in Figure 54 or the
1173   frame level as illustrated in Figure 55. Data formats are defined in section 11.
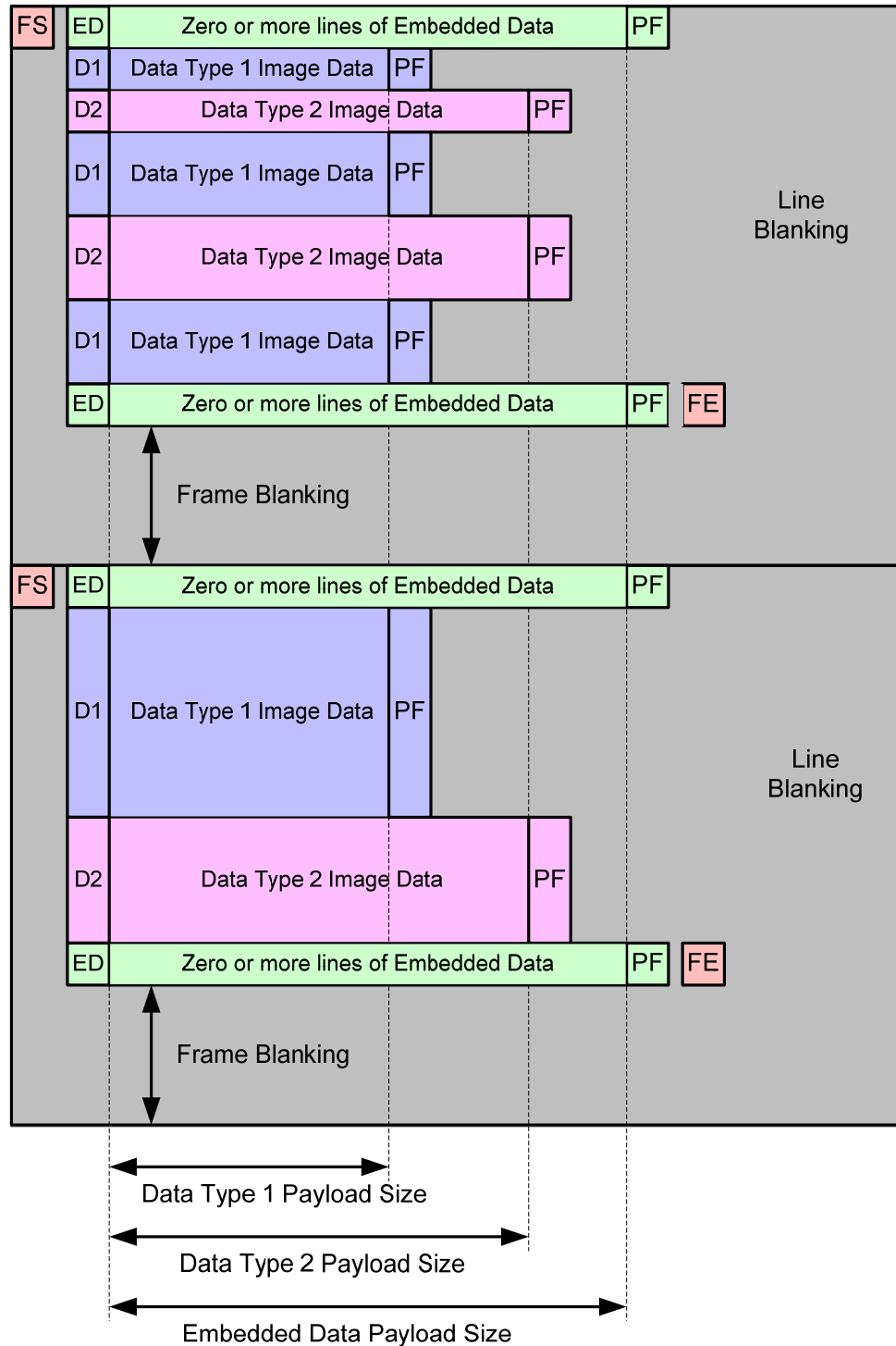
KEY:
LPS – Low Power State   ED – Packet Header containing Embedded Data type code
FS – Frame Start   D1 – Packet Header containing Data Type 1 Image Data Code
FE – Frame End   D2 – Packet Header containing Data Type 2 Image Data Code
PF – Packet Footer

Figure 54 Packet Level Interleaved Data Transmission
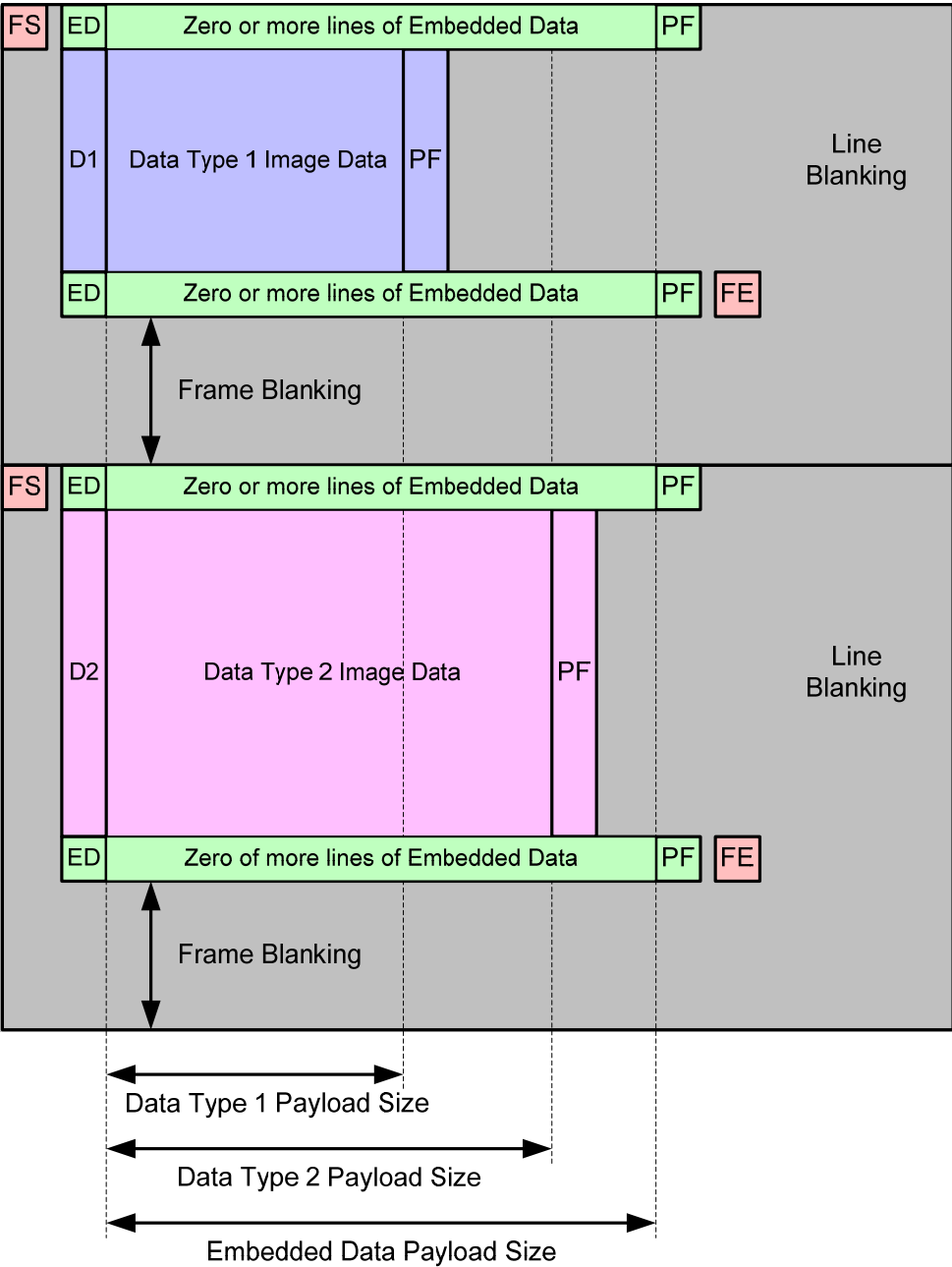
KEY:
LPS – Low Power State   ED – Packet Header containing Embedded Data type code
FS – Frame Start   D1 – Packet Header containing Data Type 1 Image Data Code
FE – Frame End   D2 – Packet Header containing Data Type 2 Image Data Code
PF – Packet Footer

1176

1177   **Figure 55 Frame Level Interleaved Data Transmission**

1178 **9.13.2   Virtual Channel Identifier Interleaving**

1179 The Virtual Channel Identifier allows different data types within a single data stream to be logically
1180 separated from each other. Figure 56 illustrates data interleaving using the Virtual Channel Identifier.

1181 Each virtual channel has its own Frame Start and Frame End packet. Therefore, it is possible for different
1182 virtual channels to have different frame rates, though the data rate for both channels would remain the
1183 same.

1184 In addition, Data Type value Interleaving can be used for each virtual channel thereby allowing different
1185 data types within a virtual channel and thus a second level of data interleaving.

1186 Therefore, receivers should be able to de-multiplex different data packets based on the combination of the
1187 Virtual Channel Identifier and the Data Type value. For example, data packets containing the same Data
1188 Type value but transmitted on different virtual channels are considered to belong to different frames
1189 (streams) of image data.

1190
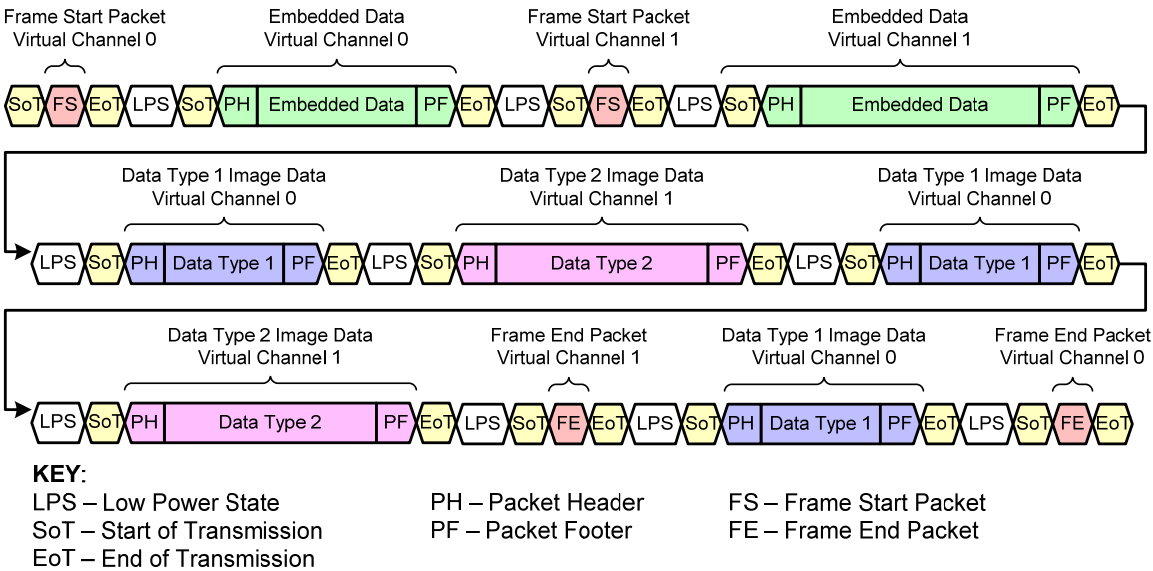
1191 **Figure 56 Interleaved Data Transmission using Virtual Channels**

1192

10 Color Spaces

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

10.1 RGB Color Space Definition

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

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

10.2 YUV Color Space Definition

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

11 Data Formats

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

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

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

Table 8 Primary and Secondary Data Formats Definitions

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

Notes:

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

1220   For clarity the Start of Transmission and End of Transmission sequences in the figures in this section have
1221   been omitted.

11.1   Generic 8-bit Long Packet Data Types

1223   Table 9 defines the generic 8-bit Long packet data types.

1224   **Table 9 Generic 8-bit Long Packet Data Types**

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

11.1.1   Null and Blanking Data

1226   For both the null and blanking data types the receiver must ignore the content of the packet payload data.

1227   A blanking packet differs from a null packet in terms of its significance within a video data stream. A null
1228   packet has no meaning whereas the blanking packet may be used, for example, as the blanking lines
1229   between frames in an ITU-R BT.656 style video stream.

11.1.2   Embedded Information

1231   It is possible to embed extra lines containing additional information to the beginning and to the end of each
1232   picture frame as presented in the Figure 57. If embedded information exists, then the lines containing the
1233   embedded data must use the embedded data code in the data identifier.

1234   There may be zero or more lines of embedded data at the start of the frame. These lines are termed the
1235   frame header.

1236   There may be zero or more line of embedded data at the end of the frame. These lines are termed the frame
1237   footer.

11.2   YUV Image Data

1239   Table 10 defines the data type codes for YUV data formats described in this section. The number of lines
1240   transmitted for the YUV420 data type shall be even.

1241   YUV420 data formats are divided into legacy and non-legacy data formats. The legacy YUV420 data
1242   format is for compatibility with existing systems. The non-legacy YUV420 data formats enable lower cost
1243   implementations.
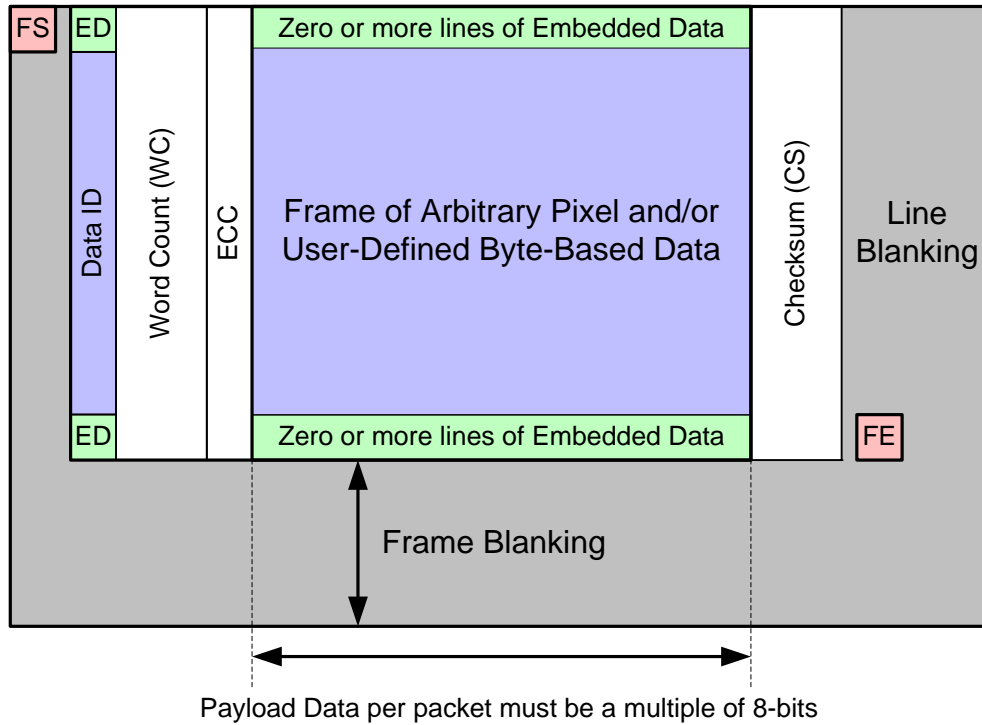
Payload Data per packet must be a multiple of 8-bits

KEY:

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

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

Table 10 YUV Image Data Types

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

11.2.1　Legacy YUV420 8-bit

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

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

1253

Table 11 Legacy YUV420 8-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

1254
1255

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

1256
1257

1258
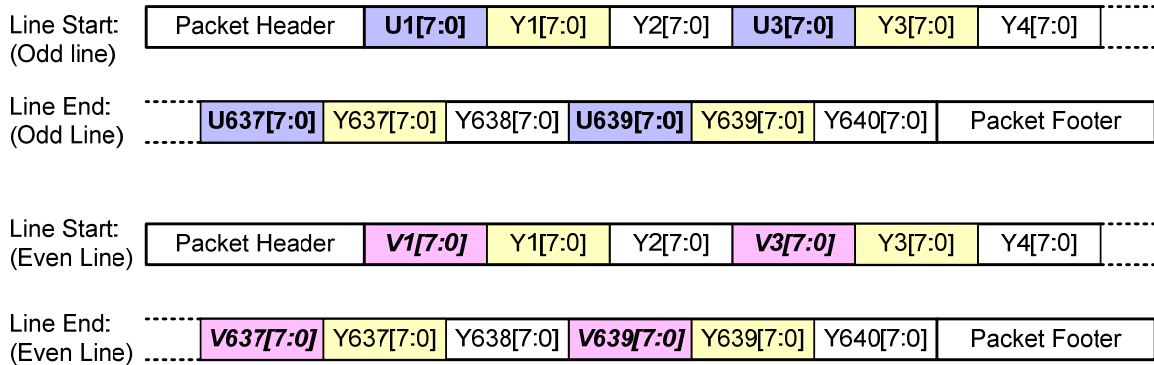
Figure 58 Legacy YUV420 8-bit Transmission

1259
1260

1261
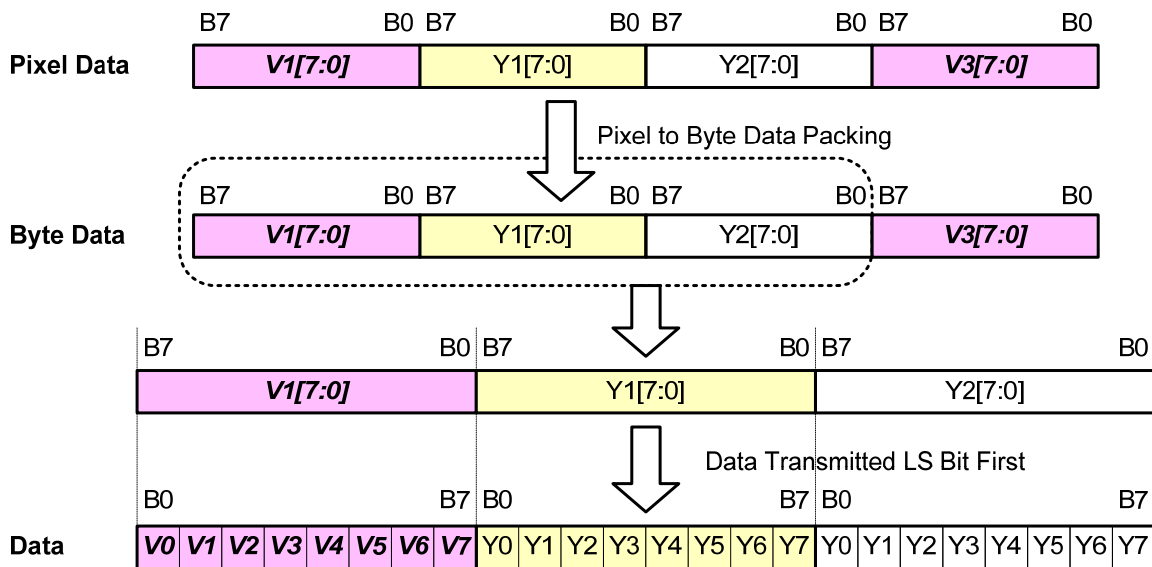
Figure 59 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

1262

There is one spatial sampling option

1263

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

1264
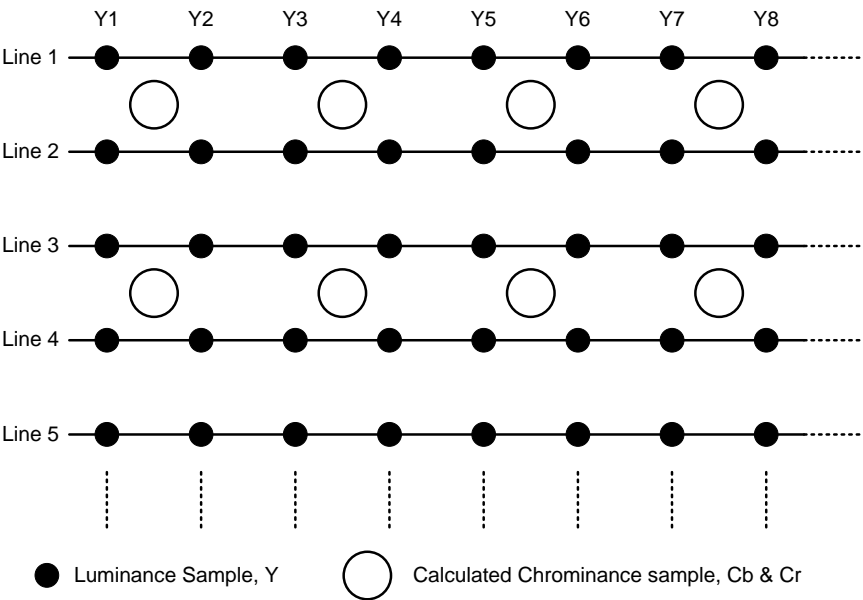
1265   **Figure 60 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1**

1266
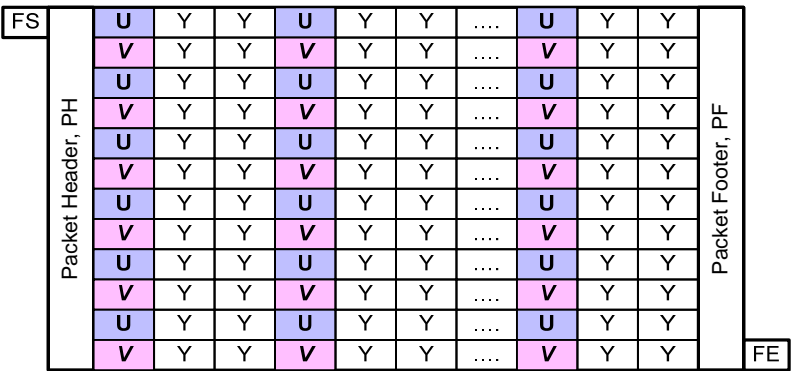
1267   **Figure 61 Legacy YUV420 8-bit Frame Format**

1268   **11.2.2   YUV420 8-bit**

1269   YUV420 8-bit data transmission is performed by transmitting YYYY… / UYVYUYVY… sequences in
1270   odd / even lines. Only the luminance component (Y) is transferred for odd lines (1, 3, 5…) and both
1271   luminance (Y) and chrominance (U and V) components are transferred for even lines (2, 4, 6…). The
1272   format for the even lines (UYVY) is identical to the YUV422 8-bit data format. The data transmission
1273   sequence is illustrated in Figure 62.

1274   The payload data size, in bytes, for even lines (UYVY) is double the payload data size for odd lines (Y).
1275   This is exception to the general CSI-2 rule that each line shall have an equal length.

1276   Table 12 specifies the packet size constraints for YUV420 8-bit packets. Each packet must be a multiple of
1277   the values in the table.

1278 **Table 12 YUV420 8-bit Packet Data Size Constraints**

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

1279 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
1280 in Figure 63.

1281
1282
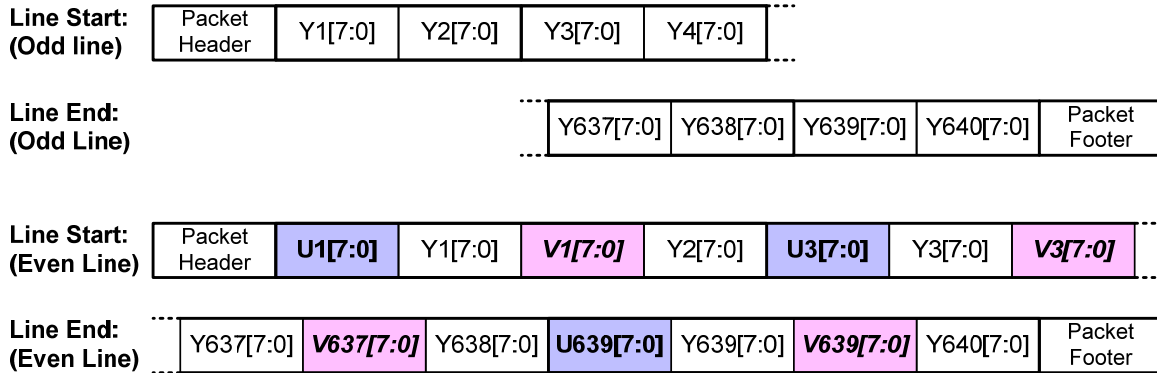
1283 **Figure 62 YUV420 8-bit Data Transmission Sequence**

1284
1285
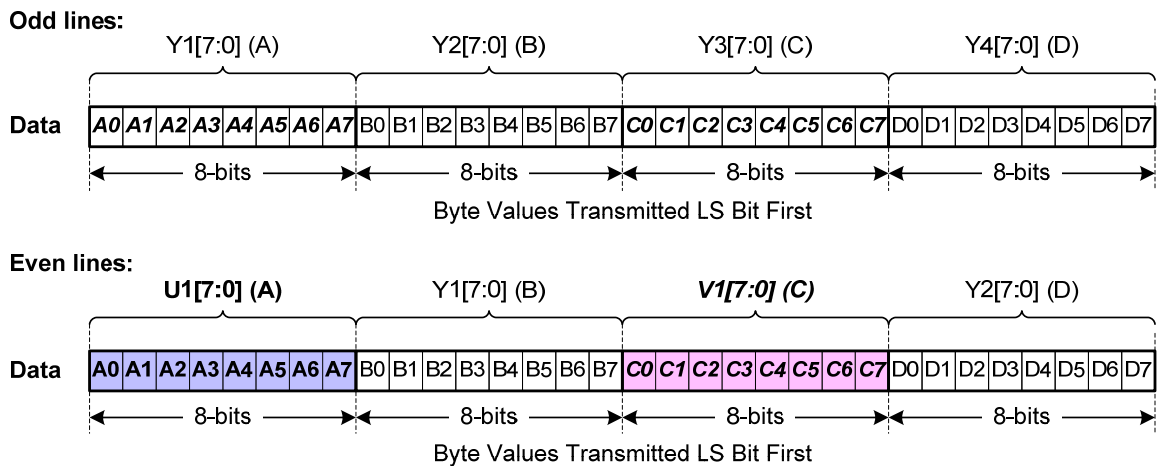
1286 **Figure 63 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration**

1287 There are two spatial sampling options

1288 • H.261, H.263 and MPEG1 Spatial Sampling (Figure 64).

1289 • Chroma Shifted Pixel Sampling (CSPS) for MPEG2, MPEG4 (Figure 65).

1290 Figure 66 shows the YUV420 frame format.
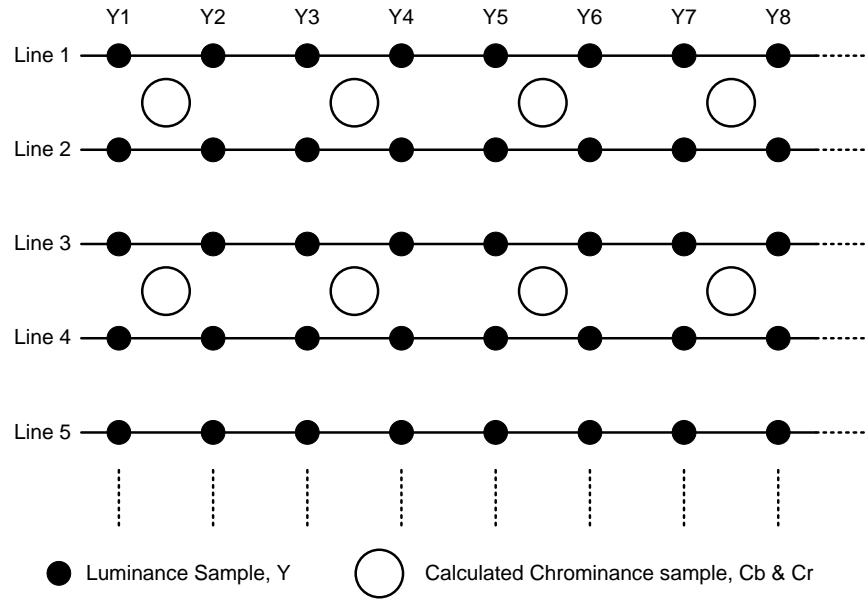
Figure 64 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1
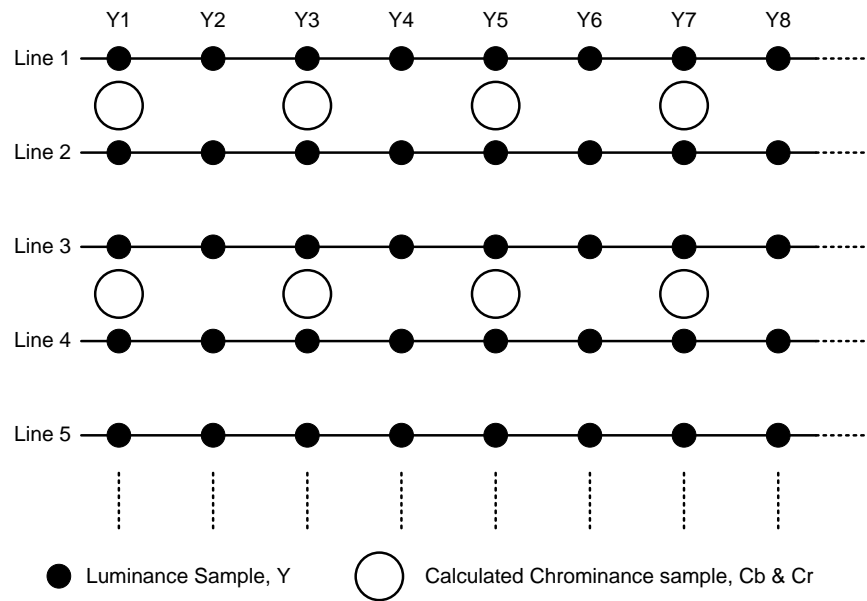
Figure 65 YUV420 Spatial Sampling for MPEG 2 and MPEG 4

1295
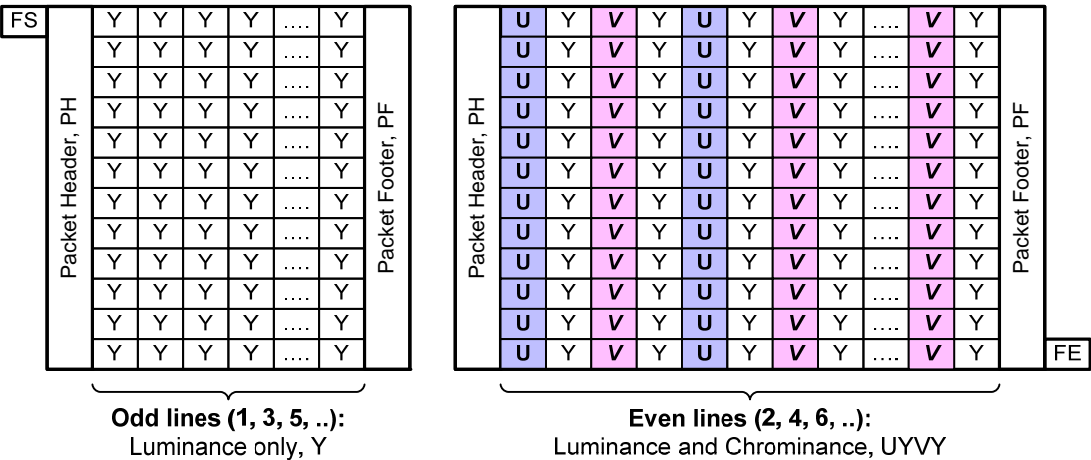
1296   **Figure 66 YUV420 8-bit Frame Format**

11.2.3   YUV420 10-bit

1298   YUV420 10-bit data transmission is performed by transmitting YYYY… / UYVYUYVY… sequences in
1299   odd / even lines. Only the luminance component (Y) is transferred in odd lines (1, 3, 5…) and both
1300   luminance (Y) and chrominance (U and V) components transferred in even lines (2, 4, 6…). The format for
1301   the even lines (UYVY) is identical to the YUV422 –10-bit data format. The sequence is illustrated in
1302   Figure 67.

1303   The payload data size, in bytes, for even lines (UYVY) is double the payload data size for odd lines (Y).
1304   This is exception to the general CSI-2 rule that each line shall have an equal length.

1305   Table 13 specifies the packet size constraints for YUV420 10-bit packets. The length of each packet must
1306   be a multiple of the values in the table.

1307   **Table 13 YUV420 10-bit Packet Data Size Constraints**

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

1308   Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
1309   in Figure 68.

Figure 67 YUV420 10-bit Transmission
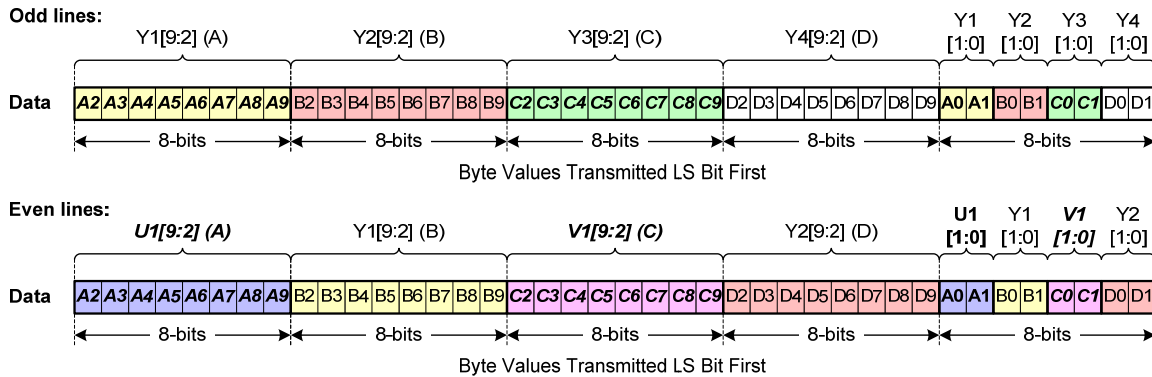
Figure 68 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration

The pixel spatial sampling options are the same as for the YUV420 8-bit data format.
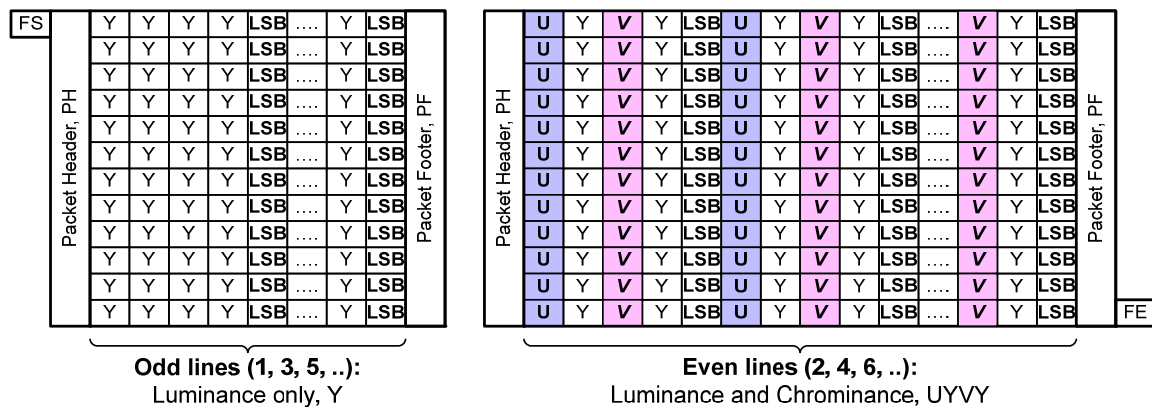
Figure 69 YUV420 10-bit Frame Format

1319 **11.2.4   YUV422 8-bit**

1320 YUV422 8-bit data transmission is performed by transmitting a UYVY sequence. This sequence is
1321 illustrated in Figure 70.

1322 Table 14 specifies the packet size constraints for YUV422 8-bit packet. The length of each packet must be
1323 a multiple of the values in the table.

1324   **Table 14 YUV422 8-bit Packet Data Size Constraints**

Pixels	Bytes	Bits
2	4	32

1325 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
1326 in Figure 71.

1327
1328

1329   **Figure 70 YUV422 8-bit Transmission**

1330
1331
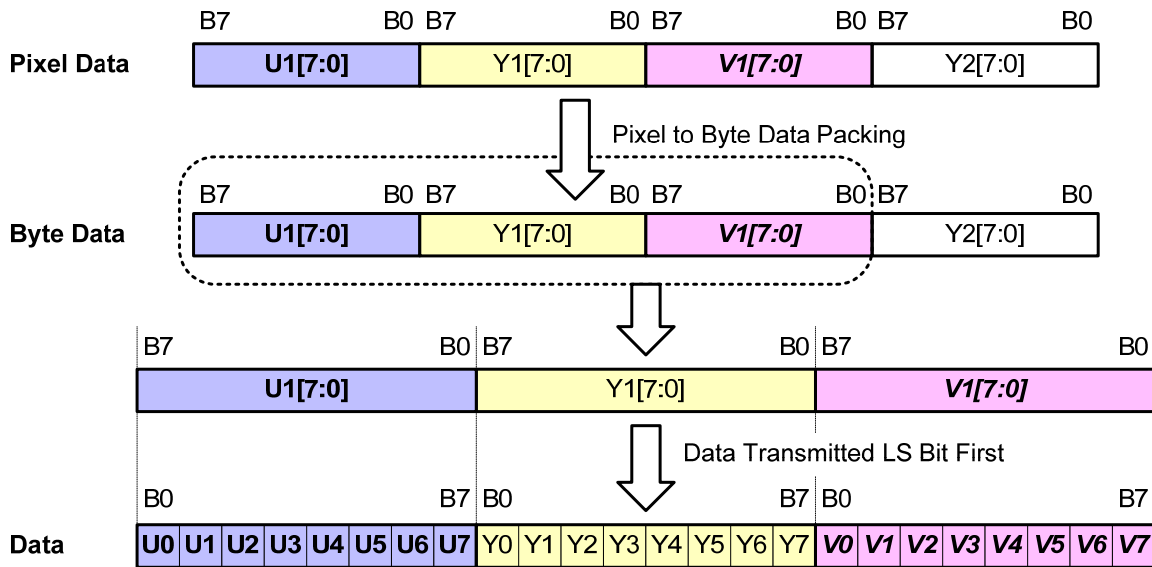
1332   **Figure 71 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration**

1333
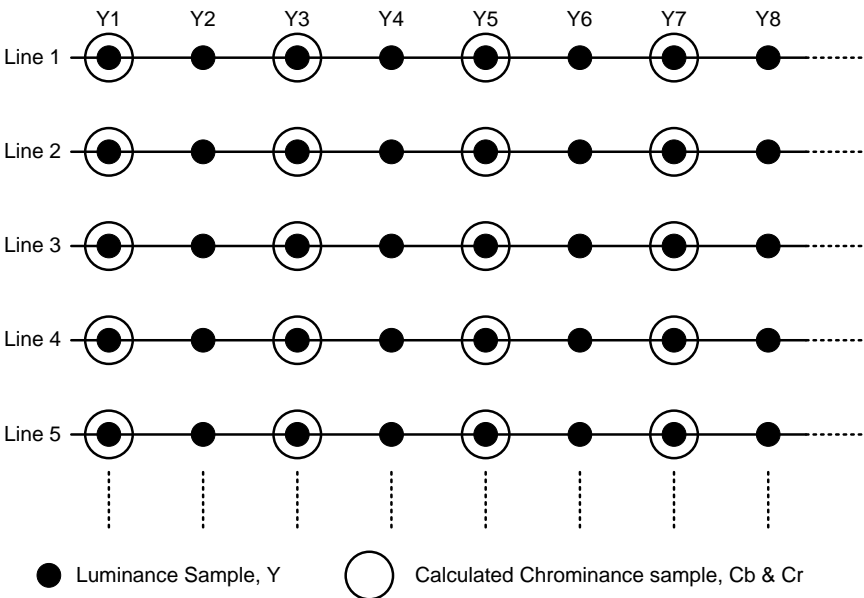
Figure 72 YUV422 Co-sited Spatial Sampling

1335 The pixel spatial alignment is the same as in CCIR-656 standard. The frame format for YUV422 is
1336 presented in Figure 73.

1337
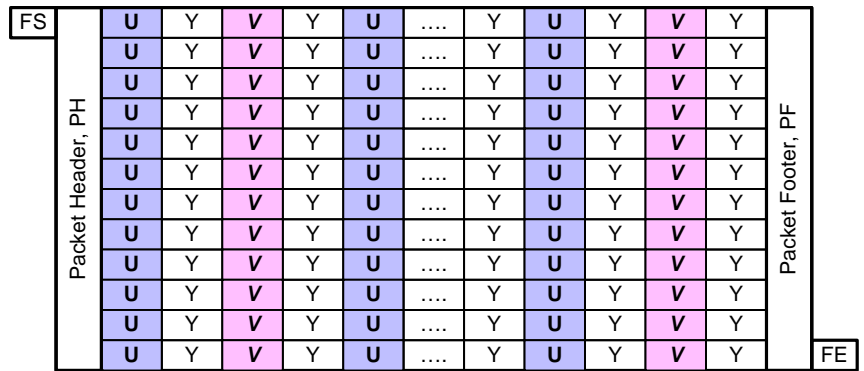
Figure 73 YUV422 8-bit Frame Format

11.2.5   YUV422 10-bit

1340 YUV422 10-bit data transmission is performed by transmitting a UYVY sequence. This sequence is
1341 illustrated in Figure 74.

1342 Table 15 specifies the packet size constraints for YUV422 10-bit packet. The length of each packet must be
1343 a multiple of the values in the table.

1344   **Table 15 YUV422 10-bit Packet Data Size Constraints**

Pixels	Bytes	Bits
2	5	40

1345 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
1346 in Figure 75.
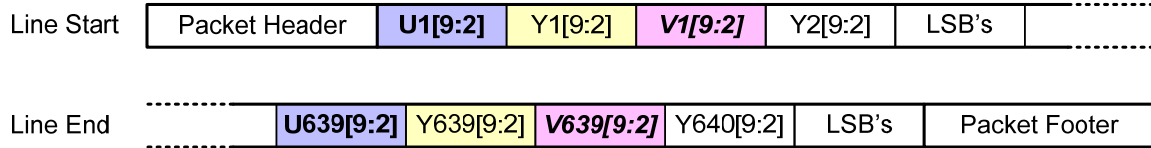
Figure 74 YUV422 10-bit Transmitted Bytes

1347
1348
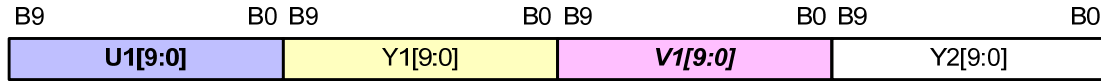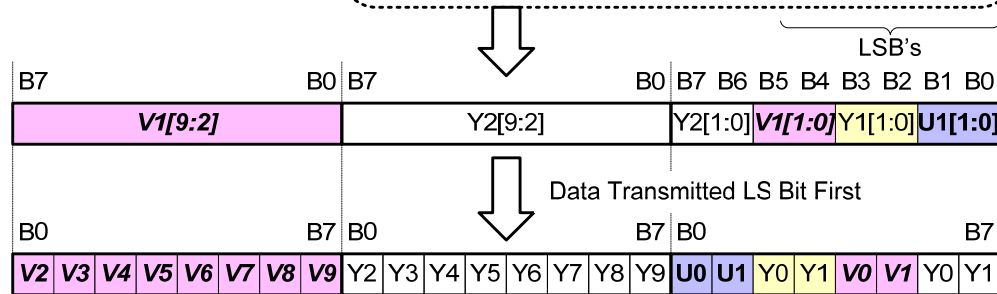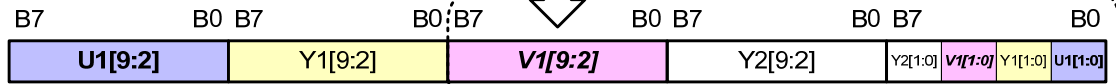
Figure 75 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration

1349
1350
1351

1352 The pixel spatial alignment is the same as in the YUV422 8-bit data case. The frame format for YUV422 is
1353 presented in the Figure 76.
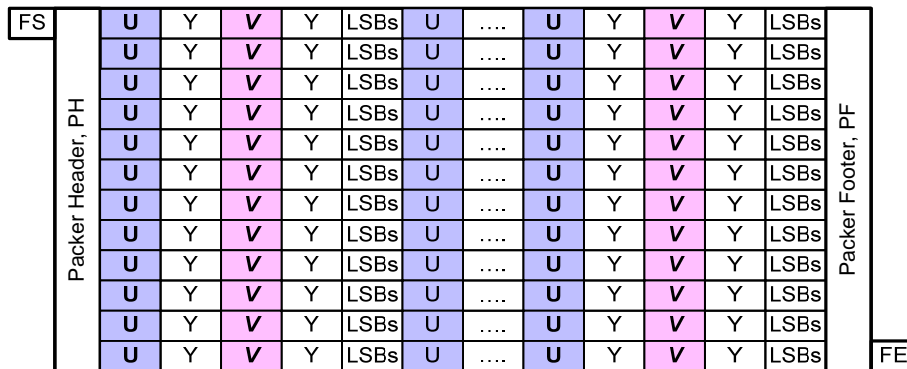
Figure 76 YUV422 10-bit Frame Format

1354
1355

11.3   RGB Image Data

1356

1357 Table 16 defines the data type codes for RGB data formats described in this section.

1358   **Table 16 RGB Image Data Types**

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

1359   ### 11.3.1   RGB888

1360   RGB888 data transmission is performed by transmitting a BGR byte sequence. This sequence is illustrated
1361   in Figure 77. The RGB888 frame format is illustrated in Figure 79.

1362   Table 17 specifies the packet size constraints for RGB888 packets. The length of each packet must be a
1363   multiple of the values in the table.

1364   **Table 17 RGB888 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	3	24

1365   Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
1366   in Figure 78.

1367
1368
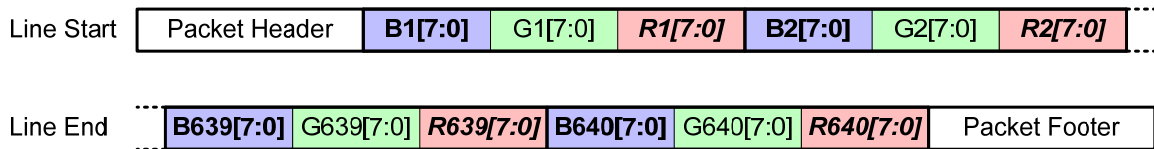
1369   **Figure 77 RGB888 Transmission**

1370
1371
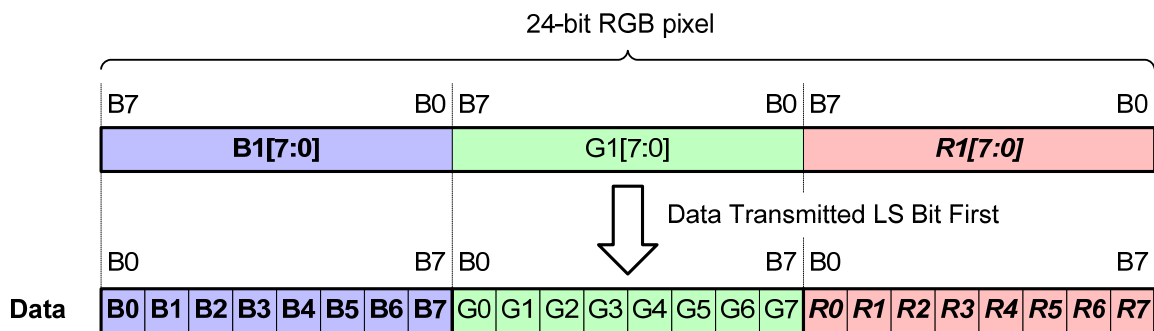
1372   **Figure 78 RGB888 Transmission in CSI-2 Bus Bitwise Illustration**
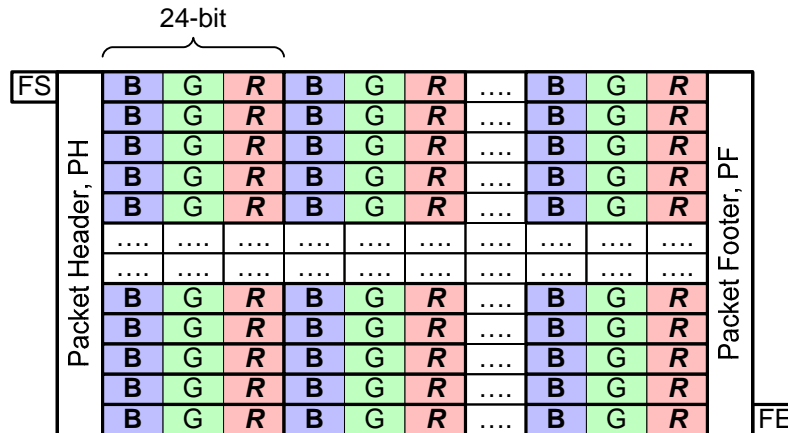
Figure 79 RGB888 Frame Format

11.3.2 RGB666

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

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

Table 18 RGB666 Packet Data Size Constraints

Pixels	Bytes	Bits
4	9	72

Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB666 case the length of one data word is 18-bits, not eight bits. The word wise flip is done for 18-bit BGR words i.e. instead of flipping each byte (8-bits), each 18-bits pixel value is flipped. This is illustrated in Figure 81.
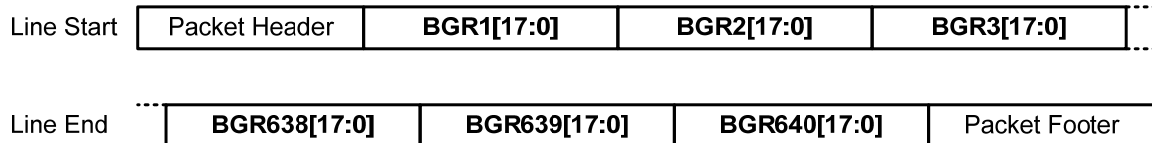
Figure 80 RGB666 Transmission with 18-bit BGR Words
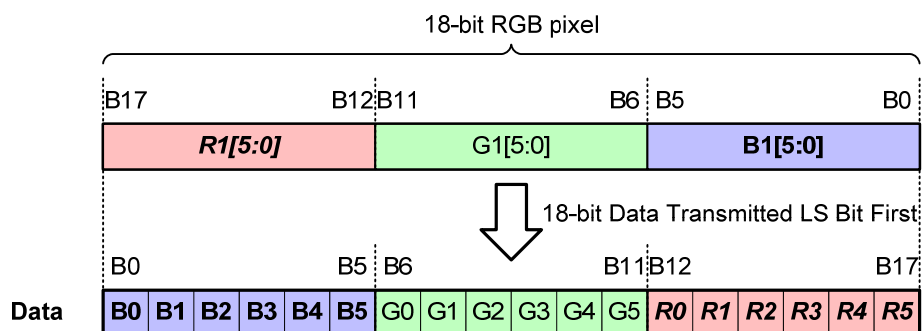
Figure 81 RGB666 Transmission on CSI-2 Bus Bitwise Illustration
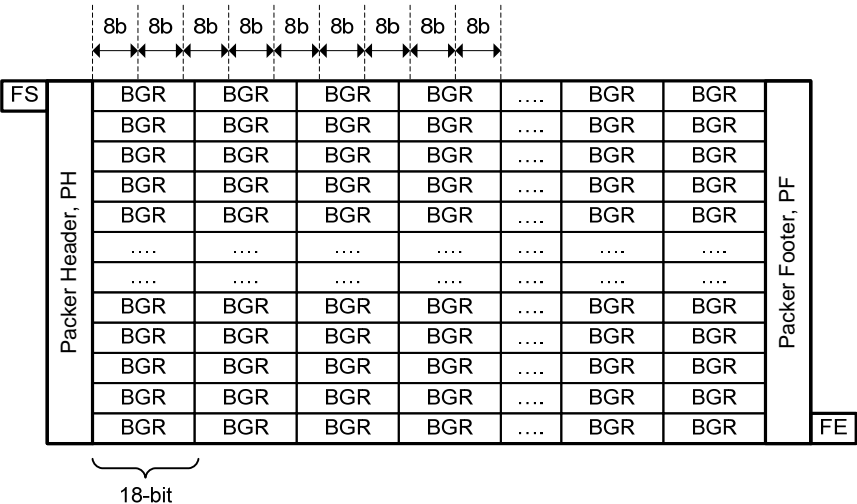
Figure 82 RGB666 Frame Format

11.3.3   RGB565

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

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

Table 19 RGB565 Packet Data Size Constraints

Pixels	Bytes	Bits
1	2	16

Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB565 case the length of one data word is 16-bits, not eight bits. The word wise flip is done for 16-bit BGR words i.e. instead of flipping each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 84.
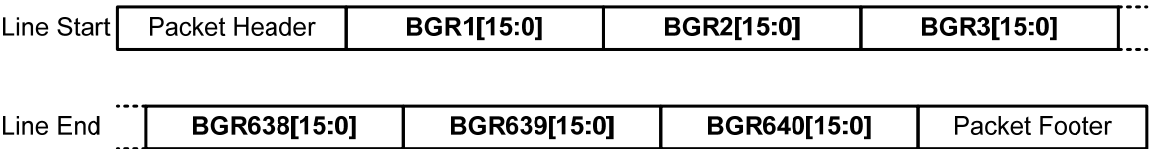
Figure 83 RGB565 Transmission with 16-bit BGR Words

16-bit RGB pixel

B15	B11	B10	B5	B4	B0
R1[4:0]		G1[5:0]		**B1[4:0]**	

⇩ 16-bit Data Transmitted LS Bit First

B0				B4	B5					B10	B11				B15
Data	B0	B1	B2	B3	B4	G0	G1	G2	G3	G4	G5	*R0*	*R1*	*R2*	*R3*	*R4*

1402
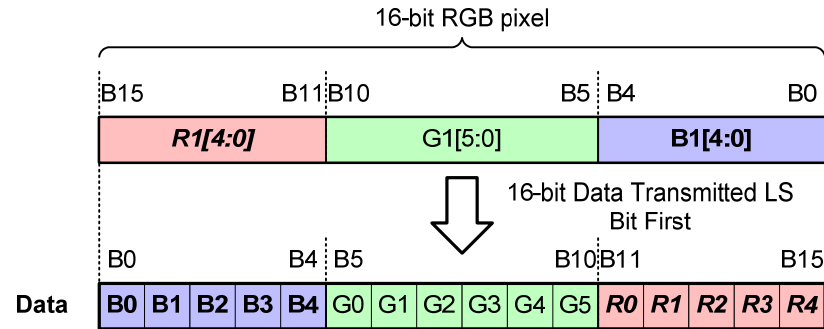
1403   **Figure 84 RGB565 Transmission on CSI-2 Bus Bitwise Illustration**

16-bit

FS	BGR	BGR	BGR	....	BGR	BGR	PF
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	
Packer Header, PH	BGR	BGR	BGR	....	BGR	BGR	Packer Footer, PF
	BGR	BGR	BGR	....	BGR	BGR	
	....	....	....	....	....	....	
	....	....	....	....	....	....	
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	FE

1404
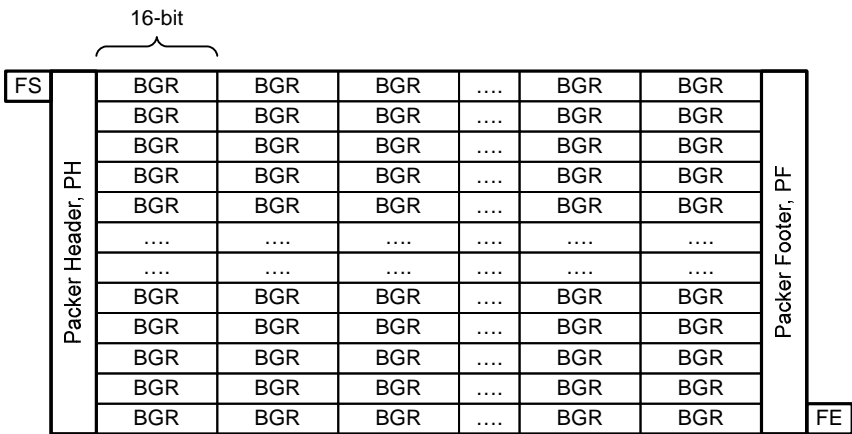
1405   **Figure 85 RGB565 Frame Format**

1406   ## 11.3.4   RGB555

1407   RGB555 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB555 data
1408   should be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs
1409   of the green color component as illustrated in Figure 86.

1410   Both the frame format and the package size constraints are the same as the RGB565 case.

1411   Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB555 case the length of one data
1412   word is 16-bits, not eight bits. The word wise flip is done for 16-bit BGR words i.e. instead of flipping each
1413   byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 86.

15-bit RGB pixel padded to 16-bits

B15	B11	B10	B5	B4	B0
R1[4:0]		G1[4:0]	0	**B1[4:0]**	

⇩ 16-bit Data Transmitted LS Bit First

B0				B4	B5						B10	B11				B15
Data	B0	B1	B2	B3	B4	0	G0	G1	G2	G3	G4	*R0*	*R1*	*R2*	*R3*	*R4*

1414
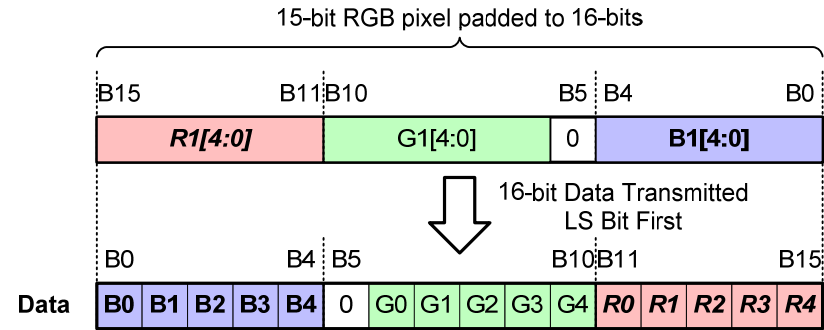
1415   **Figure 86 RGB555 Transmission on CSI-2 Bus Bitwise Illustration**

11.3.5 RGB444

RGB444 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB444 data should be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs of each color component as illustrated in Figure 87.

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

Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB444 case the length of one data word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words i.e. instead of flipping each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in Figure 87.
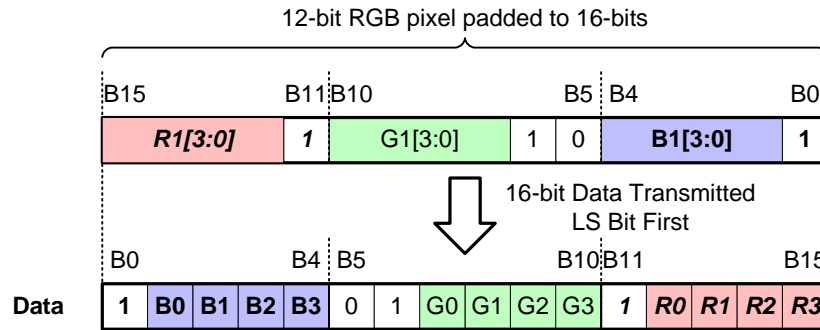
Figure 87 RGB444 Transmission on CSI-2 Bus Bitwise Illustration

11.4 RAW Image Data

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

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

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

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

Table 20 RAW Image Data Types

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

1435 ## 11.4.1   RAW6

1436 The 6-bit Raw data transmission is performed by transmitting the pixel data over CSI-2 bus. Each line is
1437 separated by line start / end synchronization codes. This sequence is illustrated in Figure 88 (VGA case).
1438 Table 21 specifies the packet size constraints for RAW6 packets. The length of each packet must be a
1439 multiple of the values in the table.

1440 **Table 21 RAW6 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	3	24

1441 Each 6-bit pixel is sent LSB first. This is an exception to general CSI-2 rule byte wise LSB first.

Line Start	Packet Header	**P1[5:0]**	P2[5:0]	*P3[5:0]*	P4[5:0]	**P5[5:0]**	P6[5:0]	*P7[5:0]*

Line End	P634[5:0]	*P635[5:0]*	P636[5:0]	**P637[5:0]**	P638[5:0]	*P639[5:0]*	P640[5:0]	Packet Footer
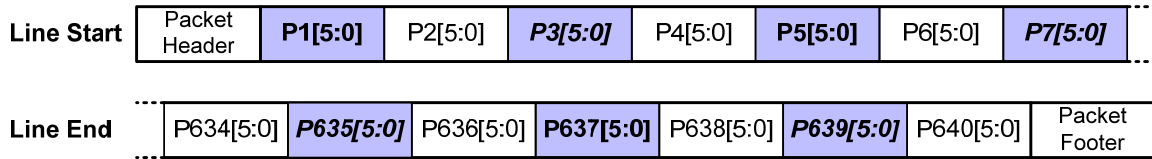
1442
1443 **Figure 88 RAW6 Transmission**

1444 6-bit Pixel Values Transmitted LS Bit First
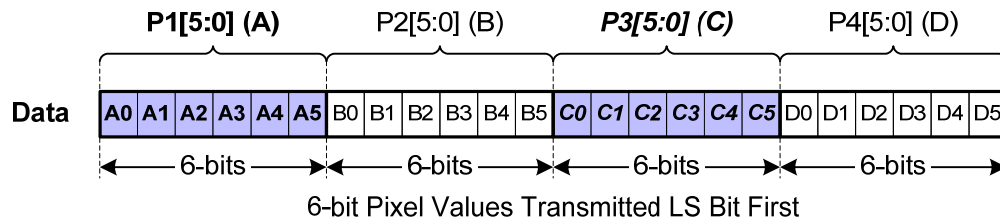
1445 **Figure 89 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration**

1446 6-bit Pixel Value
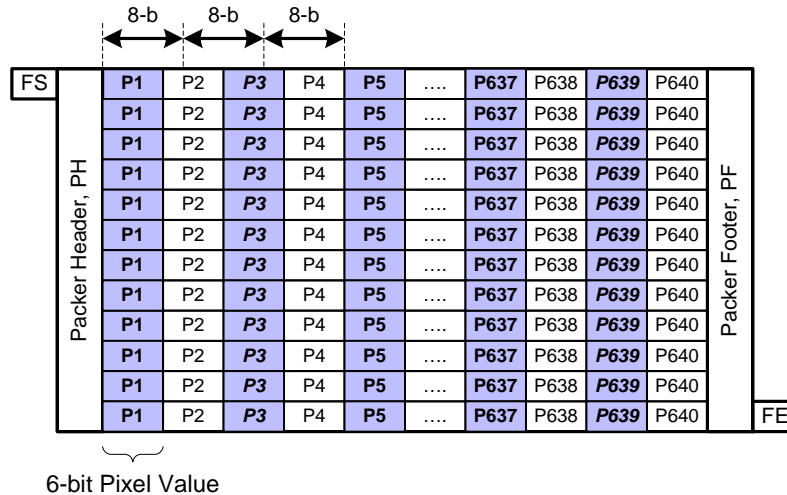
1447 **Figure 90 RAW6 Frame Format**

1448 ## 11.4.2   RAW7

1449 The 7-bit Raw data transmission is performed by transmitting the pixel data over CSI-2 bus. Each line is
1450 separated by line start / end synchronization codes. This sequence is illustrated in Figure 91 (VGA case).

1451 Table 22 specifies the packet size constraints for RAW7 packets. The length of each packet must be a
1452 multiple of the values in the table.

1453 **Table 22 RAW7 Packet Data Size Constraints**

Pixels	Bytes	Bits
8	7	56

1454 Each 7-bit pixel is sent LSB first. This is an exception to general CSI-2 rule byte-wise LSB first.

1455
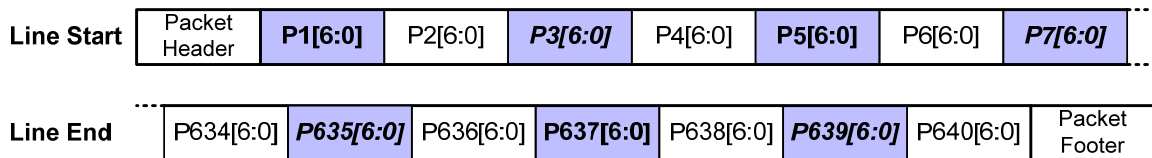
1456 **Figure 91 RAW7 Transmission**

1457
1458
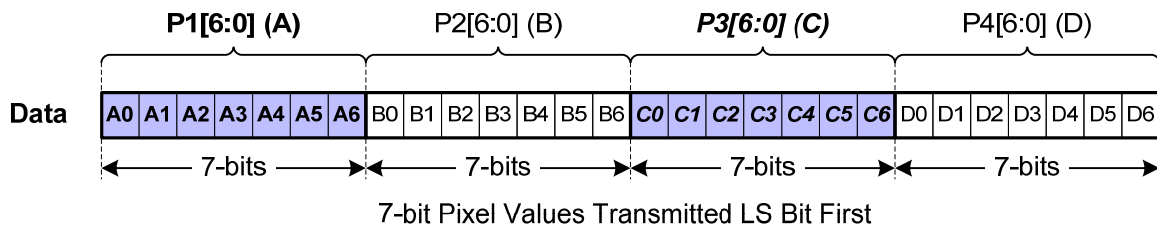
1459 **Figure 92 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration**

1460
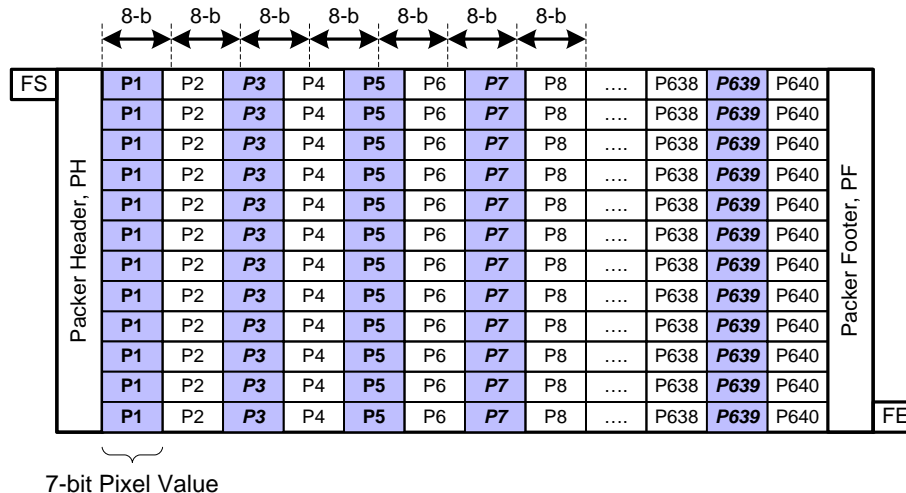
1461 **Figure 93 RAW7 Frame Format**

1462 ### 11.4.3   RAW8

1463 The 8-bit Raw data transmission is performed by transmitting the pixel data over a CSI-2 bus. Table 23
1464 specifies the packet size constraints for RAW8 packets. The length of each packet must be a multiple of the
1465 values in the table.

1466   **Table 23 RAW8 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	1	8

1467   This sequence is illustrated in Figure 94 (VGA case).

1468   Bit order in transmission follows the general CSI-2 rule, LSB first.

1469
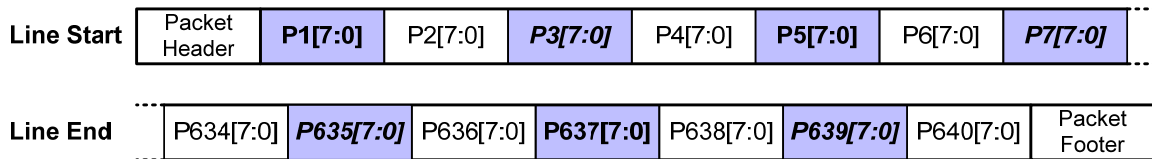
1470   **Figure 94 RAW8 Transmission**

1471
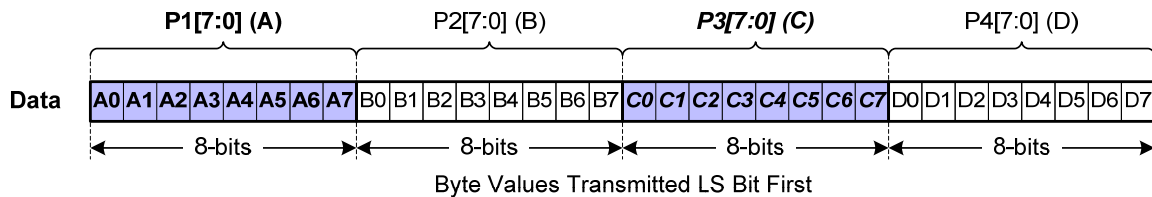
1472   **Figure 95 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration**

1473
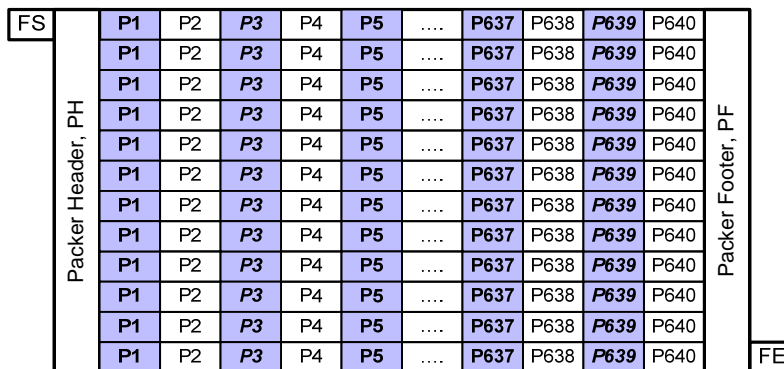
1474   **Figure 96 RAW8 Frame Format**

1475   ## 11.4.4   RAW10

1476   The transmission of 10-bit Raw data is accomplished by packing the 10-bit pixel data to look like 8-bit data
1477   format. Table 24 specifies the packet size constraints for RAW10 packets. The length of each packet must
1478   be a multiple of the values in the table.

1479   **Table 24 RAW10 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	5	40

1480   This sequence is illustrated in Figure 97 (VGA case).

1481   Bit order in transmission follows the general CSI-2 rule, LSB first.

Figure 97 RAW10 Transmission

Byte Values Transmitted LS Bit First
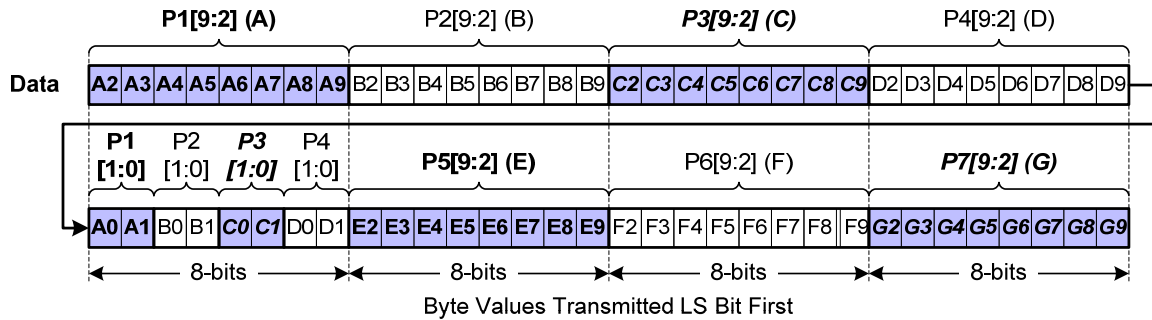
Figure 98 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration
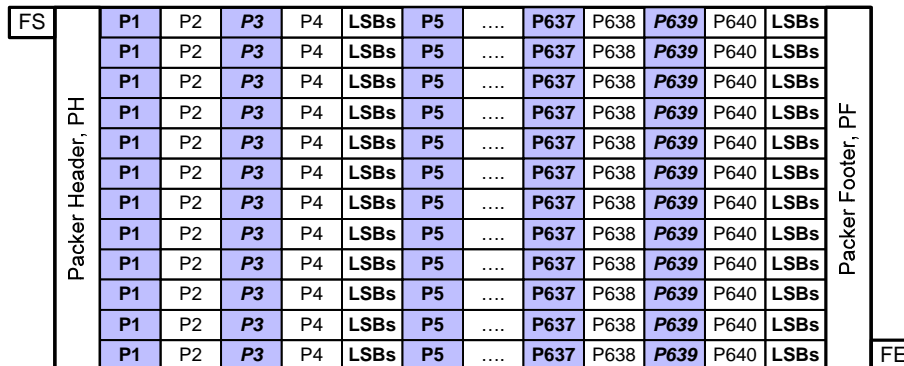
Figure 99 RAW10 Frame Format

11.4.5   RAW12

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

Table 25 RAW12 Packet Data Size Constraints

Pixels	Bytes	Bits
2	3	24

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

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

1497
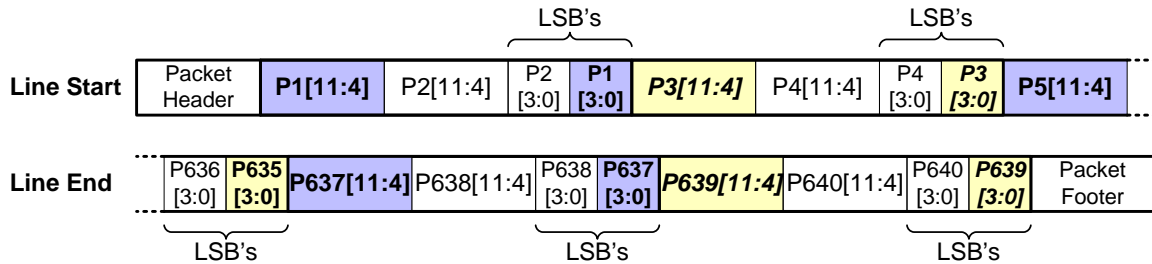
1498   **Figure 100 RAW12 Transmission**

1499
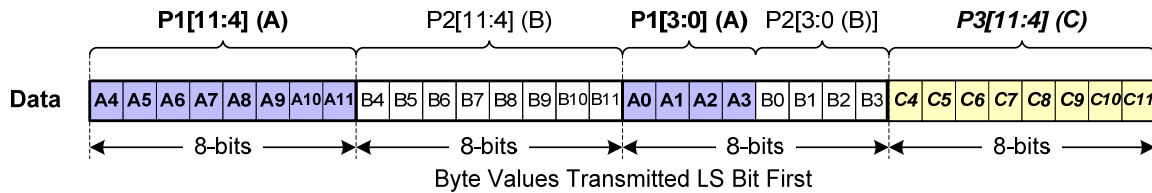
1500   **Figure 101 RAW12 Transmission on CSI-2 Bus Bitwise Illustration**

1501
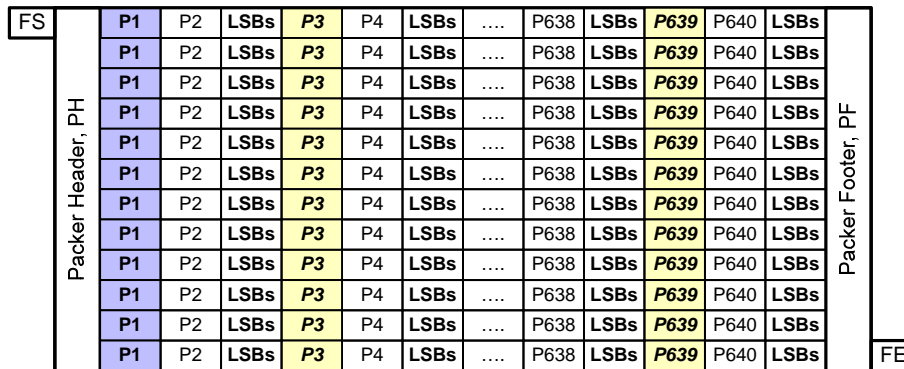
1502   **Figure 102 RAW12 Frame Format**

1503   **11.4.6   RAW14**

1504   The transmission of 14-bit Raw data is accomplished by packing the 14-bit pixel data in 8-bit slices. For
1505   every four pixels, seven bytes of data is generated. Table 26 specifies the packet size constraints for
1506   RAW14 packets. The length of each packet must be a multiple of the values in the table.

1507   **Table 26 RAW14 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	7	56

1508   The sequence is illustrated in Figure 103 (VGA case).

1509   The LS bits for P1, P2, P3 and P4 are distributed in three bytes as shown in Figure 104. The same is true
1510   for the LS bits for P637, P638, P639 and P640. The bit order during transmission follows the general CSI-2
1511   rule, i.e. LSB first.

1512
1513
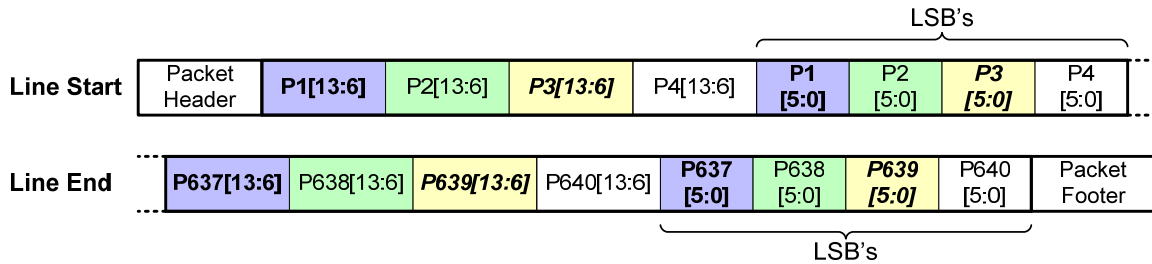
Figure 103 RAW14 Transmission

1515
1516
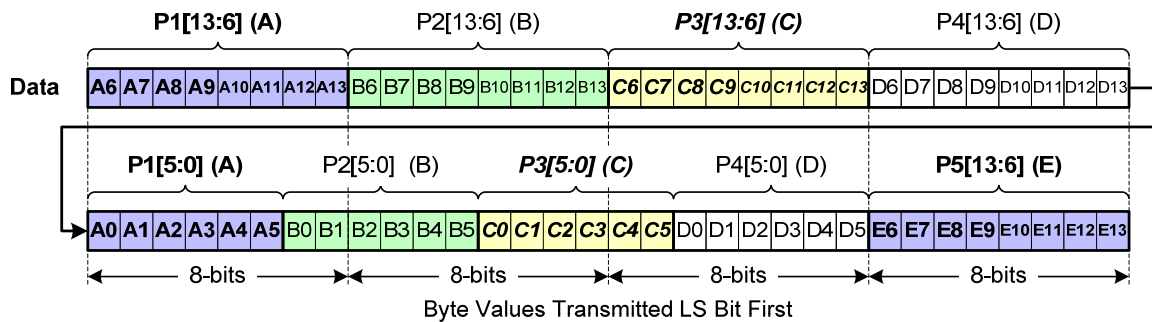
Figure 104 RAW14 Transmission on CSI-2 Bus Bitwise Illustration

1518
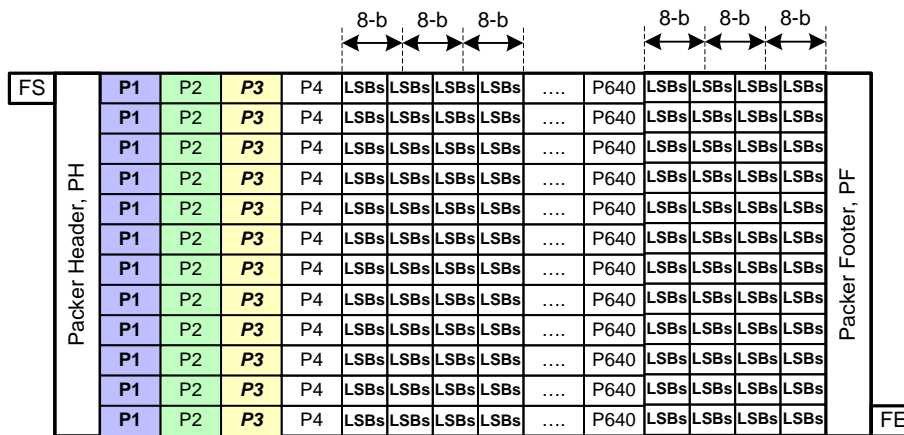
Figure 105 RAW14 Frame Format

1520 ## 11.5   User Defined Data Formats

1521 The User Defined Data Type values shall be used to transmit arbitrary data, such as JPEG and MPEG4
1522 data, over the CSI-2 bus. Data shall be packed so that the data length is divisible by eight bits. If data
1523 padding is required, the padding shall be added before data is presented to the CSI-2 protocol interface.

1524 Bit order in transmission follows the general CSI-2 rule, LSB first.

Line Start	Packet Header	**B1[7:0]**	B2[7:0]	***B3[7:0]***	B4[7:0]	**B5[7:0]**	B6[7:0]	***B7[7:0]***

Line End	**B121[7:0]**	B122[7:0]	***B123[7:0]***	B124[7:0]	**B125[7:0]**	B126[7:0]	***B127[7:0]***	Packet Footer
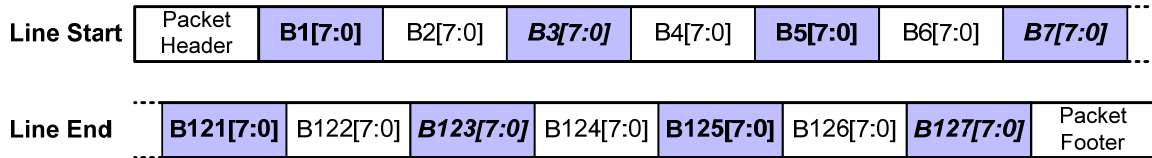
Figure 106 User Defined 8-bit Data (128 Byte Packet)
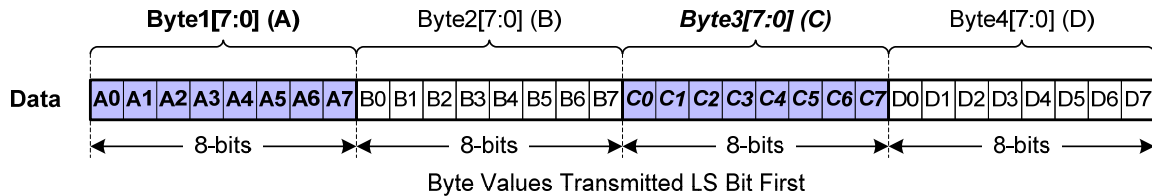
Figure 107 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration

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

For User Defined data:

- The frame is transmitted as a sequence of arbitrary sized packets.

- The packet size may vary from packet to packet.

- The packet spacing may vary between packets.

KEY:
SoT – Start of Transmission   EoT – End of Transmission   LPS – Low Power State
PH – Packet Header   PF – Packet Footer
FS – Frame Start   FE – Frame End
                                   LE – Line End
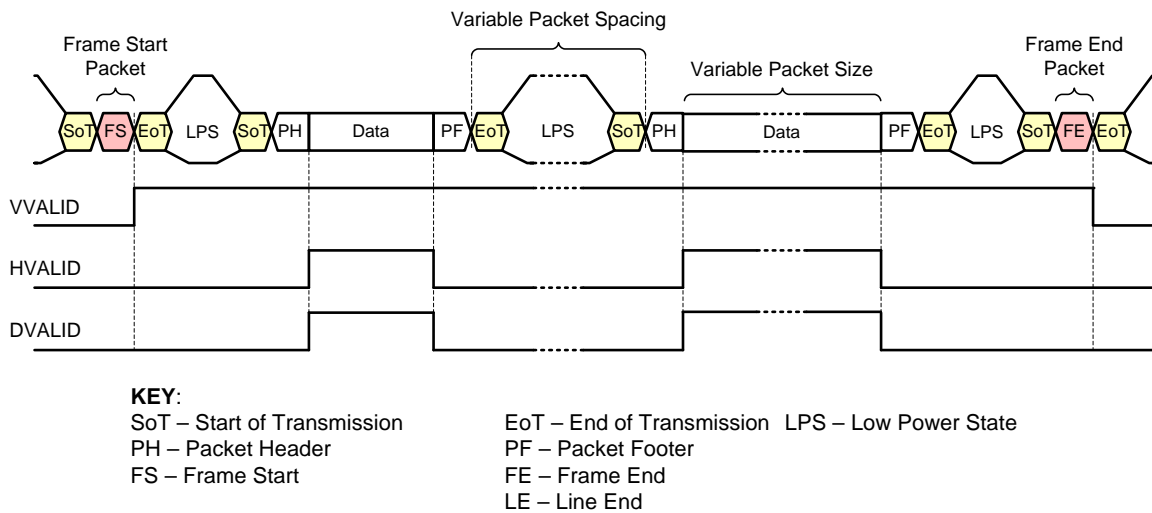
Figure 108 Transmission of User Defined 8-bit Data

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

Table 27 User Defined 8-bit Data Types

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

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

1540
1541

12   Recommended Memory Storage

1542

1543   This section is informative.

1544   The CSI-2 data protocol requires certain behavior from the receiver connected to the CSI transmitter. The
1545   following sections describe how different data formats should be stored inside the receiver. While
1546   informative, this section is provided to ease application software development by suggesting a common
1547   data storage format among different receivers.

12.1   General/Arbitrary Data Reception

1548

1549   In the generic case and for arbitrary data the first byte of payload data transmitted maps the LS byte of the
1550   32-bit memory word and the fourth byte of payload data transmitted maps to the MS byte of the 32-bit
1551   memory word.

1552   The below is the generic CSI-2 byte to 32-bit memory word mapping rule.

1553
1554
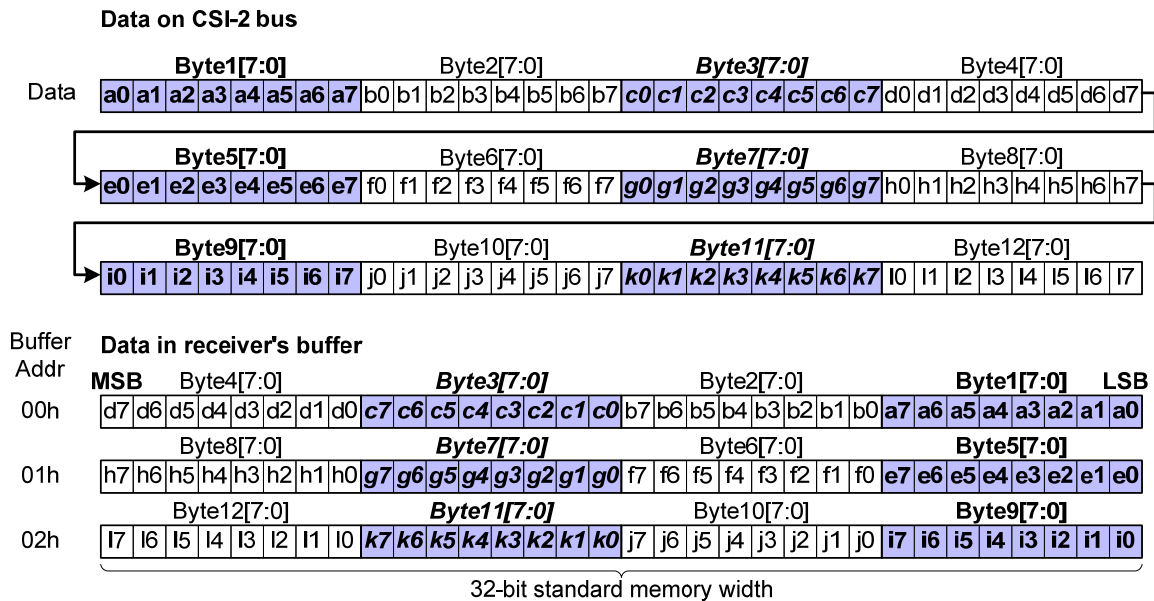
1555   **Figure 109 General/Arbitrary Data Reception**

12.2   RGB888 Data Reception

1556

1557   The RGB888 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.

Data on CSI-2 bus

Data in receiver's buffer
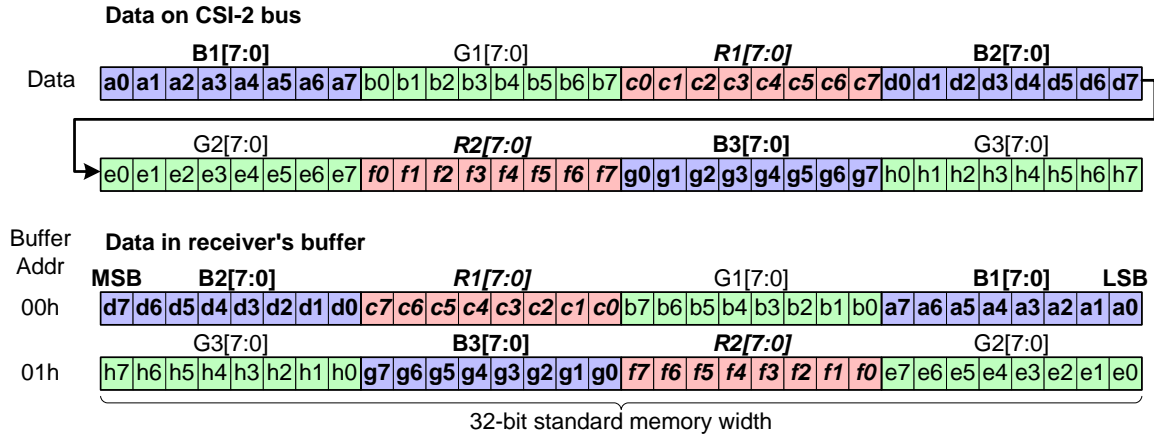
Figure 110 RGB888 Data Format Reception

12.3   RGB666 Data Reception

Data on CSI-2 bus

Data in receiver's buffer
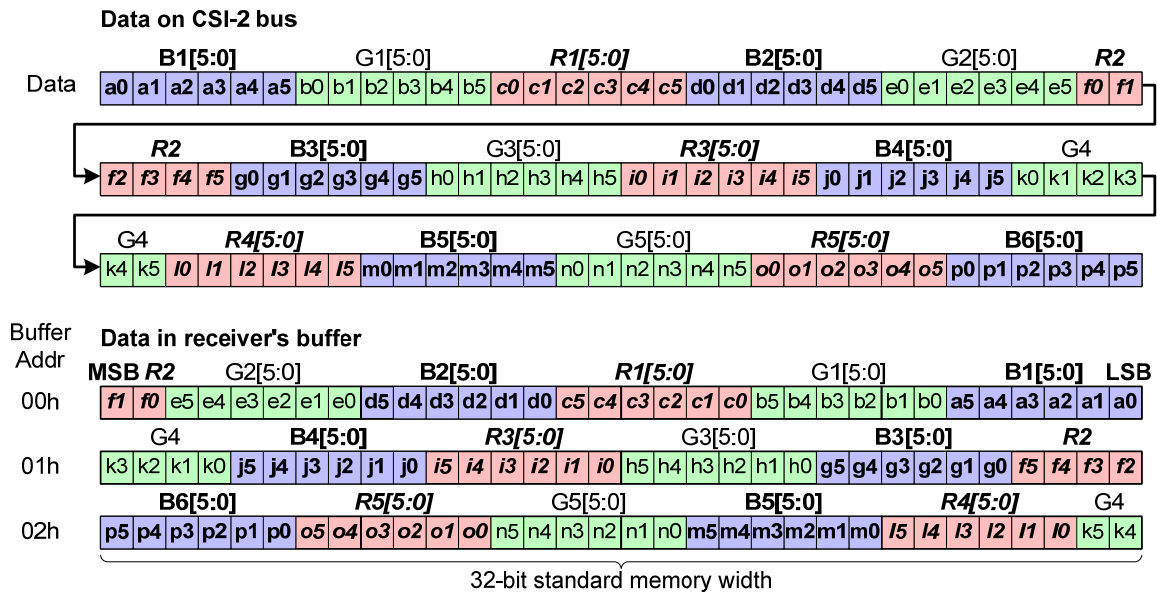
Figure 111 RGB666 Data Format Reception

1563 ## 12.4 RGB565 Data Reception
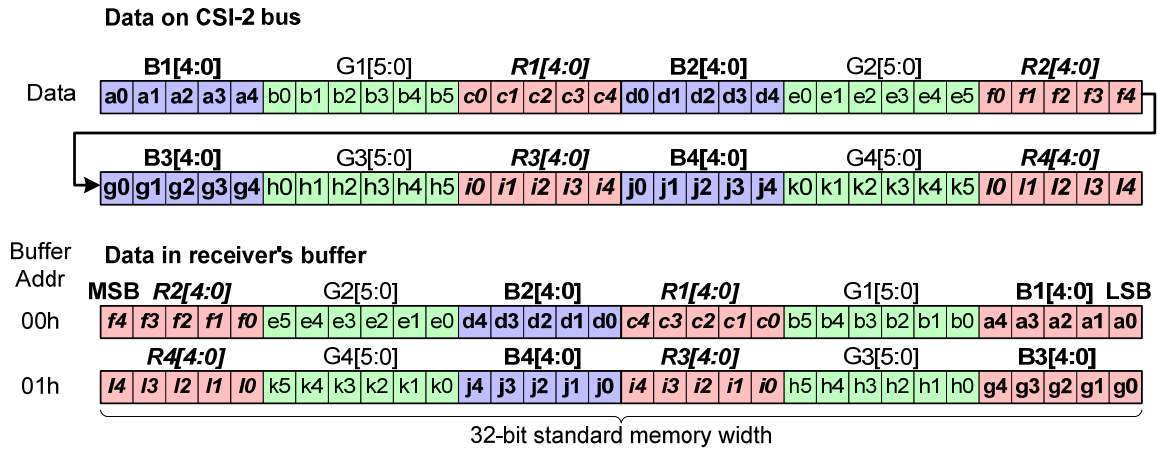
1564
1565 **Figure 112 RGB565 Data Format Reception**

1566 ## 12.5 RGB555 Data Reception
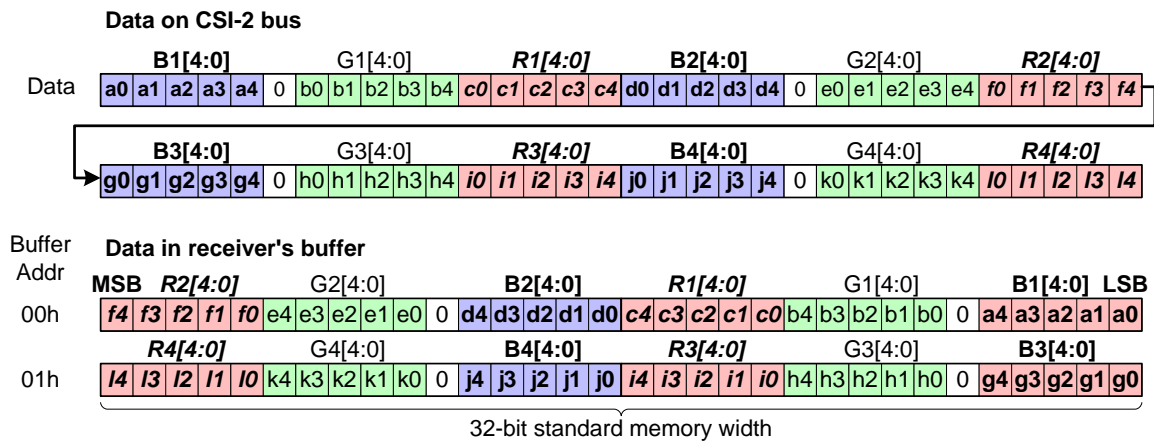
1567
1568 **Figure 113 RGB555 Data Format Reception**

1569 ## 12.6 RGB444 Data Reception

1570 The RGB444 data format byte to 32-bit memory word mapping has a special transform as shown in Figure
1571 114.
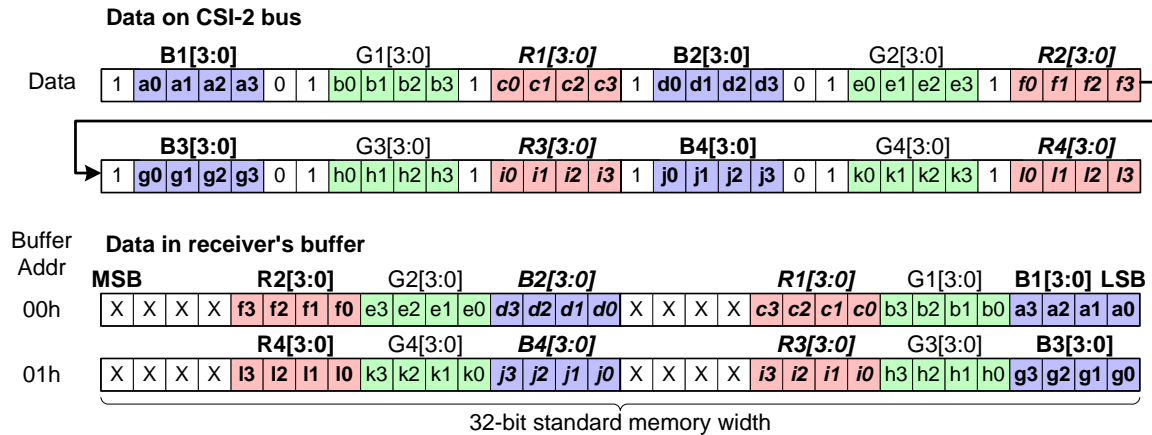
Figure 114 RGB444 Data Format Reception

12.7 YUV422 8-bit Data Reception

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

For YUV422 8-bit data format the first byte of payload data transmitted maps the MS byte of the 32-bit memory word and the fourth byte of payload data transmitted maps to the LS byte of the 32-bit memory word.
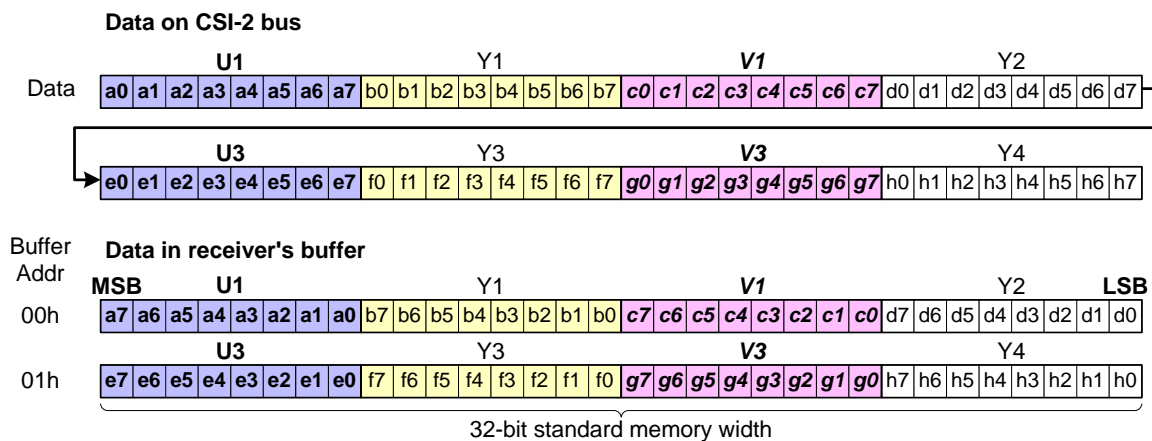
Figure 115 YUV422 8-bit Data Format Reception

12.8 YUV422 10-bit Data Reception

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

Data on CSI-2 bus

1584
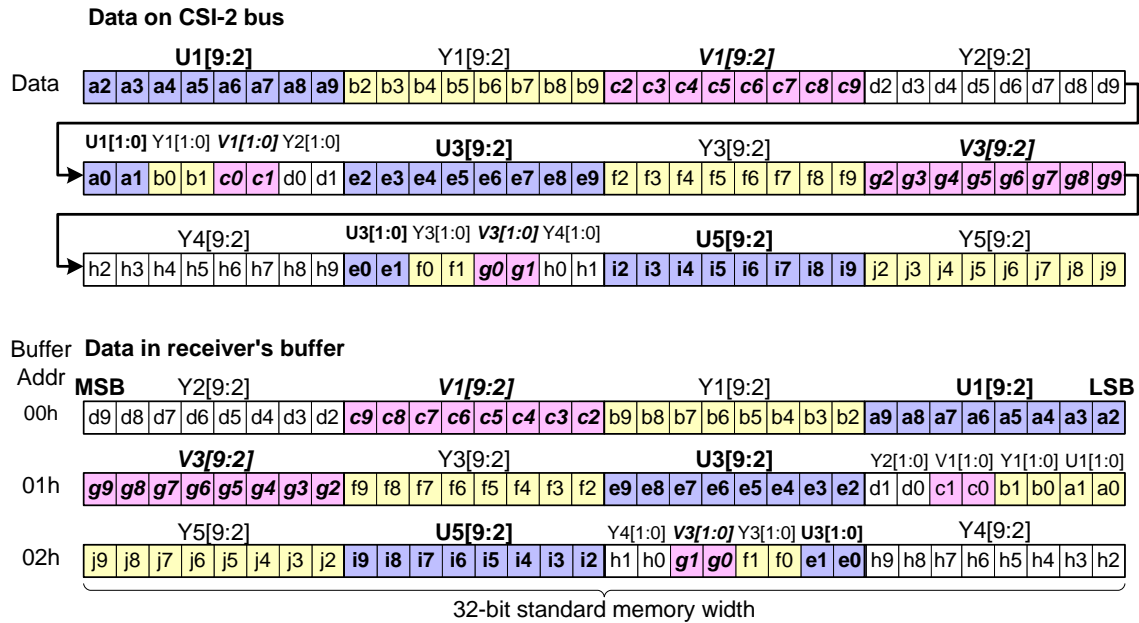
1585   **Figure 116 YUV422 10-bit Data Format Reception**

12.9   YUV420 8-bit (Legacy) Data Reception

1587   The YUV420 8-bit (legacy) data format the byte to 32-bit memory word mapping does not follow the
1588   generic CSI-2 rule.

1589   For YUV422 8-bit (legacy) data format the first byte of payload data transmitted maps the MS byte of the
1590   32-bit memory word and the fourth byte of payload data transmitted maps to the LS byte of the 32-bit
1591   memory word.

Data on CSI-2 bus (Odd Line)

Data on CSI-2 bus (Even Line)

Figure 117 YUV420 8-bit Legacy Data Format Reception

12.10 YUV420 8-bit Data Reception

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

Data on CSI-2 bus (Odd Line)

	Y1								Y2								Y3								Y4							
Data	a0	a1	a2	a3	a4	a5	a6	a7	b0	b1	b2	b3	b4	b5	b6	b7	c0	c1	c2	c3	c4	c5	c6	c7	d0	d1	d2	d3	d4	d5	d6	d7

	Y5								Y6								Y7								Y8							
	e0	e1	e2	e3	e4	e5	e6	e7	f0	f1	f2	f3	f4	f5	f6	f7	g0	g1	g2	g3	g4	g5	g6	g7	h0	h1	h2	h3	h4	h5	h6	h7

Buffer Addr   **Data in receiver's buffer**

	MSB	Y4							Y3								Y2								Y1							LSB
00h	d7	d6	d5	d4	d3	d2	d1	d0	c7	c6	c5	c4	c3	c2	c1	c0	b7	b6	b5	b4	b3	b2	b1	b0	a7	a6	a5	a4	a3	a2	a1	a0

		Y8							Y7								Y6								Y5							
01h	h7	h6	h5	h4	h3	h2	h1	h0	g7	g6	g5	g4	g3	g2	g1	g0	f7	f6	f5	f4	f3	f2	f1	f0	e7	e6	e5	e4	e3	e2	e1	e0

32-bit standard memory width

Data on CSI-2 bus (Even Line)

	U1								Y1								V1								Y2							
Data	a0	a1	a2	a3	a4	a5	a6	a7	b0	b1	b2	b3	b4	b5	b6	b7	c0	c1	c2	c3	c4	c5	c6	c7	d0	d1	d2	d3	d4	d5	d6	d7

	U3								Y3								V3								Y4							
	e0	e1	e2	e3	e4	e5	e6	e7	f0	f1	f2	f3	f4	f5	f6	f7	g0	g1	g2	g3	g4	g5	g6	g7	h0	h1	h2	h3	h4	h5	h6	h7

Buffer Addr   **Data in receiver's buffer**

	MSB	Y2							V1								Y1								U1							LSB
N	d7	d6	d5	d4	d3	d2	d1	d0	c7	c6	c5	c4	c3	c2	c1	c0	b7	b6	b5	b4	b3	b2	b1	b0	a7	a6	a5	a4	a3	a2	a1	a0

		Y4							V3								Y3								U3							
N+1	h7	h6	h5	h4	h3	h2	h1	h0	g7	g6	g5	g4	g3	g2	g1	g0	f7	f6	f5	f4	f3	f2	f1	f0	e7	e6	e5	e4	e3	e2	e1	e0

32-bit standard memory width

Figure 118 YUV420 8-bit Data Format Reception

12.11 YUV420 10-bit Data Reception

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

Data on CSI-2 bus (Odd Line)

Data in receiver's buffer

32-bit standard memory width

Data on CSI-2 bus (Even Line)

Data in receiver's buffer

32-bit standard memory width

1600

1601
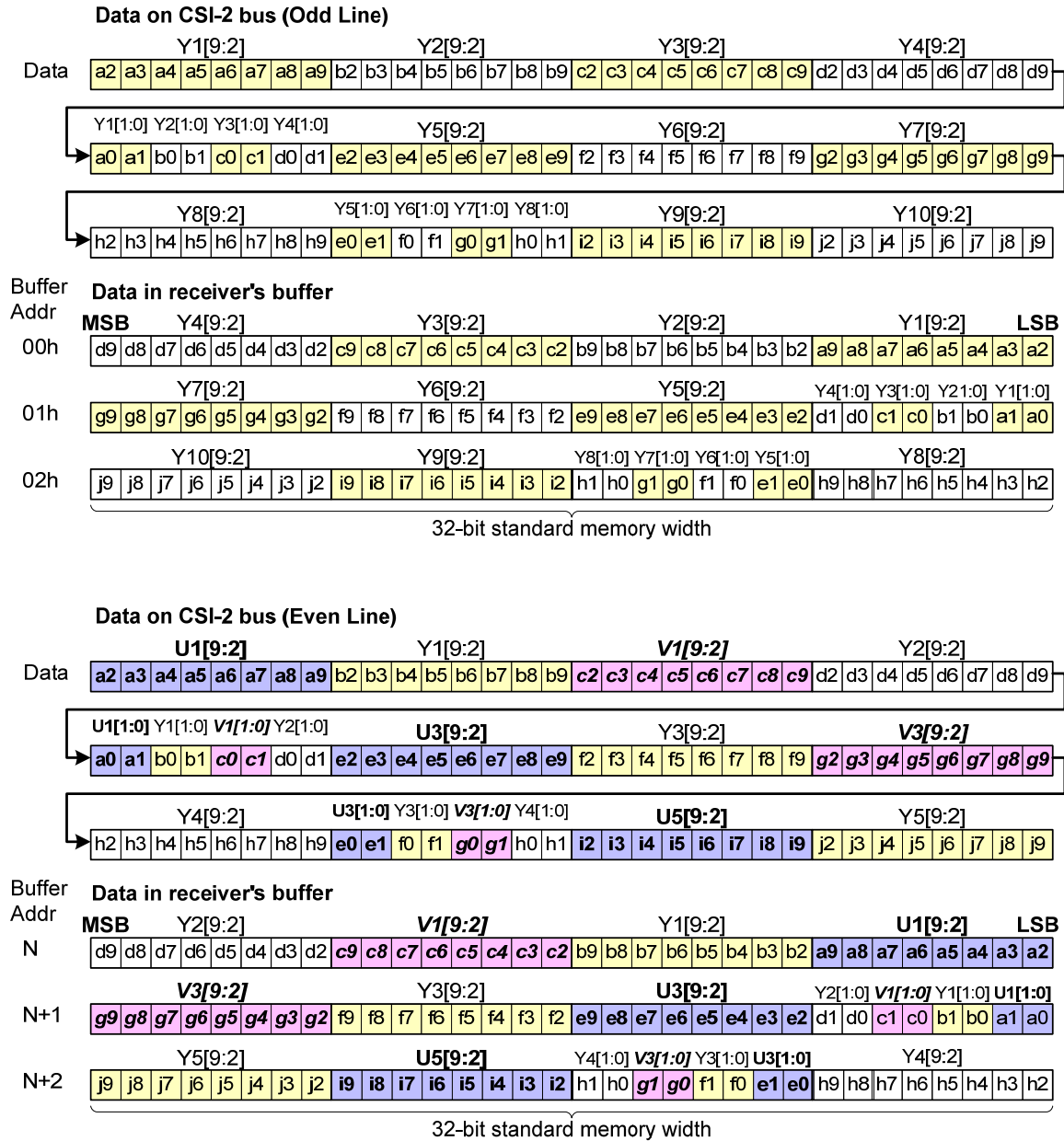
Figure 119 YUV420 10-bit Data Format Reception

1602

12.12 RAW6 Data Reception
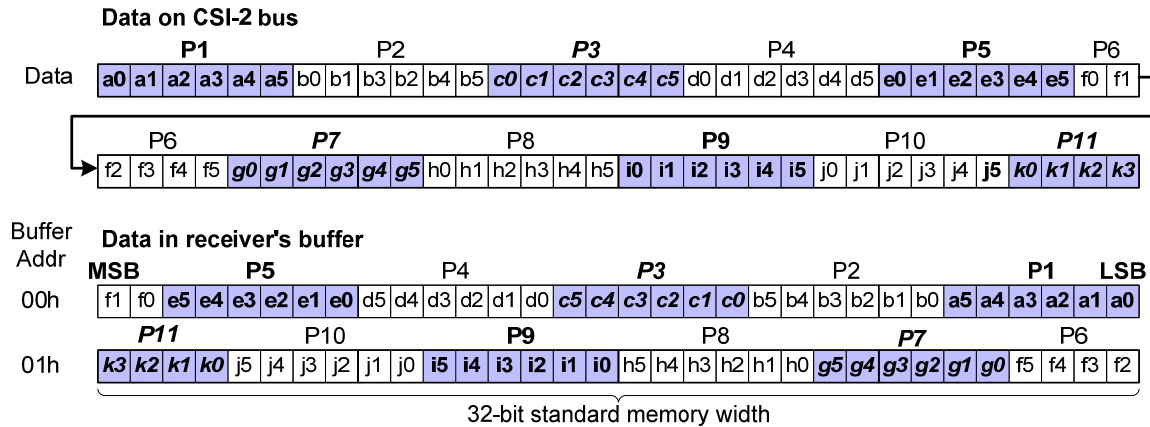
Figure 120 RAW6 Data Format Reception

1603

1604

1605

12.13 RAW7 Data Reception
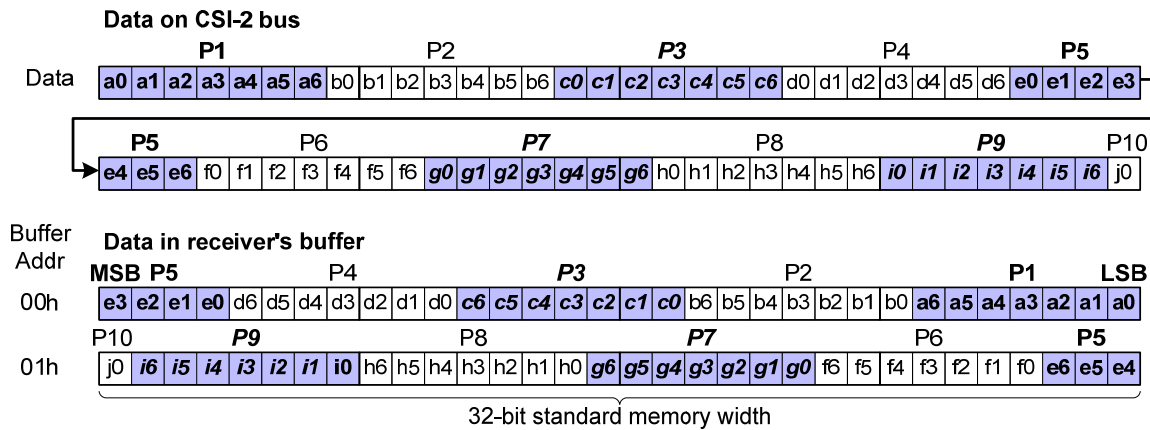
Figure 121 RAW7 Data Format Reception

1606

1607

1608

12.14 RAW8 Data Reception

1609　The RAW8 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

Data on CSI-2 bus

1610
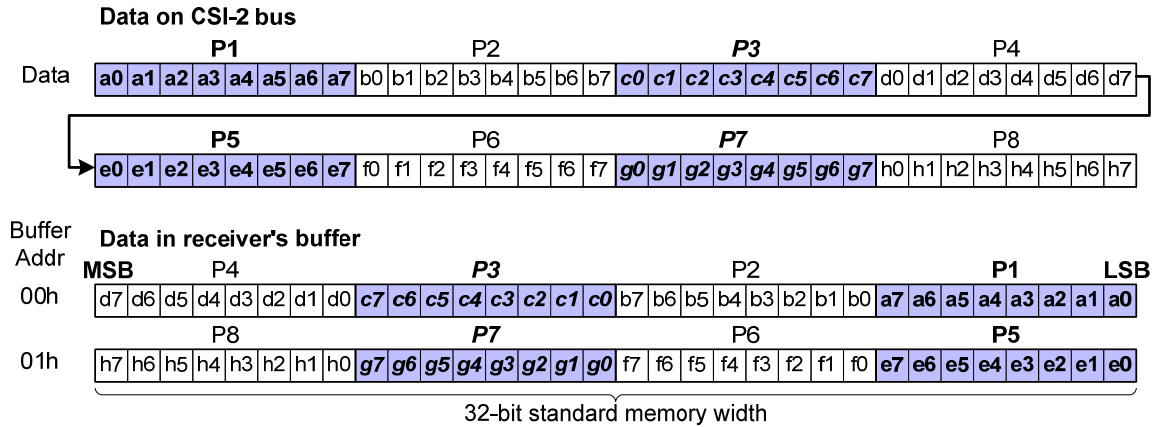
Figure 122 RAW8 Data Format Reception

12.15 RAW10 Data Reception

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

1614
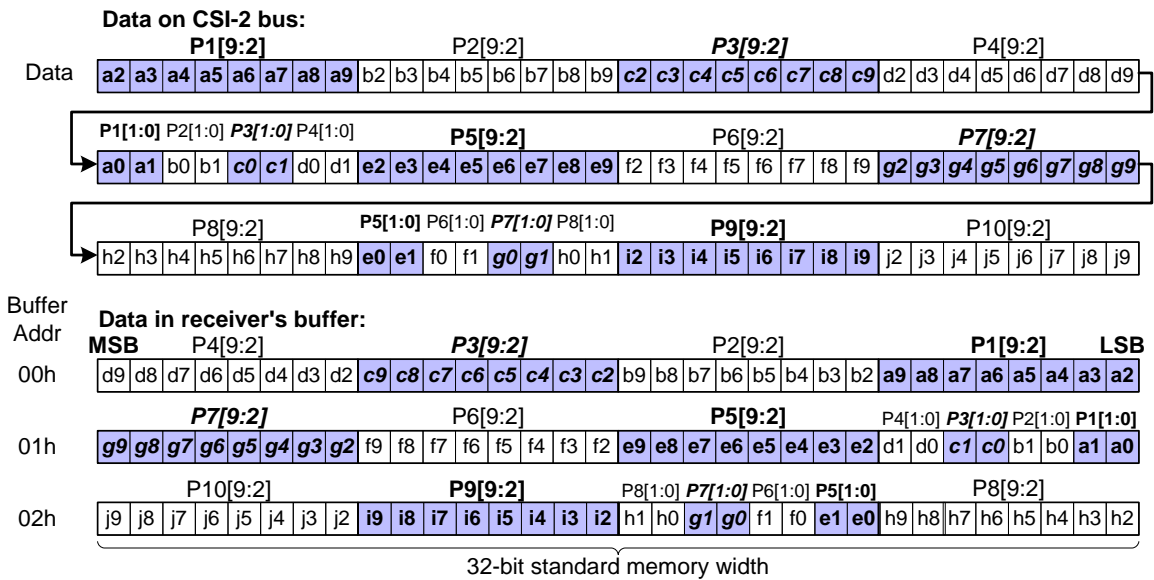
Figure 123 RAW10 Data Format Reception

12.16 RAW12 Data Reception

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

1618
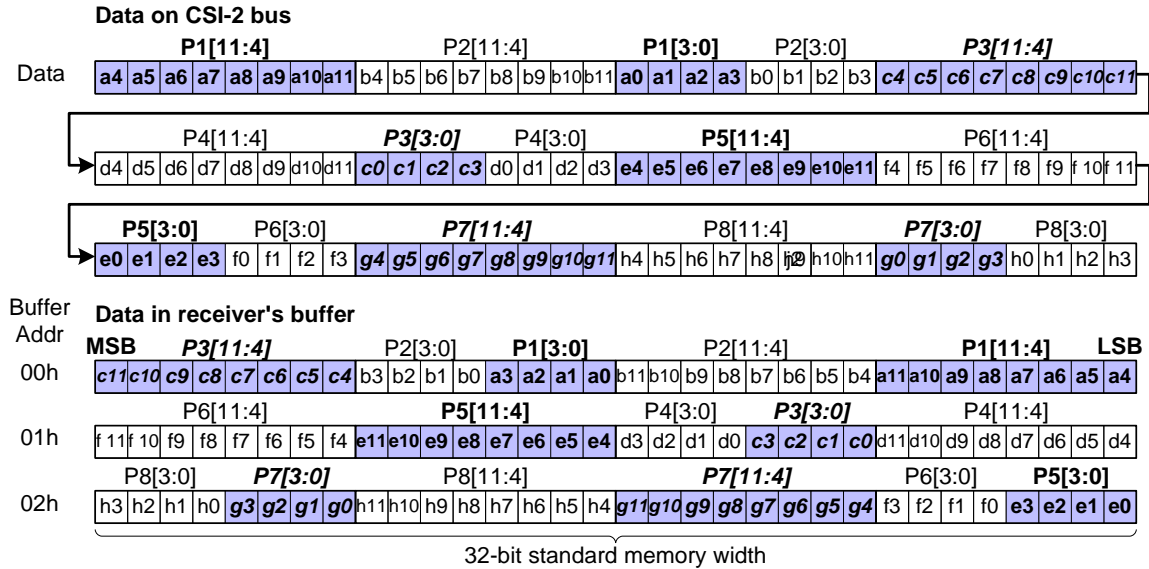
Figure 124 RAW12 Data Format Reception

1619

1620   ## 12.17 RAW14 Data Reception

1621
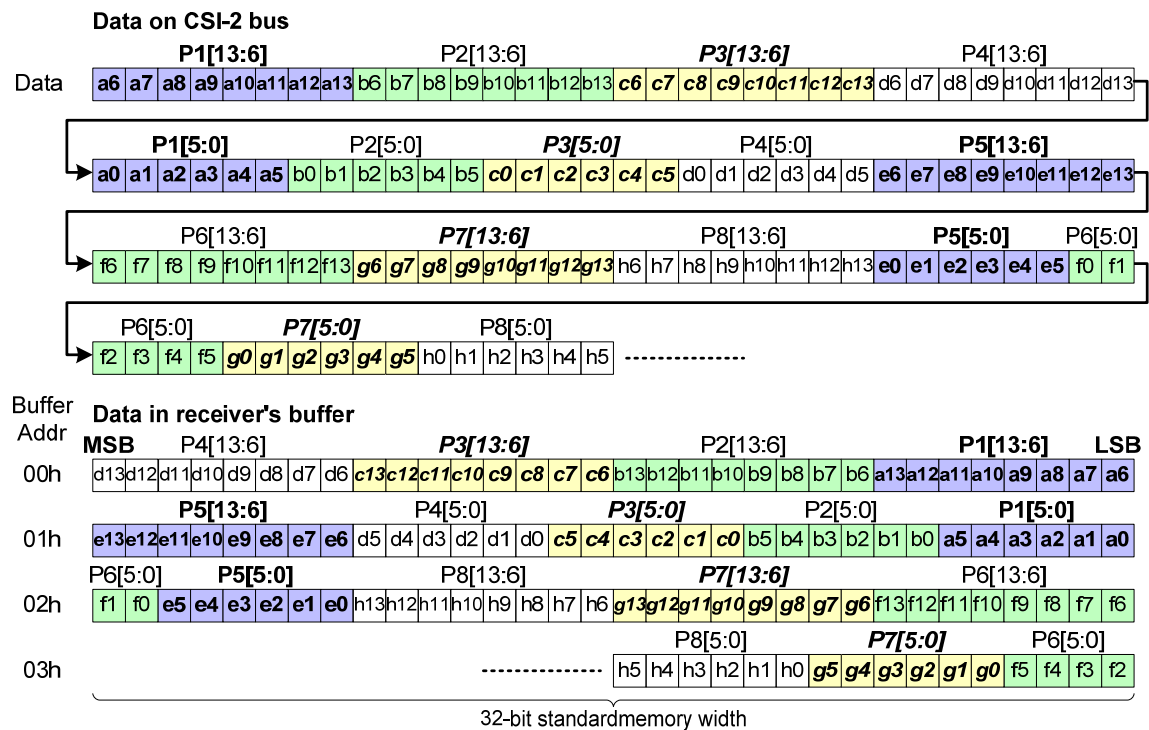
1622   **Figure 125 RAW 14 Data Format Reception**

1623

1624 Annex A JPEG8 Data Format (informative)

A.1   Introduction

1625

1626 This Annex contains an informative example of the transmission of compressed image data format using
1627 the arbitrary Data Type values.

1628 JPEG8 has two non-standard extensions:

1629   • Status information (mandatory)

1630   • Embedded Image information e.g. a thumbnail image (optional)

1631 Any non-standard or additional data inside the baseline JPEG data structure has to be removed from JPEG8
1632 data before it is compliant with e.g. standard JPEG image viewers in e.g. a personal computer.

1633 The JPEG8 data flow is illustrated in the Figure 126 and Figure 127.

1634
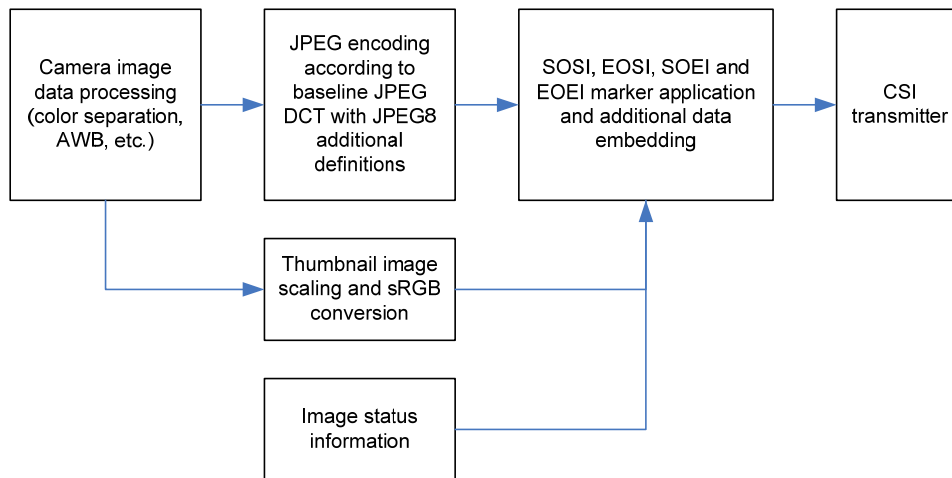
1635   **Figure 126 JPEG8 Data Flow in the Encoder**

1636
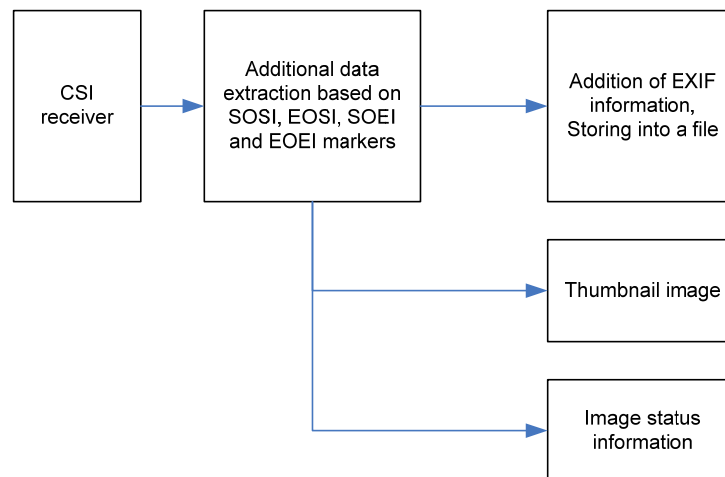
1637   **Figure 127 JPEG8 Data Flow in the Decoder**

1638 ## A.2   JPEG Data Definition

1639 The JPEG data generated in camera module is baseline JPEG DCT format defined in ISO/IEC 10918-1,
1640 with following additional definitions or modifications:

1641 • sRGB color space shall be used. The JPEG is generated from YcbCr format after sRGB to YcbCr
1642   conversion.

1643 • The JPEG metadata has to be EXIF compatible, i.e. metadata within application segments has to
1644   be placed in beginning of file, in the order illustrated in Figure 128.

1645 • A status line is added in the end of JPEG data as defined in section A.3.

1646 • If needed, an embedded image is interlaced in order which is free of choice as defined in section
1647   A.4.

1648 • Prior to storing into a file, the CSI-2 JPEG data is processed by the data separation process
1649   described in section A.1.

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

1650
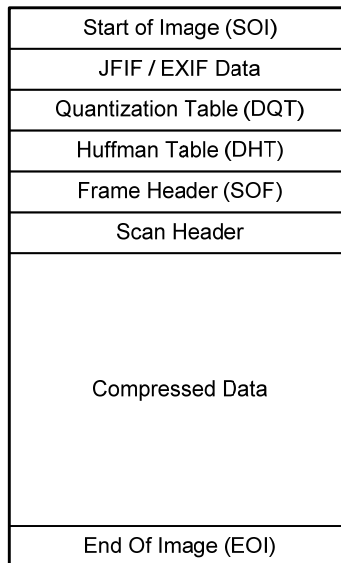
1651 **Figure 128 EXIF Compatible Baseline JPEG DCT Format**

1652 ## A.3   Image Status Information

1653 Information of at least the following items has to be stored in the end of the JPEG sequence as illustrated in
1654 Figure 129:

1655 • Image exposure time

1656 • Analog & digital gains used

1657 • White balancing gains for each color component

1658 • Camera version number

1659 • Camera register settings

1660 • Image resolution and possible thumbnail resolution

1661   The camera register settings may include a subset of camera's registers. The essential information needed
1662   for JPEG8 image is the information needed for converting the image back to linear space. This is necessary
1663   e.g. for printing service. An example of register settings is following:

1664   • Sample frequency

1665   • Exposure

1666   • Analog and digital gain

1667   • Gamma

1668   • Color gamut conversion matrix

1669   • Contrast

1670   • Brightness

1671   • Pre-gain

1672   The status information content has to be defined in the product specification of each camera module
1673   containing the JPEG8 feature. The format and content is manufacturer specific.

1674   The image status data should be arranged so that each byte is split into two 4-bit nibbles and "1010"
1675   padding sequence is added to MSB, as presented in the Table 28. This ensures that no JPEG escape
1676   sequences (0xFF 0x00) are present in the status data.

1677   The SOSI and EOSI markers are defined in 14.5.

1678   **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)

1679
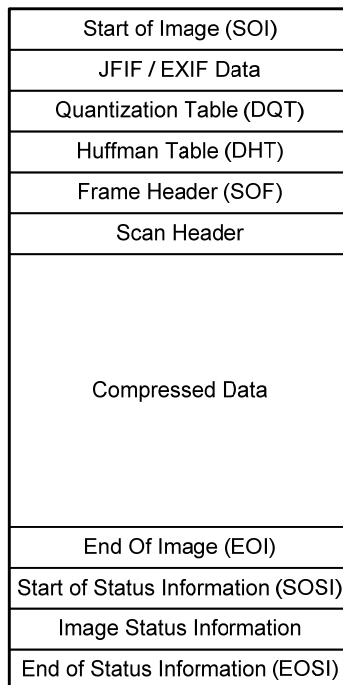
1680   **Figure 129 Status Information Field in the End of Baseline JPEG Frame**

1681   ## A.4   Embedded Images

1682   An image may be embedded inside the JPEG data, if needed. The embedded image feature is not
1683   compulsory for each camera module containing the JPEG8 feature. An example of embedded data is a 24-
1684   bit RGB thumbnail image.

1685   The philosophy of embedded / interleaved thumbnail additions is to minimize the needed frame memory.
1686   The EI (Embedded Image) data can be included in any part of the compressed image data segment and in as
1687   many pieces as needed. See Figure 130.

1688   Embedded Image data is separated from compressed data by SOEI (Start Of Embedded Image) and EOEI
1689   (End Of Embedded Image) non-standard markers, which are defined in 14.5. The amount of fields
1690   separated by SOEI and EOEI is not limited.

1691   The pixel to byte packing for image data within an EI data field should be as specified for the equivalent
1692   CSI-2 data format. However there is an additional restriction; the embedded image data must not generate
1693   any false JPEG marker sequences (0xFFXX).

1694   The suggested method of preventing false JPEG marker codes from occurring within the embedded image
1695   data it to limit the data range for the pixel values. For example

1696   - For RGB888 data the suggested way to solve the false synchronization code issue is to constrain
1697   the numerical range of R, G and B values from 1 to 254.

1698   - For RGB565 data the suggested way to solve the false synchronization code issue is to constrain
1699   the numerical range of G component from 1-62 and R component from 1-30.

1700   Each EI data field is separated by the SOEI / EOEI markers, has to contain an equal amount bytes and a
1701   complete number of pixels. An EI data field may contain multiple lines or a full frame of image data.

1702   The embedded image data is decoded and removed apart from the JPEG compressed data prior to writing
1703   the JPEG into a file. In the process, EI data fields are appended one after each other, in order of occurrence
1704   in the received JPEG data.

1705
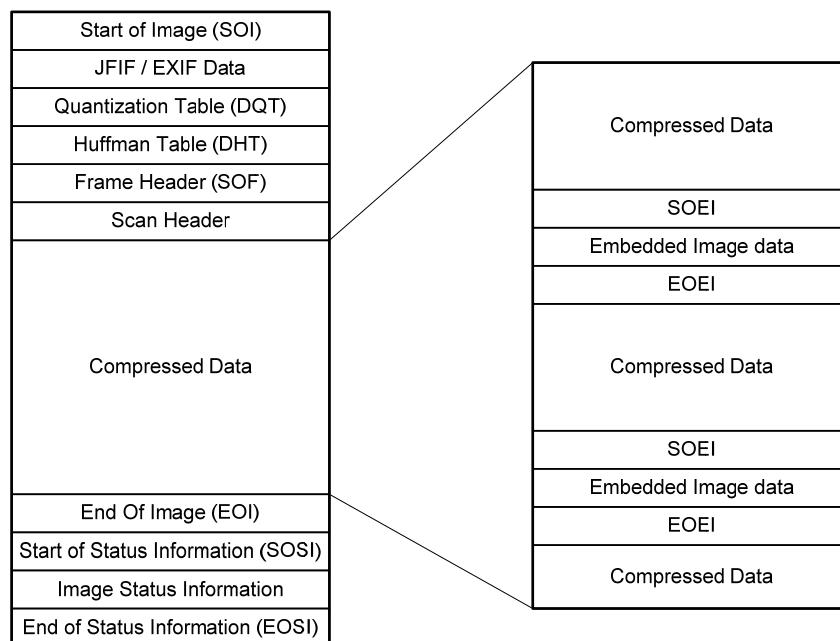
1706   **Figure 130 Example of TN Image Embedding Inside the Compressed JPEG Data Block**

1707   A.5   JPEG8 Non-standard Markers

1708   JPEG8 uses the reserved JPEG data markers for special purposes, marking the additional segments inside
1709   the data file. These segments are not part of the JPEG, JFIF [0], EXIF [0] or any other specifications;
1710   instead their use is specified in this document in sections 14.3 and 14.4.

1711   The use of the non-standard markers is always internal to a product containing the JPEG8 camera module,
1712   and these markers are always removed from the JPEG data before storing it into a file

1713   **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

1714   A.6   JPEG8 Data Reception

1715   The compressed data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.
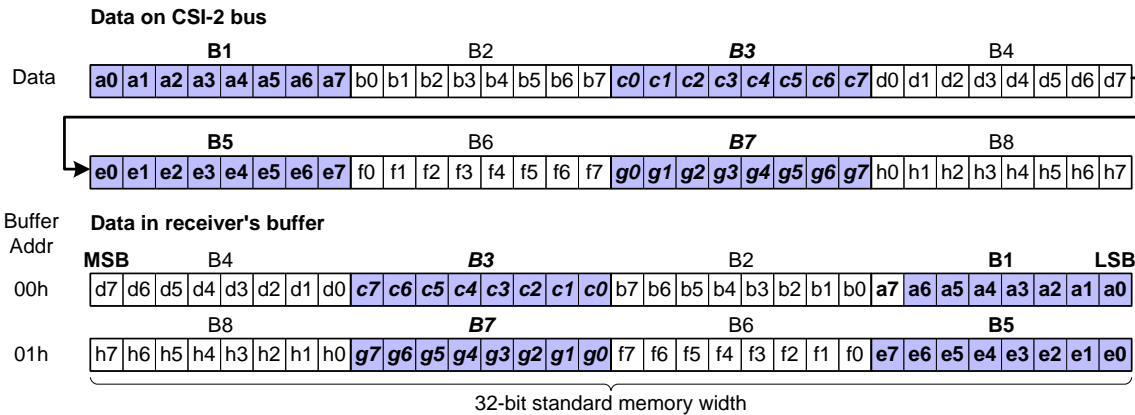
1716
1717   **Figure 131 JPEG8 Data Format Reception**

1718

1719 Annex B CSI-2 Implementation Example
1720 (informative)

1721 B.1   Overview

1722 The CSI-2 implementation example assumes that the interface comprises of D-PHY unidirectional Clock
1723 and Data, with forward escape mode functionality. The scope in this implementation example refers only to
1724 the unidirectional data link without any references to the CCI interface, as it can be seen in Figure 132. This
1725 implementation example varies from the informative PPI example in [MIPI01].

1726
1727
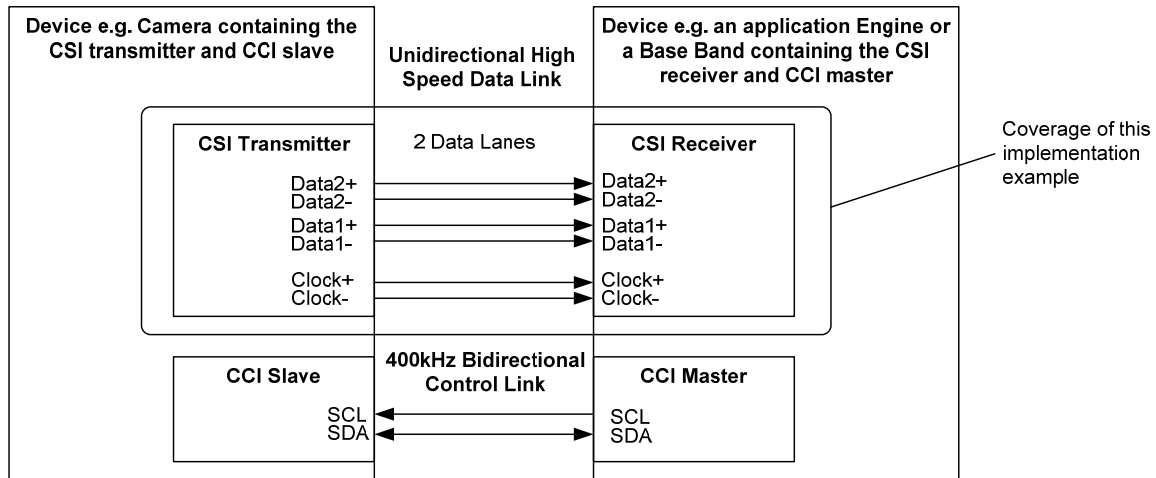
1728 **Figure 132 Implementation Example Block Diagram and Coverage**

1729 For this implementation example a layered structure is described with the following parts:

1730   • D-PHY implementation details

1731   • Multi lane merger details

1732   • Protocol layer details

1733 This implementation example refers to a RAW8 data type only; hence no packing/unpacking or byte
1734 clock/pixel clock timing will be referenced as for this type of implementation they are not needed.

1735 No error recovery mechanism or error processing details will be presented, as the intent of the document is
1736 to present an implementation from the data flow perspective.

1737 B.2   CSI-2 Transmitter Detailed Block Diagram

1738 Using the layered structure described in the overview the CSI-2 transmitter could have the block diagram in
1739 Figure 133.

1740
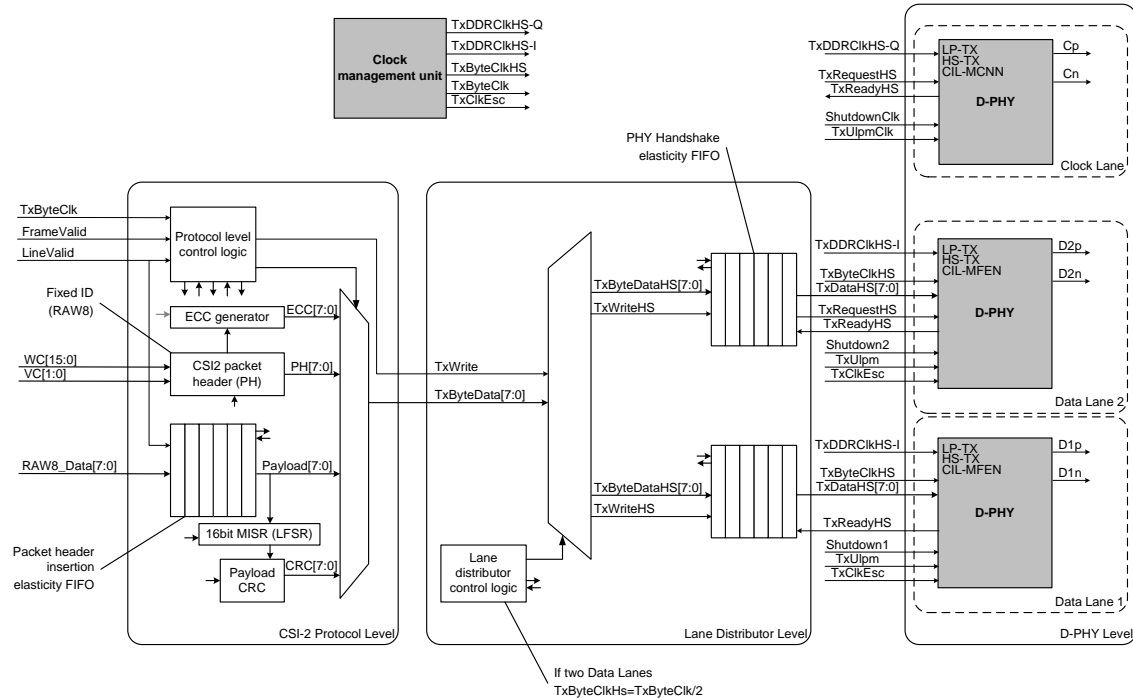
1741 **Figure 133 CSI-2 Transmitter Block Diagram**

1742 ## B.3   CSI-2 Receiver Detailed Block Diagram

1743 Using the layered structure described in the overview, the CSI-2 receiver could have the block diagram in
1744 Figure 134.

1745
1746
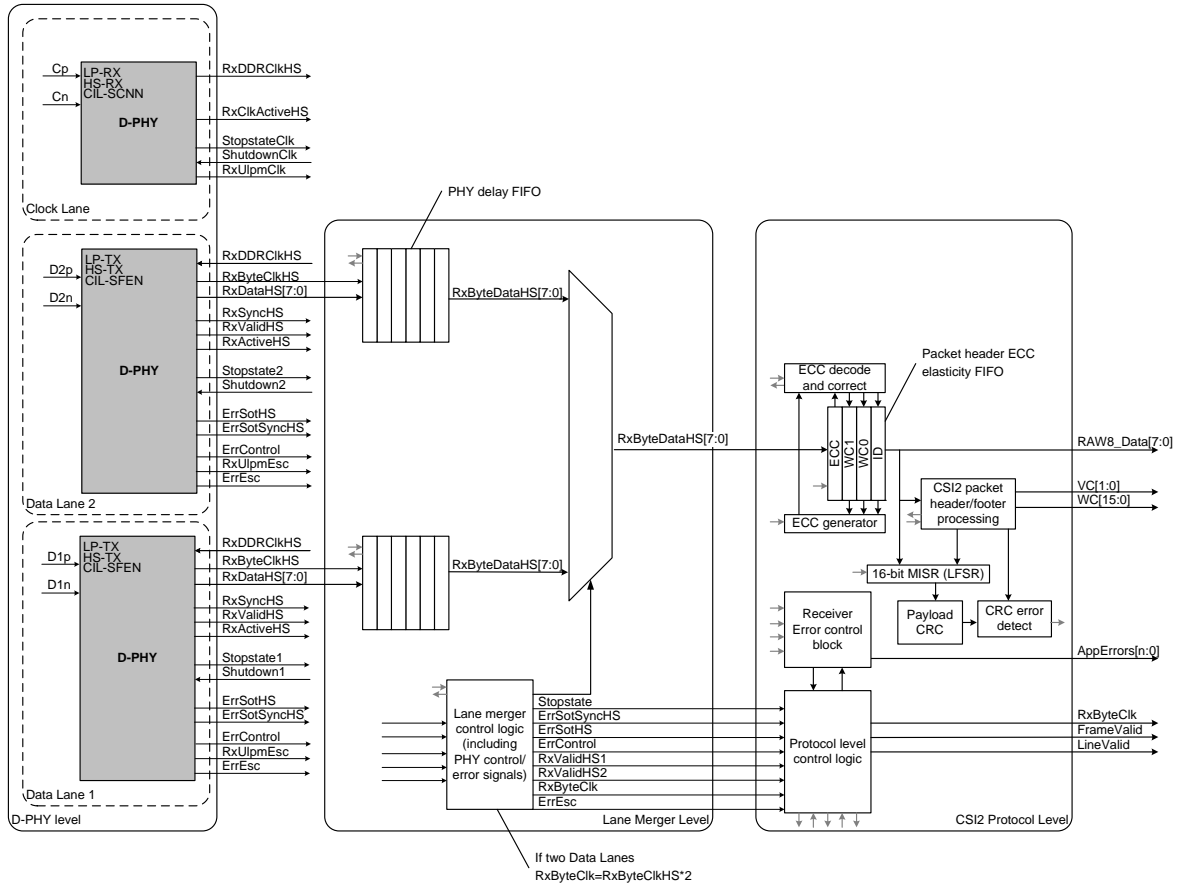
Figure 134 CSI-2 Receiver Block Diagram

B.4   Details on the D-PHY implementation

1749   The PHY level of implementation has the top level structure as seen in Figure 135.
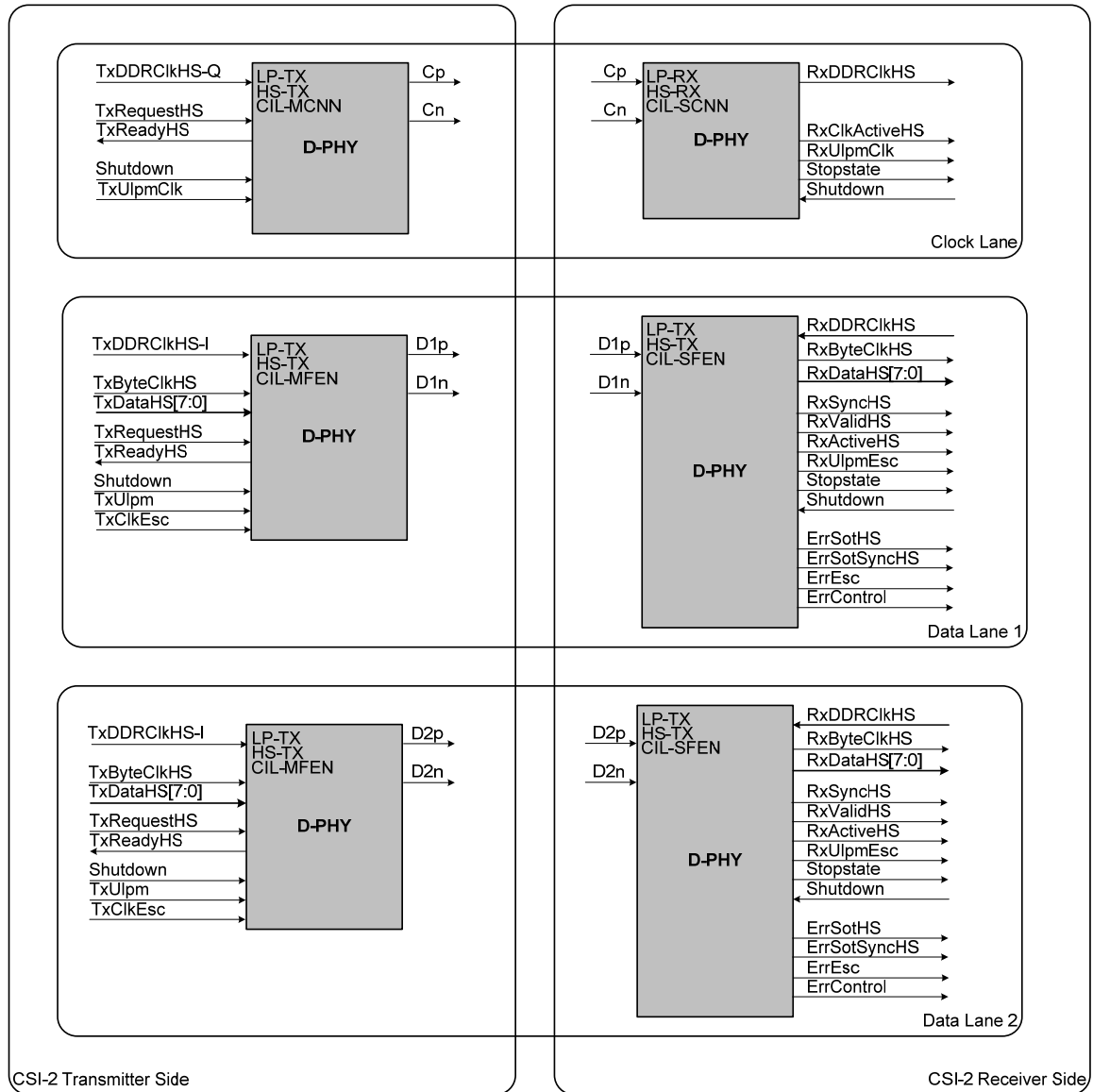
Figure 135 D-PHY Level Block Diagram

1753   The components can be categorized as:

1754   • CSI-2 Transmitter side:

1755   • Clock lane (Transmitter)

1756   • Data1 lane (Transmitter)

1757   • Data2 lane (Transmitter)

1758   • CSI-2 Receiver side:

1759   • Clock lane (Receiver)

1760   • Data1 lane (Receiver)

1761   • Data2 lane (Receiver)

1762 **B.4.1   CSI-2 Clock Lane Transmitter**

1763   The suggested implementation can be seen in Figure 136.

1764
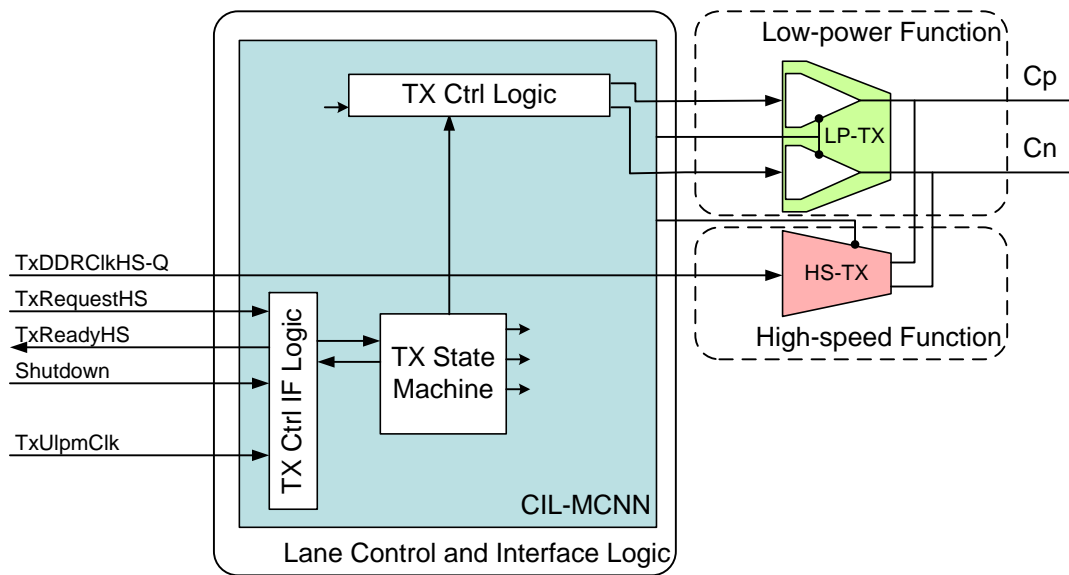
1765   **Figure 136 CSI-2 Clock Lane Transmitter**

1766   The modular D-PHY components used to build a CSI-2 clock lane transmitter are:

1767   • **LP-TX** for the Low-power function

1768   • **HS-TX** for the High-speed function

1769   • **CIL-MCNN** for the Lane control and interface logic

1770   The PPI interface signals to the CSI-2 clock lane transmitter are:

1771   • **TxDDRClkHS-Q** (Input): High-Speed Transmit DDR Clock (Quadrature).

1772   • **TxRequestHS** (Input): High-Speed Transmit Request. This active high signal causes the lane
1773   module to begin transmitting a high-speed clock.

1774   • **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that the
1775   clock lane is transmitting HS clock.

1776   • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
1777   "shutdown", disabling all activity. All line drivers, including terminators, are turned off when
1778   Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs
1779   are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on
1780   any clock.

1781   • **TxUlpmClk** (Input): Transmit Ultra Low-Power mode on Clock Lane This active high signal is
1782   asserted to cause a Clock Lane module to enter the Ultra Low-Power mode. The lane module
1783   remains in this mode until TxUlpmClk is de-asserted.

1784   **B.4.2   CSI-2 Clock Lane Receiver**

1785   The suggested implementation can be seen in Figure 137.
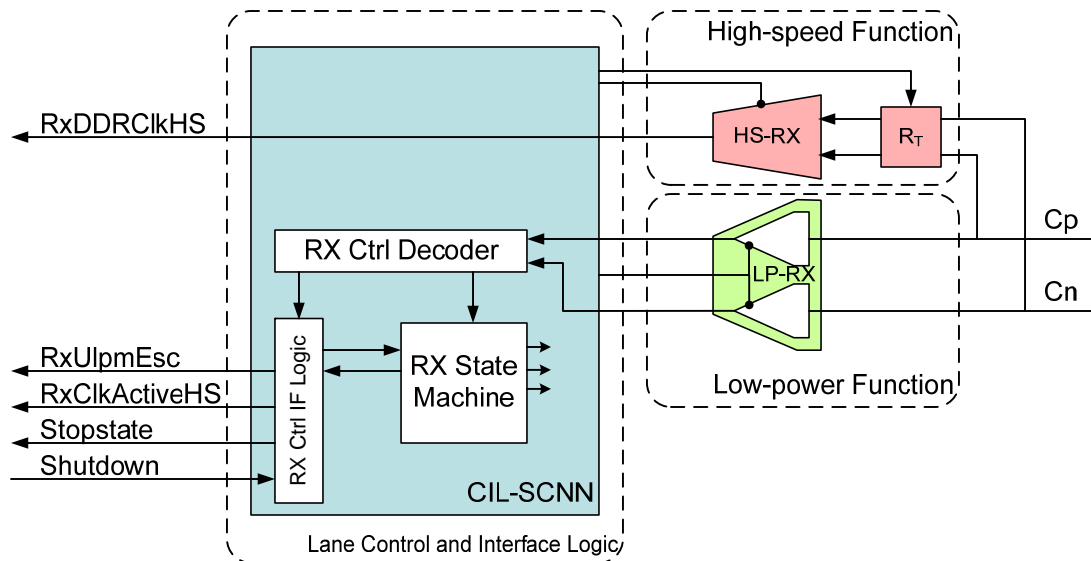
Figure 137 CSI-2 Clock Lane Receiver

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

- **LP-RX** for the Low-power function

- **HS-RX** for the High-speed function

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

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

- **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

The suggested implementation can be seen in Figure 138.

1809
1810
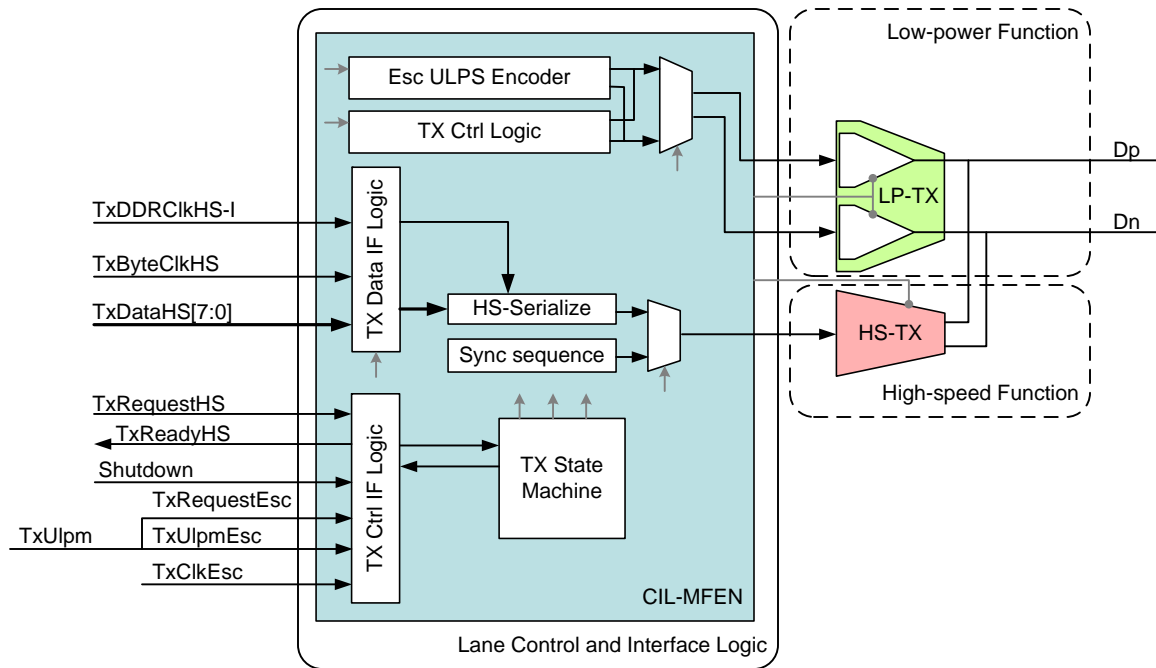
1811   **Figure 138 CSI-2 Data Lane Transmitter**

1812   The modular D-PHY components used to build a CSI-2 data lane transmitter are:

1813   • **LP-TX** for the Low-power function

1814   • **HS-TX** for the High-speed function

1815   • **CIL-MFEN** for the Lane control and interface logic

1816   The PPI interface signals to the CSI-2 data lane transmitter are:

1817   • **TxDDRClkHS-I** (Input): High-Speed Transmit DDR Clock (in-phase).

1818   • **TxByteClkHS** (Input): High-Speed Transmit Byte Clock. This is used to synchronize PPI signals
1819   in the high-speed transmit clock domain. It is recommended that both transmitting data lane
1820   modules share one TxByteClkHS signal. The frequency of TxByteClkHS must be exactly 1/8 the
1821   high-speed bit rate.

1822   • **TxDataHS[7:0]** (Input): High-Speed Transmit Data. Eight bit high-speed data to be transmitted.
1823   The signal connected to TxDataHS[0] is transmitted first. Data is registered on rising edges of
1824   TxByteClkHS.

1825   • **TxRequestHS** (Input): High-Speed Transmit Request. A low-to-high transition on TxRequestHS
1826   causes the lane module to initiate a Start-of-Transmission sequence. A high-to-low transition on
1827   TxRequest causes the lane module to initiate an End-of-Transmission sequence. This active high
1828   signal also indicates that the protocol is driving valid data on TxByteDataHS to be transmitted.
1829   The lane module accepts the data when both TxRequestHS and TxReadyHS are active on the
1830   same rising TxByteClkHS clock edge. The protocol always provides valid transmit data when
1831   TxRequestHS is active. Once asserted, TxRequestHS should remain high until the all the data has
1832   been accepted.

1833   • **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that
1834   TxDataHS is accepted by the lane module to be serially transmitted. TxReadyHS is valid on rising
1835   edges of TxByteClkHS. Valid data has to be provided for the whole duration of active
1836   TxReadyHS.

1837   •   **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
1838   "shutdown", disabling all activity. All line drivers, including terminators, are turned off when
1839   Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs
1840   are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on
1841   any clock.

1842   •   **TxUlpmEsc** (Input): Escape mode Transmit Ultra Low Power. This active high signal is asserted
1843   with TxRequestEsc to cause the lane module to enter the ultra low power mode. The lane module
1844   remains in this mode until TxRequestEsc is de-asserted.

1845   •   **TxRequestEsc** (Input): This active high signal, asserted together with TxUlpmEsc is used to
1846   request entry into escape mode. Once in escape mode, the lane stays in escape mode until
1847   TxRequestEsc is de-asserted. TxRequestEsc is only asserted by the protocol while TxRequestHS
1848   is low.

1849   •   **TxClkEsc** (Input): Escape mode Transmit Clock. This clock is directly used to generate escape
1850   sequences. The period of this clock determines the symbol time for low power signals. It is
1851   therefore constrained by the normative part of the [MIPI01].

1852   B.4.4   CSI-2 Data Lane Receiver

1853   The suggested implementation can be seen in Figure 139.

1854
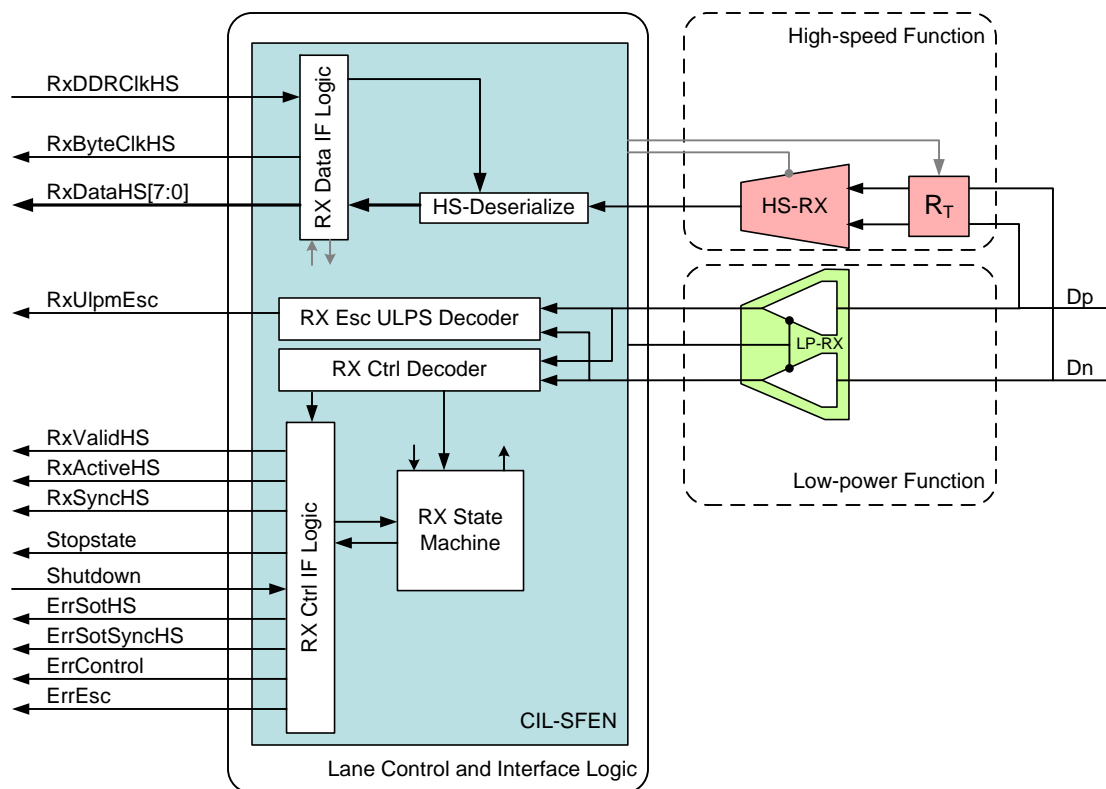
1855   **Figure 139 CSI-2 Data Lane Receiver**

1856   The modular D-PHY components used to build a CSI-2 data lane receiver are:

1857   •   **LP-RX** for the Low-power function

1858   •   **HS-RX** for the High-speed function

1859   • **CIL-SFEN** for the Lane control and interface logic

1860 The PPI interface signals to the CSI-2 data lane receiver are:

1861   • **RxDDRClkHS** (Input): High-Speed Receive DDR Clock used to sample the date in all data lanes.
1862   This signal is supplied by the CSI-2 clock lane receiver.

1863   • **RxByteClkHS** (Output): High-Speed Receive Byte Clock. This signal is used to synchronize
1864   signals in the high-speed receive clock domain. The RxByteClkHS is generated by dividing the
1865   received RxDDRClkHS.

1866   • **RXDataHS[7:0]** (Output): High-Speed Receive Data. Eight bit high-speed data received by the
1867   lane module. The signal connected to RxDataHS[0] was received first. Data is transferred on
1868   rising edges of RxByteClkHS.

1869   • **RxValidHS** (Output): High-Speed Receive Data Valid. This active high signal indicates that the
1870   lane module is driving valid data to the protocol on the RxDataHS output. There is no
1871   "RxReadyHS" signal, and the protocol is expected to capture RxDataHS on every rising edge of
1872   RxByteClkHS where RxValidHS is asserted. There is no provision for the protocol to slow down
1873   ("throttle") the receive data.

1874   • **RxActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the
1875   lane module is actively receiving a high-speed transmission from the lane interconnect.

1876   • **RxSyncHS** (Output): Receiver Synchronization Observed. This active high signal indicates that
1877   the lane module has seen an appropriate synchronization event. In a typical high-speed
1878   transmission, RxSyncHS is high for one cycle of RxByteClkHS at the beginning of a high-speed
1879   transmission when RxActiveHS is first asserted. This signal missing is signaled using
1880   ErrSotSyncHS.

1881   • **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is
1882   asserted to indicate that the lane module has entered the ultra low power mode. The lane module
1883   remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane
1884   interconnect.

1885   • **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is
1886   currently in Stop state. This signal is asynchronous.

1887   • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
1888   "shutdown", disabling all activity. All line drivers including terminators, are turned off when
1889   Shutdown is asserted. When Shutdown is high, all PPI outputs are driven to the default inactive
1890   state. Shutdown is a level sensitive signal and does not depend on any clock.

1891   • **ErrSotHS** (Output): Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is
1892   corrupted, but in such a way that proper synchronization can still be achieved, this error signal is
1893   asserted for one cycle of RxByteClkHS. This is considered to be a "soft error" in the leader
1894   sequence and confidence in the payload data is reduced.

1895   • **ErrSotSyncHS** (Output): Start-of-Transmission Synchronization Error. If the high-speed SoT
1896   leader sequence is corrupted in a way that proper synchronization cannot be expected, this error is
1897   asserted for one cycle of RxByteClkHS.

1898   • **ErrControl** (Output): Control Error. This signal is asserted when an incorrect line state sequence
1899   is detected.

1900   • **ErrEsc** (Output): Escape Entry Error. If an unrecognized escape entry command is received, this
1901   signal is asserted and remains high until the next change in line state. The only escape entry
1902   command supported by the receiver is the ULPS.

1903

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

C.1   Overview

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

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

- *D-PHY Level errors*
  Refers to any PHY related transmission error and is unrelated to the transmission's contents:

  - *Start of Transmission (SoT) errors,* which can be:

    - Recoverable, if the PHY successfully identifies the Sync code but an error was detected.

    - Unrecoverable, if the PHY does not successfully identify the sync code but does detect a HS transmission.

  - *Control Error,* which signals that the PHY has detected a control sequence that should not be present in this implementation of the Link.

- *Packet Level errors*
  This type of error refers strictly to data integrity of the received Packet Header and payload data:

  - *Packet Header errors*, signaled through the ECC code, that result in:

    - A single bit-error, which can be detected and corrected by the ECC code

    - Two bit-errors in the header, which can be detected but not corrected by the ECC code, resulting in a corrupt header

  - *Packet payload errors*, signaled through the CRC code

- *Protocol Decoding Level errors*
  This type of error refers to errors present in the decoded Packet Header or errors resulting from an incomplete sequence of events:

  - *Frame Sync Error*, caused when a FS could not be successfully paired with a FE on a given virtual channel

  - *Unrecognized ID,* caused by the presence of an unimplemented or unrecognized ID in the header

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

C.2   D-PHY Level Error

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

* **ErrSotHS:** Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is corrupted, but in such a way that proper synchronization can still be achieved, this error signal is asserted for one cycle of RxByteClkHS. This is considered to be a "soft error" in the leader sequence and confidence in the payload data is reduced.

* **ErrSotSyncHS:** Start-of-Transmission Synchronization Error. If the high-speed SoT leader sequence is corrupted in a way that proper synchronization cannot be expected, this error signal is asserted for one cycle of RxByteClkHS.

* **ErrControl:** Control Error. This signal is asserted when an incorrect line state sequence is detected. For example, if a Turn-around request or Escape Mode request is immediately followed by a Stop state instead of the required Bridge state, this signal is asserted and remains high until the next change in line state.

The recommended receiver error behavior for this level is:

* **ErrSotHS** should be passed to the Application Layer. Even though the error was detected and corrected and the Sync mechanism was unaffected, confidence in the data integrity is reduced and the application should be informed. This signal should be referenced to the corresponding data packet.

* **ErrSotSyncHS** should be passed to the Protocol Decoding Level, since this is an unrecoverable error. An unrecoverable type of error should also be signaled to the Application Layer, since the whole transmission until the first D-PHY Stop state should be ignored if this type of error occurs.

* **ErrControl** should be passed to the Application Layer, since this type of error doesn't normally occur if the interface is configured to be unidirectional. Even so, the application needs to be aware of the error and configure the interface accordingly through other, implementation specific means.

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

C.3   Packet Level Error

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

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

* **ErrEccDouble:** Asserted when an ECC syndrome was computed and two bit-errors are detected in the received Packet Header.

* **ErrEccCorrected:** Asserted when an ECC syndrome was computed and a single bit-error in the Packet Header was detected and corrected.

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

1984 Also, computing the CRC code over the whole payload of the received packet could generate the following
1985 errors:

1986 • **ErrCrc:** Asserted when the computed CRC code is different than the received CRC code.

1987 • **ErrID:** Asserted when a Packet Header is decoded with an unrecognized or unimplemented data
1988   ID.

1989 The recommended receiver error behavior for this level is:

1990 • **ErrEccDouble** should be passed to the Application Layer since assertion of this signal proves that
1991   the Packet Header information is corrupt, and therefore the WC is not usable, and thus the packet
1992   end cannot be estimated. Commonly, this type of error will be accompanied with an ErrCrc. This
1993   type of error should also be passed to the Protocol Decoding Level, since the whole transmission
1994   until D-PHY Stop state should be ignored.

1995 • **ErrEccCorrected** should be passed to the Application Layer since the application should be
1996   informed that an error had occurred but was corrected, so the received Packet Header was
1997   unaffected, although the confidence in the data integrity is reduced.

1998 • **ErrEccNoError** can be passed to the Protocol Decoding Level to signal the validity of the current
1999   Packet Header.

2000 • **ErrCrc** should be passed to the Protocol Decoding Level to indicate that the packet's payload data
2001   might be corrupt.

2002 • **ErrID** should be passed to the Application Layer to indicate that the data packet is unidentified
2003   and cannot be unpacked by the receiver. This signal should be asserted after the ID has been
2004   identified and de-asserted on the first Frame End (FE) on same virtual channel.

2005 ## C.4   Protocol Decoding Level Error

2006 The recommended behavior for this error level covers errors caused by decoding the Packet Header
2007 information and detecting a sequence that is not allowed by the CSI-2 protocol or a sequence of detected
2008 errors by the previous layers. CSI-2 implementers will commonly choose to implement this level of error
2009 handling using a state machine that should be paired with the corresponding virtual channel. The state
2010 machine should generate at least the following error signals:

2011 • **ErrFrameSync:** Asserted when a Frame End (FE) is not paired with a Frame Start (FS) on the
2012   same virtual channel. A ErrSotSyncHS should also generate this error signal.

2013 • **ErrFrameData:** Asserted after a FE when the data payload received between FS and FE contains
2014   errors.

2015 The recommended receiver error behavior for this level is:

2016 • **ErrFrameSync** should be passed to the Application Layer with the corresponding virtual channel,
2017   since the frame could not be successfully identified. Several error cases on the same virtual
2018   channel can be identified for this type of error.

2019 • If a FS is followed by a second FS on the same virtual channel, the frame corresponding to the
2020   first FS is considered in error.

2021 • If a Packet Level ErrEccDouble was signaled from the Protocol Layer, the whole transmission
2022   until the first D-PHY Stop-state should be ignored since it contains no information that can be
2023   safely decoded and cannot be qualified with a data valid signal.

2024 • If a FE is followed by a second FE on the same virtual channel, the frame corresponding to
2025   the second FE is considered in error.

2026
2027
2028

- 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.

2029
2030

- **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.

2031

2032   Annex D CSI-2 Sleep Mode (informative)

2033   D.1   Overview

2034   Since a camera in a mobile terminal spends most of its time in an inactive state, implementers need a way
2035   to put the CSI-2 Link into a low power mode that approaches, or may be as low as, the leakage level. This
2036   section proposes one approach for putting a CSI-2 Link in a "Sleep Mode" (SLM). Although the section is
2037   informative and therefore does not affect compliance for CSI-2, the approach is offered by the MIPI
2038   Camera Working Group as a recommended approach.

2039   This approach relies on an aspect of a D-PHY transmitter's behavior that permits regulators to be disabled
2040   safely when LP-00 (Space state) is on the Link. Accordingly, this will be the output state for a CSI-2
2041   camera transmitter in SLM.

2042   SLM can be thought of as a three-phase process:

2043   1.   SLM Command Phase. The 'ENTER SLM' command is issued to the TX side only, or to both
2044   sides of the Link.

2045   2.   SLM Entry Phase. The CSI-2 Link has entered, or is entering, the SLM in a controlled or
2046   synchronized manner. This phase is also part of the power-down process.

2047   3.   SLM Exit Phase. The CSI-2 Link has exited the SLM and the interface/device is operational. This
2048   phase is also part of the power-up process.

2049   In general, when in SLM, both sides of the interface will be in ULPS, as defined in [MIPI01].

2050   D.2   SLM Command Phase

2051   For the first phase, initiation of SLM occurs by a mechanism outside the scope of CSI-2. Of the many
2052   mechanisms available, two examples would be:

2053   1.   An External SLEEP signal input to the CSI-2 transmitter and optionally also to the CSI-2
2054   Receiver. When at logic 0, the CSI-2 Transmitter and, if connected, the CSI Receiver, will enter
2055   Sleep mode. When at logic 1, normal operation will take place.

2056   2.   A CCI control command, provided on the I2C control Link, is used to trigger ULPS.

2057   D.3   SLM Entry Phase

2058   For the second phase, consider one option:

2059   Only the TX side enters SLM and propagates the ULPS to the RX side by sending a D-PHY 'ULPS'
2060   command on Clock Lane and on Data Lane(s). In the following picture only Data Lane 'ULPS' command
2061   is used as an example.

Using signal XSHUTDOWN to confirm entry and exit from "Sleep" mode

Using the ULPS Sequence on Data Lane to confirm entry and exit from "Sleep" mode

2062
2063
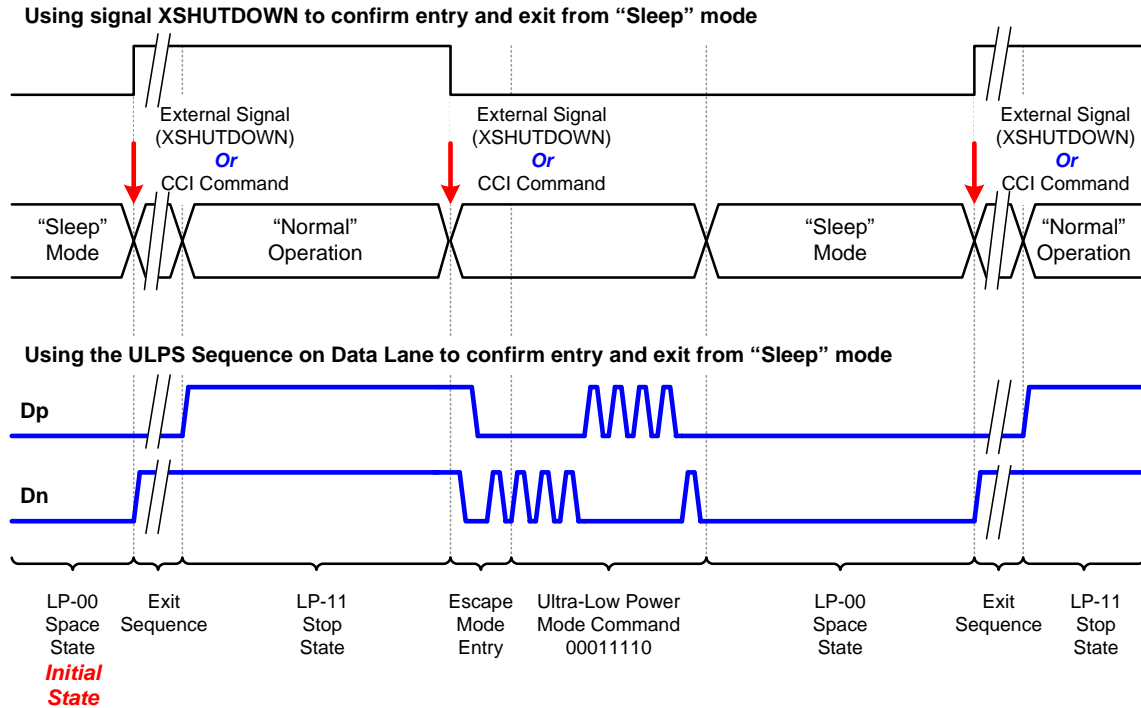
2064   **Figure 140 SLM Synchronization**

2065   **D.4   SLM Exit Phase**

2066   For the third phase, three options are presented and assume the camera peripheral is in ULPS or Sleep
2067   mode at power-up:

2068   1. Use a SLEEP signal to power-up both sides of the interface.

2069   2. Detect any CCI activity on the I2C control Link, which have been in 00 state ({SCL, SDA}), after
2070   receiving the I2C instruction to enter ULPS command as per Section D.2, option 2. Any change on
2071   those lines should wake up the camera peripheral. The drawback of this method is that I2C lines
2072   are used exclusively for control of the camera.

2073   3. Detect a wake-up sequence on the I2C lines. This sequence, which may vary by implementation,
2074   shall not disturb the I2C interface so that it can be used by other devices. One example sequence
2075   is: StopI2C-StartI2C-StopI2C. See section 6 for details on CCI.

2076   A handshake using the 'ULPS' mechanism in the as described in [MIPI01] should be used for powering up
2077   the interface.

2078

Annex E Data Compression for RAW Data Types (normative)

2081 A CSI-2 implementation using RAW data types may support compression on the interface to reduce the
2082 data bandwidth requirements between the host processor and a camera module. Data compression is not
2083 mandated by this specification. However, if data compression is used, it shall be implemented as described
2084 in this annex.

2085 Data compression schemes use an X–Y–Z naming convention where X is the number of bits per pixel in
2086 the original image, Y is the encoded (compressed) bits per pixel and Z is the decoded (uncompressed) bits
2087 per pixel.

2088 The following data compression schemes are defined:

2089 - 12–8–12

2090 - 12–7–12

2091 - 12–6–12

2092 - 10–8–10

2093 - 10–7–10

2094 - 10–6–10

2095 To identify the type of data on the CSI-2 interface, packets with compressed data shall have a User Defined
2096 Data Type value as indicated in Table 27. Note that User Defined data type codes are not reserved for
2097 compressed data types. Therefore, a CSI-2 device shall be able to communicate over the CCI the data
2098 compression scheme represented by a particular User Defined data type code for each scheme supported by
2099 the device. Note that the method to communicate the data compression scheme to Data Type code mapping
2100 is beyond the scope of this document.

2101 The number of bits in a packet shall be a multiple of eight. Therefore, implementations with data
2102 compression schemes that result in each pixel having less than eight encoded bits per pixel shall transfer the
2103 encoded data in a packed pixel format. For example, the 12–7–12 data compression scheme uses a packed
2104 pixel format as described in section 11.4.2 except the Data Type value in the Packet Header is a User
2105 Defined data type code.

2106 The data compression schemes in this annex are lossy and designed to encode each line independent of the
2107 other lines in the image.

2108 The following definitions are used in the description of the data compression schemes:

2109 - **Xorig** is the original pixel value

2110 - **Xpred** is the predicted pixel value

2111 - **Xdiff** is the difference value (**Xorig** - **Xpred**)

2112 - **Xenco** is the encoded value

2113 - **Xdeco** is the decoded pixel value

2114 The data compression system consists of encoder, decoder and predictor blocks as shown in Figure 141.

2115
2116
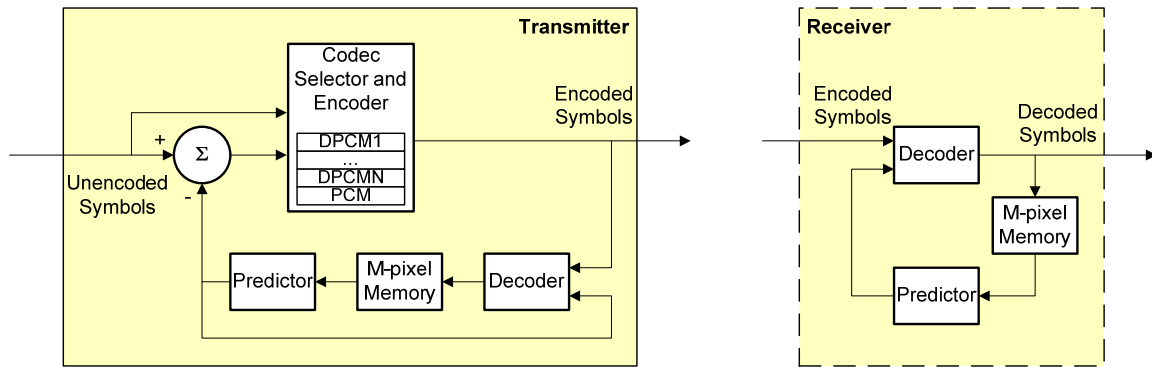
Figure 141 Data Compression System Block Diagram

2118　The encoder uses a simple algorithm to encode the pixel values. A fixed number of pixel values at the
2119　beginning of each line are encoded without using prediction. These first few values are used to initialize the
2120　predictor block. The remaining pixel values on the line are encoded using prediction.

2121　If the predicted value of the pixel, **Xpred**, is close enough to the original value of the pixel, **Xorig**,
2122　(abs(**Xorig - Xpred**) < difference limit) its difference value, **Xdiff**, is quantized using a DPCM codec.
2123　Otherwise, **Xorig** is quantized using a PCM codec. The quantized value is combined with a code word
2124　describing the codec used to quantize the pixel and the sign bit, if applicable, to create the encoded value,
2125　**Xenco**.

E.1　Predictors

2126

2127　In order to have meaningful data transfer, both the transmitter and the receiver need to use the same
2128　predictor block.

2129　The order of pixels in a raw image is shown in Figure 142.

2130
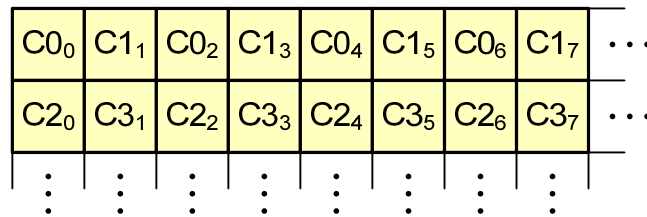
Figure 142 Pixel Order of the Original Image

2132　Figure 143 shows an example of the pixel order with RGB data.

2133
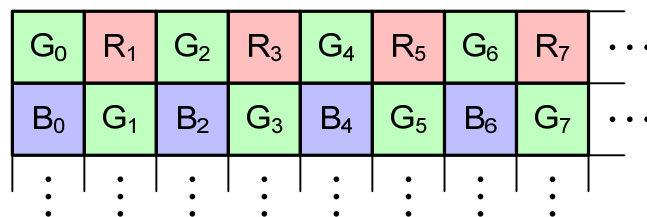
Figure 143 Example Pixel Order of the Original Image

2135　Two predictors are defined for use in the data compression schemes.

2136 Predictor1 uses a very simple algorithm and is intended to minimize processing power and memory size
2137 requirements. Typically, this predictor is used when the compression requirements are modest and the
2138 original image quality is high. Predictor1 should be used with 10–8–10, 10–7–10 and 12–8–12 data
2139 compression schemes.

2140 The second predictor, Predictor2, is more complex than Predictor1. This predictor provides slightly better
2141 prediction than Predictor1 and therefore the decoded image quality can be improved compared to
2142 Predictor1. Predictor2 should be used with 10–6–10, 12–7–12, and 12–6–12 data compression schemes.

2143 Both receiver and transmitter shall support Predictor1 for all data compression schemes.

E.1.1   Predictor1

2144

2145 Predictor1 uses only the previous same color component value as the prediction value. Therefore, only a
2146 two-pixel deep memory is required.

2147 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
2148 prediction.

2149 The prediction values for the remaining pixels in the line are calculated using the previous same color
2150 decoded value, **Xdeco**. Therefore, the predictor equation can be written as follows:

2151   **Xpred**(**n**) = **Xdeco**(**n-2**)

E.1.2   Predictor2

2152

2153 Predictor2 uses the four previous pixel values, when the prediction value is evaluated. This means that also
2154 the other color component values are used, when the prediction value has been defined. The predictor
2155 equations can be written as below.

2156 Predictor2 uses all color components of the four previous pixel values to create the prediction value.
2157 Therefore, a four-pixel deep memory is required.

2158 The first pixel ($C0_0$ / $C2_0$, or as in example G_0 / B_0) in a line is coded without prediction.

2159 The second pixel ($C1_1$ / $C3_1$ or as in example R_1 / G_1) in a line is predicted using the previous decoded
2160 different color value as a prediction value. The predictor equation for the second pixel is shown below:

2161   **Xpred**(**n**) = **Xdeco**(**n-1**)

2162 The third pixel ($C0_2$ / $C2_2$ or as in example G_2 / B_2) in a line is predicted using the previous decoded same
2163 color value as a prediction value. The predictor equation for the third pixel is shown below:

2164   **Xpred**(**n**) = **Xdeco**(**n-2**)

2165 The fourth pixel ($C1_3$ / $C3_3$ or as in example R_3 / G_3) in a line is predicted using the following equation:

2166   if ((**Xdeco**(**n-1**) <= **Xdeco**(**n-2**) AND **Xdeco**(**n-2**) <= **Xdeco**(**n-3**)) OR
2167   (**Xdeco**(**n-1**) >= **Xdeco**(**n-2**) AND **Xdeco**(**n-2**) >= **Xdeco**(**n-3**))) then
2168   **Xpred**(**n**) = **Xdeco**(**n-1**)
2169   else
2170   **Xpred**(**n**) = **Xdeco**(**n-2**)
2171   endif

2172   Other pixels in all lines are predicted using the equation:

2173   if ((**Xdeco**(**n-1**) <= **Xdeco**(**n-2**) AND **Xdeco**(**n-2**) <= **Xdeco**(**n-3**)) OR
2174   (**Xdeco**(**n-1**) >= **Xdeco**(**n-2**) AND **Xdeco**(**n-2**) >= **Xdeco**(**n-3**))) then
2175   **Xpred**(**n**) = **Xdeco**(**n-1**)
2176   else if ((**Xdeco**(**n-1**) <= **Xdeco**(**n-3**) AND **Xdeco**(**n-2**) <= **Xdeco**(**n-4**)) OR
2177   (**Xdeco**(**n-1**) >= **Xdeco**(**n-3**) AND **Xdeco**(**n-2**) >= **Xdeco**(**n-4**))) then
2178   **Xpred**(**n**) = **Xdeco**(**n-2**)
2179   else
2180   **Xpred**(**n**) = (**Xdeco**(**n-2**) + **Xdeco**(**n-4**) + 1) / 2
2181   endif

E.2   Encoders

2183   There are six different encoders available, one for each data compression scheme.

2184   For all encoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
2185   for predicted pixels.

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

2187   The 10–8–10 coder offers a 20% bit rate reduction with very high image quality.

2188   Pixels without prediction are encoded using the following formula:

2189   **Xenco**(**n**) = **Xorig**(**n**) / 4

2190   To avoid a full-zero encoded value, the following check is performed:

2191   if (**Xenco**(**n**) == 0) then
2192   **Xenco**(**n**) = 1
2193   endif

2194   Pixels with prediction are encoded using the following formula:

2195   if (abs(**Xdiff**(**n**)) < 32) then
2196   use **DPCM1**
2197   else if (abs(**Xdiff**(**n**)) < 64) then
2198   use **DPCM2**
2199   else if (abs(**Xdiff**(**n**)) < 128) then
2200   use **DPCM3**
2201   else
2202   use **PCM**
2203   endif

E.2.1.1   DPCM1 for 10–8–10 Coder

2205   **Xenco**(**n**) has the following format:

2206   **Xenco**(**n**) = "00 s xxxxx"

2207   where,

2208   "00" is the code word
2209   "s" is the **sign** bit
2210   "xxxxx" is the five bit **value** field

2211   The coder equation is described as follows:

2212   if (**Xdiff**(**n**) <= 0) then
2213   **sign** = 1
2214   else
2215   **sign** = 0
2216   endif
2217   **value** = abs(**Xdiff**(**n**))

2218   Note: Zero code has been avoided (0 is sent as -0).

2219   **E.2.1.2   DPCM2 for 10–8–10 Coder**

2220   **Xenco**(**n**) has the following format:

2221   **Xenco**(**n**) = "010 s xxxx"

2222   where,

2223   "010" is the code word
2224   "s" is the **sign** bit
2225   "xxxx" is the four bit **value** field

2226   The coder equation is described as follows:

2227   if (**Xdiff**(**n**) < 0) then
2228   **sign** = 1
2229   else
2230   **sign** = 0
2231   endif
2232   **value** = (abs(**Xdiff**(**n**)) - 32) / 2
2233

2234   **E.2.1.3   DPCM3 for 10–8–10 Coder**

2235   **Xenco**(**n**) has the following format:

2236   **Xenco**(**n**) = "011 s xxxx"

2237   where,

2238   "010" is the code word
2239   "s" is the **sign** bit
2240   "xxxx" is the four bit **value** field

2241   The coder equation is described as follows:

2242   if (**Xdiff**(**n**) < 0) then
2243   **sign** = 1
2244   else
2245   **sign** = 0
2246   endif
2247   **value** = (abs(**Xdiff**(**n**)) - 64) / 4

2248   **E.2.1.4   PCM for 10–8–10 Coder**

2249   **Xenco**(**n**) has the following format:

2250   **Xenco**(**n**) = "1 xxxxxxx"

2251   where,

2252   "1" is the code word
2253   the **sign** bit is not used
2254   "xxxxxxx" is the seven bit **value** field

2255   The coder equation is described as follows:

2256   **value** = **Xorig**(**n**) / 8

2257   **E.2.2   Coder for 10–7–10 Data Compression**

2258   The 10–7–10 coder offers 30% bit rate reduction with high image quality.

2259   Pixels without prediction are encoded using the following formula:

2260   **Xenco**(**n**) = **Xorig**(**n**) / 8

2261   To avoid a full-zero encoded value, the following check is performed:

2262   if (**Xenco**(**n**) == 0) then
2263   **Xenco**(**n**) = 1

2264   Pixels with prediction are encoded using the following formula:

2265   if (abs(**Xdiff**(**n**)) < 8) then
2266   use **DPCM1**
2267   else if (abs(**Xdiff**(**n**)) < 16) then
2268   use **DPCM2**
2269   else if (abs(**Xdiff**(**n**)) < 32) then
2270   use **DPCM3**
2271   else if (abs(**Xdiff**(**n**)) < 160) then
2272   use **DPCM4**
2273   else
2274   use **PCM**
2275   endif

2276   **E.2.2.1   DPCM1 for 10–7–10 Coder**

2277   **Xenco**(**n**) has the following format:

2278   **Xenco**(**n**) = "000 s xxx"

2279   where,

2280   "000" is the code word
2281   "s" is the **sign** bit
2282   "xxx" is the three bit **value** field

2283   The coder equation is described as follows:

2284   if (**Xdiff**(**n**) <= 0) then
2285   **sign** = 1
2286   else
2287   **sign** = 0
2288   endif
2289   **value** = abs(**Xdiff**(**n**))

2290   Note: Zero code has been avoided (0 is sent as -0).

2291   **E.2.2.2   DPCM2 for 10–7–10 Coder**

2292   **Xenco**(**n**) has the following format:

2293   **Xenco**(**n**) = "0010 s xx"

2294   where,

2295   "0010" is the code word
2296   "s" is the **sign** bit
2297   "xx" is the two bit **value** field

2298   The coder equation is described as follows:

2299   if (**Xdiff**(**n**) < 0) then
2300   **sign** = 1
2301   else
2302   **sign** = 0
2303   endif
2304   **value** = (abs(**Xdiff**(**n**)) - 8) / 2

2305   **E.2.2.3   DPCM3 for 10–7–10 Coder**

2306   **Xenco**(**n**) has the following format:

2307   **Xenco**(**n**) = "0011 s xx"

2308   where,

2309   "0011" is the code word
2310   "s" is the **sign** bit
2311   "xx" is the two bit **value** field

2312   The coder equation is described as follows:

2313   if (**Xdiff**(**n**) < 0) then
2314   **sign** = 1
2315   else
2316   **sign** = 0
2317   endif
2318   **value** = (abs(**Xdiff**(**n**)) - 16) / 4

2319   **E.2.2.4   DPCM4 for 10–7–10 Coder**

2320   **Xenco**(**n**) has the following format:

2321   **Xenco**(**n**) = "01 s xxxx"

2322   where,

2323   "01" is the code word
2324   "s" is the **sign** bit
2325   "xxxx" is the four bit **value** field

2326   The coder equation is described as follows:

2327   if (**Xdiff**(**n**) < 0) then
2328   **sign** = 1
2329   else
2330   **sign** = 0
2331   endif
2332   **value** = (abs(**Xdiff**(**n**)) - 32) / 8

2333   **E.2.2.5   PCM for 10–7–10 Coder**

2334   **Xenco**(**n**) has the following format:

2335   **Xenco**(**n**) = "1 xxxxxx"

2336   where,

2337   "1" is the code word
2338   the **sign** bit is not used
2339   "xxxxxx" is the six bit **value** field

2340   The coder equation is described as follows:

2341   **value** = **Xorig**(**n**) / 16

2342   ## E.2.3   Coder for 10–6–10 Data Compression

2343   The 10–6–10 coder offers 40% bit rate reduction with acceptable image quality.

2344   Pixels without prediction are encoded using the following formula:

2345   **Xenco**(**n**) = **Xorig**(**n**) / 16

2346   To avoid a full-zero encoded value, the following check is performed:

2347   if (**Xenco**(**n**) == 0) then
2348   **Xenco**(**n**) = 1
2349   endif

2350   Pixels with prediction are encoded using the following formula:

2351   if (abs(**Xdiff**(**n**)) < 1) then
2352   use **DPCM1**
2353   else if (abs(**Xdiff**(**n**)) < 3) then
2354   use **DPCM2**
2355   else if (abs(**Xdiff**(**n**)) < 11) then
2356   use **DPCM3**
2357   else if (abs(**Xdiff**(**n**)) < 43) then
2358   use **DPCM4**
2359   else if (abs(**Xdiff**(**n**)) < 171) then
2360   use **DPCM5**
2361   else
2362   use **PCM**
2363   endif

2364   ### E.2.3.1   DPCM1 for 10–6–10 Coder

2365   **Xenco**(**n**) has the following format:

2366   **Xenco**(**n**) = "00000 s"

2367   where,

2368   "00000" is the code word
2369   "s" is the **sign** bit
2370   the **value** field is not used

2371   The coder equation is described as follows:

2372   **sign** = 1

2373   Note: Zero code has been avoided (0 is sent as -0).

2374   ### E.2.3.2   DPCM2 for 10–6–10 Coder

2375   **Xenco**(**n**) has the following format:

2376   **Xenco**(**n**) = "00001 s"

2377   where,

2378   "00001" is the code word
2379   "s" is the **sign** bit
2380   the **value** field is not used

2381   The coder equation is described as follows:

2382   if (**Xdiff**(**n**) < 0) then
2383   **sign** = 1
2384   else
2385   **sign** = 0
2386   endif

2387   **E.2.3.3   DPCM3 for 10–6–10 Coder**

2388   **Xenco**(**n**) has the following format:

2389   **Xenco**(**n**) = "0001 s x"

2390   where,

2391   "0001" is the code word
2392   "s" is the **sign** bit
2393   "x" is the one bit **value** field

2394   The coder equation is described as follows:

2395   if (**Xdiff**(**n**) < 0) then
2396   **sign** = 1
2397   else
2398   **sign** = 0
2399   **value** = (abs(**Xdiff**(**n**)) - 3) / 4
2400   endif

2401   **E.2.3.4   DPCM4 for 10–6–10 Coder**

2402   **Xenco**(**n**) has the following format:

2403   **Xenco**(**n**) = "001 s xx"

2404   where,

2405   "001" is the code word
2406   "s" is the **sign** bit
2407   "xx" is the two bit **value** field

2408   The coder equation is described as follows:

2409   if (**Xdiff**(**n**) < 0) then
2410   **sign** = 1
2411   else
2412   **sign** = 0
2413   endif

2414   **value** = (abs(**Xdiff**(**n**)) - 11) / 8

E.2.3.5   DPCM5 for 10–6–10 Coder

2416   **Xenco**(**n**) has the following format:

2417   **Xenco**(**n**) = "01 s xxx"

2418   where,

2419   "01" is the code word
2420   "s" is the **sign** bit
2421   "xxx" is the three bit **value** field

2422   The coder equation is described as follows:

2423   if (**Xdiff**(**n**) < 0) then
2424   **sign** = 1
2425   else
2426   **sign** = 0
2427   endif
2428   **value** = (abs(**Xdiff**(**n**)) - 43) / 16

E.2.3.6   PCM for 10–6–10 Coder

2430   **Xenco**(**n**) has the following format:

2431   **Xenco**(**n**) = "1 xxxxx"

2432   where,

2433   "1" is the code word
2434   the **sign** bit is not used
2435   "xxxxx" is the five bit **value** field

2436   The coder equation is described as follows:

2437   **value** = **Xorig**(**n**) / 32

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

2439   The 12–8–12 coder offers 33% bit rate reduction with very high image quality.

2440   Pixels without prediction are encoded using the following formula:

2441   **Xenco**(**n**) = **Xorig**(**n**) / 16

2442   To avoid a full-zero encoded value, the following check is performed:

2443   if (**Xenco**(**n**) == 0) then
2444   **Xenco**(**n**) = 1
2445   endif

2446   Pixels with prediction are encoded using the following formula:

2447   if (abs(**Xdiff**(**n**)) < 8) then
2448   use **DPCM1**
2449   else if (abs(**Xdiff**(**n**)) < 40) then
2450   use **DPCM2**
2451   else if (abs(**Xdiff**(**n**)) < 104) then
2452   use **DPCM3**
2453   else if (abs(**Xdiff**(**n**)) < 232) then
2454   use **DPCM4**
2455   else if (abs(**Xdiff**(**n**)) < 360) then
2456   use **DPCM5**
2457   else
2458   use **PCM**

2459   **E.2.4.1   DPCM1 for 12–8–12 Coder**

2460   **Xenco**(**n**) has the following format:

2461   **Xenco**(**n**) = "0000 s xxx"

2462   where,

2463   "0000" is the code word
2464   "s" is the **sign** bit
2465   "xxx" is the three bit **value** field

2466   The coder equation is described as follows:

2467   if (**Xdiff**(**n**) <= 0) then
2468   **sign** = 1
2469   else
2470   **sign** = 0
2471   endif
2472   **value** = abs(**Xdiff**(**n**))

2473   Note: Zero code has been avoided (0 is sent as -0).

2474   **E.2.4.2   DPCM2 for 12–8–12 Coder**

2475   **Xenco**(**n**) has the following format:

2476   **Xenco**(**n**) = "011 s xxxx"

2477   where,

2478   "011" is the code word
2479   "s" is the **sign** bit
2480   "xxxx" is the four bit **value** field

2481   The coder equation is described as follows:

```
2482          if (Xdiff( n ) < 0) then
2483              sign = 1
2484          else
2485              sign = 0
2486          endif
2487          value = (abs(Xdiff( n )) - 8) / 2
```

2488   E.2.4.3   DPCM3 for 12–8–12 Coder

2489   **Xenco**(**n**) has the following format:

2490   **Xenco**(**n**) = "010 s xxxx"

2491   where,

2492   "010" is the code word
2493   "s" is the **sign** bit
2494   "xxxx" is the four bit **value** field

2495   The coder equation is described as follows:

```
2496          if (Xdiff( n ) < 0) then
2497              sign = 1
2498          else
2499              sign = 0
2500          endif
2501          value = (abs(Xdiff( n )) - 40) / 4
```

2502   E.2.4.4   DPCM4 for 12–8–12 Coder

2503   **Xenco**(**n**) has the following format:

2504   **Xenco**(**n**) = "001 s xxxx"

2505   where,

2506   "001" is the code word
2507   "s" is the **sign** bit
2508   "xxxx" is the four bit **value** field

2509   The coder equation is described as follows:

```
2510          if (Xdiff( n ) < 0) then
2511              sign = 1
2512          else
2513              sign = 0
2514          endif
2515          value = (abs(Xdiff( n )) - 104) / 8
```

2516 **E.2.4.5   DPCM5 for 12–8–12 Coder**

2517 **Xenco**(**n**) has the following format:

2518   **Xenco**(**n**) = "0001 s xxx"

2519   where,

2520   "0001" is the code word
2521   "s" is the **sign** bit
2522   "xxx" is the three bit **value** field

2523 The coder equation is described as follows:

2524   if (**Xdiff**(**n**) < 0) then
2525   **sign** = 1
2526   else
2527   **sign** = 0
2528   endif
2529   **value** = (abs(**Xdiff**(**n**)) - 232) / 16

2530 **E.2.4.6   DPCM5 for 12–8–12 Coder**

2531 **Xenco**(**n**) has the following format:

2532   **Xenco**(**n**) = "1 xxxxxxx"

2533   where,

2534   "1" is the code word
2535   the **sign** bit is not used
2536   "xxxxxxx" is the seven bit **value** field

2537 The coder equation is described as follows:

2538   **value** = **Xorig**(**n**) / 32

2539 **E.2.5   Coder for 12–7–12 Data Compression**

2540 The 12–7–12 coder offers 42% bit rate reduction with high image quality.

2541 Pixels without prediction are encoded using the following formula:

2542   **Xenco**(**n**) = **Xorig**(**n**) / 32

2543 To avoid a full-zero encoded value, the following check is performed:

2544   if (**Xenco**(**n**) == 0) then
2545   **Xenco**(**n**) = 1
2546   endif

2547   Pixels with prediction are encoded using the following formula:

```
2548        if (abs(Xdiff( n )) < 4) then
2549            use DPCM1
2550        else if (abs(Xdiff( n )) < 12) then
2551            use DPCM2
2552        else if (abs(Xdiff( n )) < 28) then
2553            use DPCM3
2554        else if (abs(Xdiff( n )) < 92) then
2555            use DPCM4
2556        else if (abs(Xdiff( n )) < 220) then
2557            use DPCM5
2558        else if (abs(Xdiff( n )) < 348) then
2559            use DPCM6
2560        else
2561            use PCM
2562        endif
```

2563   **E.2.5.1   DPCM1 for 12–7–12 Coder**

2564   **Xenco**(**n**) has the following format:

2565   **Xenco**(**n**) = "0000 s xx"

2566   where,

2567   "0000" is the code word
2568   "s" is the **sign** bit
2569   "xx" is the two bit **value** field

2570   The coder equation is described as follows:

```
2571        if (Xdiff( n ) <= 0) then
2572            sign = 1
2573        else
2574            sign = 0
2575        endif
2576        value = abs(Xdiff( n ))
```

2577   Note: Zero code has been avoided (0 is sent as -0).

2578   **E.2.5.2   DPCM2 for 12–7–12 Coder**

2579   **Xenco**(**n**) has the following format:

2580   **Xenco**(**n**) = "0001 s xx"

2581   where,

2582   "0001" is the code word
2583   "s" is the **sign** bit
2584   "xx" is the two bit **value** field

2585   The coder equation is described as follows:

```
2586            if (Xdiff( n ) < 0) then
2587                sign = 1
2588            else
2589                sign = 0
2590            endif
2591            value = (abs(Xdiff( n )) - 4) / 2
```

2592   **E.2.5.3   DPCM3 for 12–7–12 Coder**

2593   **Xenco**(**n**) has the following format:

2594   **Xenco**(**n**) = "0010 s xx"

2595   where,

2596   "0010" is the code word
2597   "s" is the **sign** bit
2598   "xx" is the two bit **value** field

2599   The coder equation is described as follows:

```
2600            if (Xdiff( n ) < 0) then
2601                sign = 1
2602            else
2603                sign = 0
2604            endif
2605            value = (abs(Xdiff( n )) - 12) / 4
```

2606   **E.2.5.4   DPCM4 for 12–7–12 Coder**

2607   **Xenco**(**n**) has the following format:

2608   **Xenco**(**n**) = "010 s xxx"

2609   where,

2610   "010" is the code word
2611   "s" is the **sign** bit
2612   "xxx" is the three bit **value** field

2613   The coder equation is described as follows:

```
2614            if (Xdiff( n ) < 0) then
2615                sign = 1
2616            else
2617                sign = 0
2618            endif
2619            value = (abs(Xdiff( n )) - 28) / 8
```

2620   **E.2.5.5   DPCM5 for 12–7–12 Coder**

2621   **Xenco**(**n**) has the following format:

2622   **Xenco**(**n**) = "011 s xxx"

2623   where,

2624   "011" is the code word
2625   "s" is the **sign** bit
2626   "xxx" is the three bit **value** field

2627   The coder equation is described as follows:

2628   if (**Xdiff**(**n**) < 0) then
2629   **sign** = 1
2630   else
2631   **sign** = 0
2632   endif
2633   **value** = (abs(**Xdiff**(**n**)) - 92) / 16

2634   **E.2.5.6   DPCM6 for 12–7–12 Coder**

2635   **Xenco**(**n**) has the following format:

2636   **Xenco**(**n**) = "0011 s xx"

2637   where,

2638   "0011" is the code word
2639   "s" is the **sign** bit
2640   "xx" is the two bit **value** field

2641   The coder equation is described as follows:

2642   if (**Xdiff**(**n**) < 0) then
2643   **sign** = 1
2644   else
2645   **sign** = 0
2646   endif
2647   **value** = (abs(**Xdiff**(**n**)) - 220) / 32

2648   **E.2.5.7   PCM for 12–7–12 Coder**

2649   **Xenco**(**n**) has the following format:

2650   **Xenco**(**n**) = "1 xxxxxx"

2651   where,

2652   "1" is the code word
2653   the **sign** bit is not used
2654   "xxxxxx" is the six bit **value** field

2655   The coder equation is described as follows:

2656   **value** = **Xorig**(**n**) / 64

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

2658   The 12–6–12 coder offers 50% bit rate reduction with acceptable image quality.

2659   Pixels without prediction are encoded using the following formula:

2660   **Xenco**(**n**) = **Xorig**(**n**) / 64

2661   To avoid a full-zero encoded value, the following check is performed:

2662   if (**Xenco**(**n**) == 0) then
2663   **Xenco**(**n**) = 1
2664   endif

2665   Pixels with prediction are encoded using the following formula:

2666   if (abs(**Xdiff**(**n**)) < 2) then
2667   use **DPCM1**
2668   else if (abs(**Xdiff**(**n**)) < 10) then
2669   use **DPCM3**
2670   else if (abs(**Xdiff**(**n**)) < 42) then
2671   use **DPCM4**
2672   else if (abs(**Xdiff**(**n**)) < 74) then
2673   use **DPCM5**
2674   else if (abs(**Xdiff**(**n**)) < 202) then
2675   use **DPCM6**
2676   else if (abs(**Xdiff**(**n**)) < 330) then
2677   use **DPCM7**
2678   else
2679   use **PCM**
2680   endif

2681   Note: **DPCM2** is not used.

E.2.6.1   DPCM1 for 12–6–12 Coder

2683   **Xenco**(**n**) has the following format:

2684   **Xenco**(**n**) = "0000 s x"

2685   where,

2686   "0000" is the code word
2687   "s" is the **sign** bit
2688   "x" is the one bit **value** field

2689   The coder equation is described as follows:

2690   if (**Xdiff**(**n**) <= 0) then
2691   **sign** = 1
2692   else
2693   **sign** = 0
2694   endif
2695   **value** = abs(**Xdiff**(**n**))

2696   Note: Zero code has been avoided (0 is sent as -0).

2697   **E.2.6.2   DPCM3 for 12–6–12 Coder**

2698   **Xenco**(**n**) has the following format:

2699   **Xenco**(**n**) = "0001 s x"

2700   where,

2701   "0001" is the code word
2702   "s" is the **sign** bit
2703   "x" is the one bit **value** field

2704   The coder equation is described as follows:

2705   if (**Xdiff**(**n**) < 0) then
2706   **sign** = 1
2707   else
2708   **sign** = 0
2709   endif
2710   **value** = (abs(**Xdiff**(**n**)) - 2) / 4

2711   **E.2.6.3   DPCM4 for 12–6–12 Coder**

2712   **Xenco**(**n**) has the following format:

2713   **Xenco**(**n**) = "010 s xx"

2714   where,

2715   "010" is the code word
2716   "s" is the **sign** bit
2717   "xx" is the two bit **value** field

2718   The coder equation is described as follows:

2719   if (**Xdiff**(**n**) < 0) then
2720   **sign** = 1
2721   else
2722   **sign** = 0
2723   endif
2724   **value** = (abs(**Xdiff**(**n**)) - 10) / 8

2725   **E.2.6.4   DPCM5 for 12–6–12 Coder**

2726   **Xenco**(**n**) has the following format:

2727   **Xenco**(**n**) = "0010 s x"

2728   where,

2729   "0010" is the code word
2730   "s" is the **sign** bit
2731   "x" is the one bit **value** field

2732   The coder equation is described as follows:

2733   if (**Xdiff**(**n**) < 0) then
2734   **sign** = 1
2735   else
2736   **sign** = 0
2737   endif
2738   **value** = (abs(**Xdiff**(**n**)) - 42) / 16

2739   **E.2.6.5   DPCM6 for 12–6–12 Coder**

2740   **Xenco**(**n**) has the following format:

2741   **Xenco**(**n**) = "011 s xx"

2742   where,

2743   "011" is the code word
2744   "s" is the **sign** bit
2745   "xx" is the two bit **value** field

2746   The coder equation is described as follows:

2747   if (**Xdiff**(**n**) < 0) then
2748   **sign** = 1
2749   else
2750   **sign** = 0
2751   endif
2752   **value** = (abs(**Xdiff**(**n**)) - 74) / 32

2753   **E.2.6.6   DPCM7 for 12–6–12 Coder**

2754   **Xenco**(**n**) has the following format:

2755   **Xenco**(**n**) = "0011 s x"

2756   where,

2757   "0011" is the code word
2758   "s" is the **sign** bit
2759   "x" is the one bit **value** field

2760   The coder equation is described as follows:

2761   if (**Xdiff**(**n**) < 0) then
2762   **sign** = 1
2763   else
2764   **sign** = 0
2765   endif
2766   **value** = (abs(**Xdiff**(**n**)) - 202) / 64

2767   **E.2.6.7   PCM for 12–6–12 Coder**

2768   **Xenco**(**n**) has the following format:

2769   **Xenco**(**n**) = "1 xxxxx"

2770   where,

2771   "1" is the code word
2772   the **sign** bit is not used
2773   "xxxxx" is the five bit **value** field

2774   The coder equation is described as follows:

2775   **value** = **Xorig**(**n**) / 128

2776   **E.3   Decoders**

2777   There are six different decoders available, one for each data compression scheme.

2778   For all decoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
2779   for predicted pixels.

2780   **E.3.1   Decoder for 10–8–10 Data Compression**

2781   Pixels without prediction are decoded using the following formula:

2782   **Xdeco**(**n**) = 4 * **Xenco**(**n**) + 2

2783   Pixels with prediction are decoded using the following formula:

2784   if (**Xenco**(**n**) & 0xc0 == 0x00) then
2785   use **DPCM1**
2786   else if (**Xenco**(**n**) & 0xe0 == 0x40) then
2787   use **DPCM2**
2788   else if (**Xenco**(**n**) & 0xe0 == 0x60) then
2789   use **DPCM3**
2790   else
2791   use **PCM**
2792   endif
2793

2794   **E.3.1.1   DPCM1 for 10–8–10 Decoder**

2795   **Xenco**(**n**) has the following format:

2796   **Xenco**(**n**) = "00 s xxxxx"

2797   where,

2798   "00" is the code word
2799   "s" is the **sign** bit
2800   "xxxxx" is the five bit **value** field

2801   The decoder equation is described as follows:

2802   **sign** = **Xenco**(**n**) & 0x20
2803   **value** = **Xenco**(**n**) & 0x1f
2804   if (**sign** > 0) then
2805   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
2806   else
2807   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
2808   endif

2809   **E.3.1.2   DPCM2 for 10–8–10 Decoder**

2810   **Xenco**(**n**) has the following format:

2811   **Xenco**(**n**) = "010 s xxxx"

2812   where,

2813   "010" is the code word
2814   "s" is the **sign** bit
2815   "xxxx" is the four bit **value** field

2816   The decoder equation is described as follows:

2817   **sign** = **Xenco**(**n**) & 0x10
2818   **value** = 2 * (**Xenco**(**n**) & 0xf) + 32
2819   if (**sign** > 0) then
2820   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
2821   else
2822   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
2823   endif

2824   **E.3.1.3   DPCM3 for 10–8–10 Decoder**

2825   **Xenco**(**n**) has the following format:

2826   **Xenco**(**n**) = "011 s xxxx"

2827   where,

2828   "011" is the code word
2829   "s" is the **sign** bit
2830   "xxxx" is the four bit **value** field

2831   The decoder equation is described as follows:

2832   **sign** = **Xenco**(**n**) & 0x10
2833   **value** = 4 * (**Xenco**(**n**) & 0xf) + 64 + 1
2834   if (**sign** > 0) then
2835   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
2836   if (**Xdeco**(**n**) < 0) then
2837   **Xdeco**(**n**) = 0
2838   endif
2839   else
2840   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
2841   if (**Xdeco**(**n**) > 1023) then
2842   **Xdeco**(**n**) = 1023
2843   endif
2844   endif

2845   **E.3.1.4   PCM for 10–8–10 Decoder**

2846   **Xenco**(**n**) has the following format:

2847   **Xenco**(**n**) = "1 xxxxxxx"

2848   where,

2849   "1" is the code word
2850   the **sign** bit is not used
2851   "xxxxxxx" is the seven bit **value** field

2852   The codec equation is described as follows:

2853   **value** = 8 * (**Xenco**(**n**) & 0x7f)
2854   if (**value** > **Xpred**(**n**)) then
2855   **Xdeco**(**n**) = **value** + 3
2856   endif
2857   else
2858   **Xdeco**(**n**) = **value** + 4
2859   endif

2860   **E.3.2   Decoder for 10–7–10 Data Compression**

2861   Pixels without prediction are decoded using the following formula:

2862   **Xdeco**(**n**) = 8 * **Xenco**(**n**) + 4

2863   Pixels with prediction are decoded using the following formula:

2864   if (**Xenco**(**n**) & 0x70 == 0x00) then
2865   use **DPCM1**
2866   else if (**Xenco**(**n**) & 0x78 == 0x10) then
2867   use **DPCM2**
2868   else if (**Xenco**(**n**) & 0x78 == 0x18) then
2869   use **DPCM3**
2870   else if (**Xenco**(**n**) & 0x60 == 0x20) then
2871   use **DPCM4**
2872   else
2873   use **PCM**
2874   endif

2875   **E.3.2.1   DPCM1 for 10–7–10 Decoder**

2876   **Xenco**(**n**) has the following format:

2877   **Xenco**(**n**) = "000 s xxx"

2878   where,

2879   "000" is the code word
2880   "s" is the **sign** bit
2881   "xxx" is the three bit **value** field

2882   The codec equation is described as follows:

2883   **sign** = **Xenco**(**n**) & 0x8
2884   **value** = **Xenco**(**n**) & 0x7
2885   if (**sign** > 0) then
2886   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
2887   else
2888   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
2889   endif

2890   **E.3.2.2   DPCM2 for 10–7–10 Decoder**

2891   **Xenco**(**n**) has the following format:

2892   **Xenco**(**n**) = "0010 s xx"

2893   where,

2894   "0010" is the code word
2895   "s" is the **sign** bit
2896   "xx" is the two bit **value** field

2897   The codec equation is described as follows:

2898   **sign** = **Xenco**(**n**) & 0x4
2899   **value** = 2 * (**Xenco**(**n**) & 0x3) + 8

```
2900          if (sign > 0) then
2901              Xdeco( n ) = Xpred( n ) - value
2902          else
2903              Xdeco( n ) = Xpred( n ) + value
2904          endif
```

E.3.2.3   DPCM3 for 10–7–10 Decoder

2906 **Xenco**(**n**) has the following format:

2907   **Xenco**(**n**) = "0011 s xx"

2908   where,

2909   "0011" is the code word
2910   "s" is the **sign** bit
2911   "xx" is the two bit **value** field

2912 The codec equation is described as follows:

```
2913          sign = Xenco( n ) & 0x4
2914          value = 4 * (Xenco( n ) & 0x3) + 16 + 1
2915          if (sign > 0) then
2916              Xdeco( n ) = Xpred( n ) - value
2917              if (Xdeco( n ) < 0) then
2918                  Xdeco( n ) = 0
2919              endif
2920          else
2921              Xdeco( n ) = Xpred( n ) + value
2922              if (Xdeco( n ) > 1023) then
2923                  Xdeco( n ) = 1023
2924              endif
2925          endif
```

E.3.2.4   DPCM4 for 10–7–10 Decoder

2927 **Xenco**(**n**) has the following format:

2928   **Xenco**(**n**) = "01 s xxxx"

2929   where,

2930   "01" is the code word
2931   "s" is the **sign** bit
2932   "xxxx" is the four bit **value** field

2933 The codec equation is described as follows:

```
2934          sign = Xenco( n ) & 0x10
2935          value = 8 * (Xenco( n ) & 0xf) + 32 + 3
```

```
2936        if (sign > 0) then
2937            Xdeco( n ) = Xpred( n ) - value
2938            if (Xdeco( n ) < 0) then
2939                Xdeco( n ) = 0
2940            endif
2941        else
2942            Xdeco( n ) = Xpred( n ) + value
2943            if (Xdeco( n ) > 1023) then
2944                Xdeco( n ) = 1023
2945            endif
2946        endif
```

2947 **E.3.2.5   PCM for 10–7–10 Decoder**

2948 **Xenco**(**n**) has the following format:

2949   **Xenco**(**n**) = "1 xxxxxx"

2950   where,

2951   "1" is the code word
2952   the **sign** bit is not used
2953   "xxxxxx" is the six bit **value** field

2954 The codec equation is described as follows:

```
2955        value = 16 * (Xenco( n ) & 0x3f)
2956        if (value > Xpred( n )) then
2957            Xdeco( n ) = value + 7
2958        else
2959            Xdeco( n ) = value + 8
2960        endif
```

2961 **E.3.3   Decoder for 10–6–10 Data Compression**

2962 Pixels without prediction are decoded using the following formula:

2963   **Xdeco**(**n**) = 16 * **Xenco**(**n**) + 8

2964 Pixels with prediction are decoded using the following formula:

```
2965        if (Xenco( n ) & 0x3e == 0x00) then
2966            use DPCM1
2967        else if (Xenco( n ) & 0x3e == 0x02) then
2968            use DPCM2
2969        else if (Xenco( n ) & 0x3c == 0x04) then
2970            use DPCM3
2971        else if (Xenco( n ) & 0x38 == 0x08) then
2972            use DPCM4
2973        else if (Xenco( n ) & 0x30 == 0x10) then
2974            use DPCM5
2975        else
2976            use PCM
2977        endif
```

2978 **E.3.3.1   DPCM1 for 10–6–10 Decoder**

2979 **Xenco**(**n**) has the following format:

2980   **Xenco**(**n**) = "00000 s"

2981   where,

2982   "00000" is the code word
2983   "s" is the **sign** bit
2984   the **value** field is not used

2985 The codec equation is described as follows:

2986   **Xdeco**(**n**) = **Xpred**(**n**)

2987 **E.3.3.2   DPCM2 for 10–6–10 Decoder**

2988 **Xenco**(**n**) has the following format:

2989   **Xenco**(**n**) = "00001 s"

2990   where,

2991   "00001" is the code word
2992   "s" is the **sign** bit
2993   the **value** field is not used

2994 The codec equation is described as follows:

2995   **sign** = **Xenco**(**n**) & 0x1
2996   **value** = 1
2997   if (**sign** > 0) then
2998   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
2999   else
3000   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3001   endif

3002 **E.3.3.3   DPCM3 for 10–6–10 Decoder**

3003 **Xenco**(**n**) has the following format:

3004   **Xenco**(**n**) = "0001 s x"

3005   where,

3006   "0001" is the code word
3007   "s" is the **sign** bit
3008   "x" is the one bit **value** field

3009 The codec equation is described as follows:

3010   **sign** = **Xenco**(**n**) & 0x2
3011   **value** = 4 * (**Xenco**(**n**) & 0x1) + 3 + 1

```
3012          if (sign > 0) then
3013              Xdeco( n ) = Xpred( n ) - value
3014              if (Xdeco( n ) < 0) then
3015                  Xdeco( n ) = 0
3016              endif
3017          else
3018              Xdeco( n ) = Xpred( n ) + value
3019              if (Xdeco( n ) > 1023) then
3020                  Xdeco( n ) = 1023
3021              endif
3022          endif
```

3023   E.3.3.4   DPCM4 for 10–6–10 Decoder

3024   **Xenco**(**n**) has the following format:

3025   **Xenco**(**n**) = "001 s xx"

3026   where,

3027   "001" is the code word
3028   "s" is the **sign** bit
3029   "xx" is the two bit **value** field

3030   The codec equation is described as follows:

```
3031          sign = Xenco( n ) & 0x4
3032          value = 8 * (Xenco( n ) & 0x3) + 11 + 3
3033          if (sign > 0) then
3034              Xdeco( n ) = Xpred( n ) - value
3035              if (Xdeco( n ) < 0) then
3036                  Xdeco( n ) = 0
3037              endif
3038          else
3039              Xdeco( n ) = Xpred( n ) + value
3040              if (Xdeco( n ) > 1023) then
3041                  Xdeco( n ) = 1023
3042              endif
3043          endif
```

3044   E.3.3.5   DPCM5 for 10–6–10 Decoder

3045   **Xenco**(**n**) has the following format:

3046   **Xenco**(**n**) = "01 s xxx"

3047   where,

3048   "01" is the code word
3049   "s" is the **sign** bit
3050   "xxx" is the three bit **value** field

3051   The codec equation is described as follows:

3052   **sign** = **Xenco**(**n**) & 0x8
3053   **value** = 16 * (**Xenco**(**n**) & 0x7) + 43 + 7
3054   if (**sign** > 0) then
3055   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3056   if (**Xdeco**(**n**) < 0) then
3057   **Xdeco**(**n**) = 0
3058   endif
3059   else
3060   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3061   if (**Xdeco**(**n**) > 1023) then
3062   **Xdeco**(**n**) = 1023
3063   endif
3064   endif

3065   **E.3.3.6   PCM for 10–6–10 Decoder**

3066   **Xenco**(**n**) has the following format:

3067   **Xenco**(**n**) = "1 xxxxx"

3068   where,

3069   "1" is the code word
3070   the **sign** bit is not used
3071   "xxxxx" is the five bit **value** field

3072   The codec equation is described as follows:

3073   **value** = 32 * (**Xenco**(**n**) & 0x1f)
3074   if (**value** > **Xpred**(**n**)) then
3075   **Xdeco**(**n**) = **value** + 15
3076   else
3077   **Xdeco**(**n**) = **value** + 16
3078   endif

3079   **E.3.4   Decoder for 12–8–12 Data Compression**

3080   Pixels without prediction are decoded using the following formula:

3081   **Xdeco**(**n**) = 16 * **Xenco**(**n**) + 8

3082   Pixels with prediction are decoded using the following formula:

3083   if (**Xenco**(**n**) & 0xf0 == 0x00) then
3084   use **DPCM1**
3085   else if (**Xenco**(**n**) & 0xe0 == 0x60) then
3086   use **DPCM2**
3087   else if (**Xenco**(**n**) & 0xe0 == 0x40) then
3088   use **DPCM3**
3089   else if (**Xenco**(**n**) & 0xe0 == 0x20) then
3090   use **DPCM4**
3091   else if (**Xenco**(**n**) & 0xf0 == 0x10) then

3092   use **DPCM5**
3093   else
3094   use **PCM**
3095   endif

3096 **E.3.4.1   DPCM1 for 12–8–12 Decoder**

3097 **Xenco**(**n**) has the following format:

3098   **Xenco**(**n**) = "0000 s xxx"

3099   where,

3100   "0000" is the code word
3101   "s" is the **sign** bit
3102   "xxx" is the three bit **value** field

3103 The codec equation is described as follows:

3104   **sign** = **Xenco**(**n**) & 0x8
3105   **value** = **Xenco**(**n**) & 0x7
3106   if (**sign** > 0) then
3107   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3108   else
3109   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3110   endif

3111 **E.3.4.2   DPCM2 for 12–8–12 Decoder**

3112 **Xenco**(**n**) has the following format:

3113   **Xenco**(**n**) = "011 s xxxx"

3114   where,

3115   "011" is the code word
3116   "s" is the **sign** bit
3117   "xxxx" is the four bit **value** field

3118 The codec equation is described as follows:

3119   **sign** = **Xenco**(**n**) & 0x10
3120   **value** = 2 * (**Xenco**(**n**) & 0xf) + 8
3121   if (**sign** > 0) then
3122   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3123   else
3124   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3125   endif

3126   **E.3.4.3   DPCM3 for 12–8–12 Decoder**

3127   **Xenco**(**n**) has the following format:

3128   **Xenco**(**n**) = "010 s xxxx"

3129   where,

3130   "010" is the code word
3131   "s" is the **sign** bit
3132   "xxxx" is the four bit **value** field

3133   The codec equation is described as follows:

3134   **sign** = **Xenco**(**n**) & 0x10
3135   **value** = 4 * (**Xenco**(**n**) & 0xf) + 40 + 1
3136   if (**sign** > 0) then
3137   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3138   if (**Xdeco**(**n**) < 0) then
3139   **Xdeco**(**n**) = 0
3140   endif
3141   else
3142   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3143   if (**Xdeco**(**n**) > 4095) then
3144   **Xdeco**(**n**) = 4095
3145   endif
3146   endif

3147   **E.3.4.4   DPCM4 for 12–8–12 Decoder**

3148   **Xenco**(**n**) has the following format:

3149   **Xenco**(**n**) = "001 s xxxx"

3150   where,

3151   "001" is the code word
3152   "s" is the **sign** bit
3153   "xxxx" is the four bit **value** field

3154   The codec equation is described as follows:

3155   **sign** = **Xenco**(**n**) & 0x10
3156   **value** = 8 * (**Xenco**(**n**) & 0xf) + 104 + 3
3157   if (**sign** > 0) then
3158   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3159   if (**Xdeco**(**n**) < 0) then
3160   **Xdeco**(**n**) = 0
3161   endif
3162   else
3163   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3164   if (**Xdeco**(**n**) > 4095)
3165   **Xdeco**(**n**) = 4095
3166   endif
3167   endif

3168   **E.3.4.5   DPCM5 for 12–8–12 Decoder**

3169   **Xenco**(**n**) has the following format:

3170   **Xenco**(**n**) = "0001 s xxx"

3171   where,

3172   "0001" is the code word
3173   "s" is the **sign** bit
3174   "xxx" is the three bit **value** field

3175   The codec equation is described as follows:

3176   **sign** = **Xenco**(**n**) & 0x8
3177   **value** = 16 * (**Xenco**(**n**) & 0x7) + 232 + 7
3178   if (**sign** > 0) then
3179   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3180   if (**Xdeco**(**n**) < 0) then
3181   **Xdeco**(**n**) = 0
3182   endif
3183   else
3184   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3185   if (**Xdeco**(**n**) > 4095) then
3186   **Xdeco**(**n**) = 4095
3187   endif
3188   endif

3189   **E.3.4.6   PCM for 12–8–12 Decoder**

3190   **Xenco**(**n**) has the following format:

3191   **Xenco**(**n**) = "1 xxxxxxx"

3192   where,

3193   "1" is the code word
3194   the **sign** bit is not used
3195   "xxxxxxx" is the seven bit **value** field

3196   The codec equation is described as follows:

3197   **value** = 32 * (**Xenco**(**n**) & 0x7f)
3198   if (**value** > **Xpred**(**n**)) then
3199   **Xdeco**(**n**) = **value** + 15
3200   else
3201   **Xdeco**(**n**) = **value** + 16
3202   endif

3203   **E.3.5   Decoder for 12–7–12 Data Compression**

3204   Pixels without prediction are decoded using the following formula:

3205   **Xdeco**(**n**) = 32 * **Xenco**(**n**) + 16

3206   Pixels with prediction are decoded using the following formula:

```
3207            if (Xenco( n ) & 0x78 == 0x00) then
3208                use DPCM1
3209            else if (Xenco( n ) & 0x78 == 0x08) then
3210                use DPCM2
3211            else if (Xenco( n ) & 0x78 == 0x10) then
3212                use DPCM3
3213            else if (Xenco( n ) & 0x70 == 0x20) then
3214                use DPCM4
3215            else if (Xenco( n ) & 0x70 == 0x30) then
3216                use DPCM5
3217            else if (Xenco( n ) & 0x78 == 0x18) then
3218                use DPCM6
3219            else
3220                use PCM
3221            endif
```

3222   **E.3.5.1   DPCM1 for 12–7–12 Decoder**

3223   **Xenco**(**n**) has the following format:

3224   **Xenco**(**n**) = "0000 s xx"

3225   where,

3226   "0000" is the code word
3227   "s" is the **sign** bit
3228   "xx" is the two bit **value** field

3229   The codec equation is described as follows:

```
3230            sign = Xenco( n ) & 0x4
3231            value = Xenco( n ) & 0x3
3232            if (sign > 0) then
3233                Xdeco( n ) = Xpred( n ) - value
3234            else
3235                Xdeco( n ) = Xpred( n ) + value
3236            endif
```

3237   **E.3.5.2   DPCM2 for 12–7–12 Decoder**

3238   **Xenco**(**n**) has the following format:

3239   **Xenco**(**n**) = "0001 s xx"

3240   where,

3241   "0001" is the code word
3242   "s" is the **sign** bit
3243   "xx" is the two bit **value** field

3244   The codec equation is described as follows:

3245   **sign** = **Xenco**(**n**) & 0x4
3246   **value** = 2 * (**Xenco**(**n**) & 0x3) + 4
3247   if (**sign** > 0) then
3248   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3249   else
3250   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3251   endif

3252   **E.3.5.3   DPCM3 for 12–7–12 Decoder**

3253   **Xenco**(**n**) has the following format:

3254   **Xenco**(**n**) = "0010 s xx"

3255   where,

3256   "0010" is the code word
3257   "s" is the **sign** bit
3258   "xx" is the two bit **value** field

3259   The codec equation is described as follows:

3260   **sign** = **Xenco**(**n**) & 0x4
3261   **value** = 4 * (**Xenco**(**n**) & 0x3) + 12 + 1
3262   if (**sign** > 0) then
3263   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3264   if (**Xdeco**(**n**) < 0) then
3265   **Xdeco**(**n**) = 0
3266   endif
3267   else
3268   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3269   if (**Xdeco**(**n**) > 4095) then
3270   **Xdeco**(**n**) = 4095
3271   endif
3272   endif

3273   **E.3.5.4   DPCM4 for 12–7–12 Decoder**

3274   **Xenco**(**n**) has the following format:

3275   **Xenco**(**n**) = "010 s xxx"

3276   where,

3277   "010" is the code word
3278   "s" is the **sign** bit
3279   "xxx" is the three bit **value** field

3280   The codec equation is described as follows:

3281   **sign** = **Xenco**(**n**) & 0x8
3282   **value** = 8 * (**Xenco**(**n**) & 0x7) + 28 + 3

```
3283        if (sign > 0) then
3284            Xdeco( n ) = Xpred( n ) - value
3285            if (Xdeco( n ) < 0) then
3286                Xdeco( n ) = 0
3287            endif
3288        else
3289            Xdeco( n ) = Xpred( n ) + value
3290            if (Xdeco( n ) > 4095) then
3291                Xdeco( n ) = 4095
3292            endif
3293        endif
```

3294   **E.3.5.5   DPCM5 for 12–7–12 Decoder**

3295   **Xenco**(**n**) has the following format:

3296   **Xenco**(**n**) = "011 s xxx"

3297   where,

3298   "011" is the code word
3299   "s" is the **sign** bit
3300   "xxx" is the three bit **value** field

3301   The codec equation is described as follows:

```
3302        sign = Xenco( n ) & 0x8
3303        value = 16 * (Xenco( n ) & 0x7) + 92 + 7
3304        if (sign > 0) then
3305            Xdeco( n ) = Xpred( n ) - value
3306            if (Xdeco( n ) < 0) then
3307                Xdeco( n ) = 0
3308            endif
3309        else
3310            Xdeco( n ) = Xpred( n ) + value
3311            if (Xdeco( n ) > 4095) then
3312                Xdeco( n ) = 4095
3313            endif
3314        endif
```

3315   **E.3.5.6   DPCM6 for 12–7–12 Decoder**

3316   **Xenco**(**n**) has the following format:

3317   **Xenco**(**n**) = "0011 s xx"

3318   where,

3319   "0011" is the code word
3320   "s" is the **sign** bit
3321   "xx" is the two bit **value** field

3322   The codec equation is described as follows:

3323   **sign** = **Xenco**(**n**) & 0x4
3324   **value** = 32 * (**Xenco**(**n**) & 0x3) + 220 + 15
3325   if (**sign** > 0) then
3326   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3327   if (**Xdeco**(**n**) < 0) then
3328   **Xdeco**(**n**) = 0
3329   endif
3330   else
3331   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3332   if (**Xdeco**(**n**) > 4095) then
3333   **Xdeco**(**n**) = 4095
3334   endif
3335   endif

E.3.5.7   PCM for 12–7–12 Decoder

3337   **Xenco**(**n**) has the following format:

3338   **Xenco**(**n**) = "1 xxxxxx"

3339   where,

3340   "1" is the code word
3341   the **sign** bit is not used
3342   "xxxxxx" is the six bit **value** field

3343   The codec equation is described as follows:

3344   **value** = 64 * (**Xenco**(**n**) & 0x3f)
3345   if (**value** > **Xpred**(**n**)) then
3346   **Xdeco**(**n**) = **value** + 31
3347   else
3348   **Xdeco**(**n**) = **value** + 32
3349   endif

E.3.6   Decoder for 12–6–12 Data Compression

3351   Pixels without prediction are decoded using the following formula:

3352   **Xdeco**(**n**) = 64 * **Xenco**(**n**) + 32

3353   Pixels with prediction are decoded using the following formula:

3354   if (**Xenco**(**n**) & 0x3c == 0x00) then
3355   use **DPCM1**
3356   else if (**Xenco**(**n**) & 0x3c == 0x04) then
3357   use **DPCM3**
3358   else if (**Xenco**(**n**) & 0x38 == 0x10) then
3359   use **DPCM4**
3360   else if (**Xenco**(**n**) & 0x3c == 0x08) then
3361   use **DPCM5**
3362   else if (**Xenco**(**n**) & 0x38 == 0x18) then

```
3363              use DPCM6
3364          else if (Xenco( n ) & 0x3c == 0x0c) then
3365              use DPCM7
3366          else
3367              use PCM
3368          endif
```

3369   Note: **DPCM2** is not used.

E.3.6.1   DPCM1 for 12–6–12 Decoder

3371   **Xenco**(**n**) has the following format:

3372   **Xenco**(**n**) = "0000 s x"

3373   where,

3374   "0000" is the code word
3375   "s" is the **sign** bit
3376   "x" is the one bit **value** field

3377   The codec equation is described as follows:

```
3378        sign = Xenco( n ) & 0x2
3379        value = Xenco( n ) & 0x1
3380        if (sign > 0) then
3381            Xdeco( n ) = Xpred( n ) - value
3382        else
3383            Xdeco( n ) = Xpred( n ) + value
3384        endif
```


E.3.6.2   DPCM3 for 12–6–12 Decoder

3386   **Xenco**(**n**) has the following format:

3387   **Xenco**(**n**) = "0001 s x"

3388   where,

3389   "0001" is the code word
3390   "s" is the **sign** bit
3391   "x" is the one bit **value** field

3392   The codec equation is described as follows:

```
3393        sign = Xenco( n ) & 0x2
3394        value = 4 * (Xenco( n ) & 0x1) + 2 + 1
3395        if (sign > 0) then
3396            Xdeco( n ) = Xpred( n ) - value
3397            if (Xdeco( n ) < 0) then
3398                Xdeco( n ) = 0
3399            endif
3400        else
3401            Xdeco( n ) = Xpred( n ) + value
```

```
3402              if (Xdeco( n ) > 4095) then
3403                    Xdeco( n ) = 4095
3404              endif
3405          endif
```

3406   E.3.6.3   DPCM4 for 12–6–12 Decoder

3407   **Xenco**(**n**) has the following format:

3408   **Xenco**(**n**) = "010 s xx"

3409   where,

3410   "010" is the code word
3411   "s" is the **sign** bit
3412   "xx" is the two bit **value** field

3413   The codec equation is described as follows:

```
3414          sign = Xenco( n ) & 0x4
3415          value = 8 * (Xenco( n ) & 0x3) + 10 + 3
3416          if (sign > 0) then
3417              Xdeco( n ) = Xpred( n ) - value
3418              if (Xdeco( n ) < 0) then
3419                    Xdeco( n ) = 0
3420              endif
3421          else
3422              Xdeco( n ) = Xpred( n ) + value
3423              if (Xdeco( n ) > 4095) then
3424                    Xdeco( n ) = 4095
3425              endif
3426          endif
```

3427   E.3.6.4   DPCM5 for 12–6–12 Decoder

3428   **Xenco**(**n**) has the following format:

3429   **Xenco**(**n**) = "0010 s x"

3430   where,

3431   "0010" is the code word
3432   "s" is the **sign** bit
3433   "x" is the one bit **value** field

3434   The codec equation is described as follows:

```
3435          sign = Xenco( n ) & 0x2
3436          value = 16 * (Xenco( n ) & 0x1) + 42 + 7
3437          if (sign > 0) then
3438              Xdeco( n ) = Xpred( n ) - value
3439              if (Xdeco( n ) < 0) then
3440                    Xdeco( n ) = 0
3441              endif
```

3442	else
3443	**Xdeco**(**n**) = **Xpred**(**n**) + **value**
3444	if (**Xdeco**(**n**) > 4095) then
3445	**Xdeco**(**n**) = 4095
3446	endif
3447	endif

3448 **E.3.6.5   DPCM6 for 12–6–12 Decoder**

3449 **Xenco**(**n**) has the following format:

3450   **Xenco**(**n**) = "011 s xx"

3451   where,

3452   "011" is the code word
3453   "s" is the **sign** bit
3454   "xx" is the two bit **value** field

3455 The codec equation is described as follows:

3456   **sign** = **Xenco**(**n**) & 0x4
3457   **value** = 32 * (**Xenco**(**n**) & 0x3) + 74 + 15
3458   if (**sign** > 0) then
3459   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3460   if (**Xdeco**(**n**) < 0) then
3461   **Xdeco**(**n**) = 0
3462   endif
3463   else
3464   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3465   if (**Xdeco**(**n**) > 4095) then
3466   **Xdeco**(**n**) = 4095
3467   endif
3468   endif

3469 **E.3.6.6   DPCM7 for 12–6–12 Decoder**

3470 **Xenco**(**n**) has the following format:

3471   **Xenco**(**n**) = "0011 s x"

3472   where,

3473   "0011" is the code word
3474   "s" is the **sign** bit
3475   "x" is the one bit **value** field

3476 The codec equation is described as follows:

3477   **sign** = **Xenco**(**n**) & 0x2
3478   **value** = 64 * (**Xenco**(**n**) & 0x1) + 202 + 31
3479   if (**sign** > 0) then
3480   **Xdeco**(**n**) = **Xpred**(**n**) - **value**
3481   if (**Xdeco**(**n**) < 0) then

```
3482              Xdeco( n ) = 0
3483          endif
3484      else
3485          Xdeco( n ) = Xpred( n ) + value
3486          if (Xdeco( n ) > 4095) then
3487              Xdeco( n ) = 4095
3488          endif
3489      endif
```

E.3.6.7   PCM for 12–6–12 Decoder

Xenco(n) has the following format:

   Xenco(n) = "1 xxxxx"

   where,

       "1" is the code word
       the **sign** bit is not used
       "xxxxx" is the five bit **value** field

The codec equation is described as follows:

```
value = 128 * (Xenco( n ) & 0x1f)
if (value > Xpred( n )) then
    Xdeco( n ) = value + 63
else
    Xdeco( n ) = value + 64
endif
```

3505 Annex F JPEG Interleaving (informative)

3506   This annex illustrates how the standard features of the CSI-2 protocol should be used to interleave
3507   (multiplex) JPEG image data with other types of image data, e.g. RGB565 or YUV422, without requiring a
3508   custom JPEG format such as JPEG8.

3509   The Virtual Channel Identifier and Data Type value in the CSI-2 Packet Header provide simple methods of
3510   interleaving multiple data streams or image data types at the packet level. Interleaving at the packet level
3511   minimizes the amount of buffering required in the system.

3512   The Data Type value in the CSI-2 Packet Header should be used to multiplex different image data types at
3513   the CSI-2 transmitter and de-multiplex the data types at the CSI-2 receiver.

3514   The Virtual Channel Identifier in the CSI-2 Packet Header should be used to multiplex different data
3515   streams (channels) at the CSI-2 transmitter and de-multiplex the streams at the CSI-2 receiver.

3516   The main difference between the two interleaving methods is that images with different Data Type values
3517   within the same Virtual Channel use the same frame and line synchronization information, whereas
3518   multiple Virtual Channels (data streams) each have their own independent frame and line synchronization
3519   information and thus potentially each channel may have different frame rates.

3520   Since the predefined Data Type values represent only YUV, RGB and RAW data types, one of the User
3521   Defined Data Type values should be used to represent JPEG image data.

3522   Figure 144 illustrates interleaving JPEG image data with YUV422 image data using Data Type values.

3523   Figure 145 illustrates interleaving JPEG image data with YUV422 image data using both Data Type values
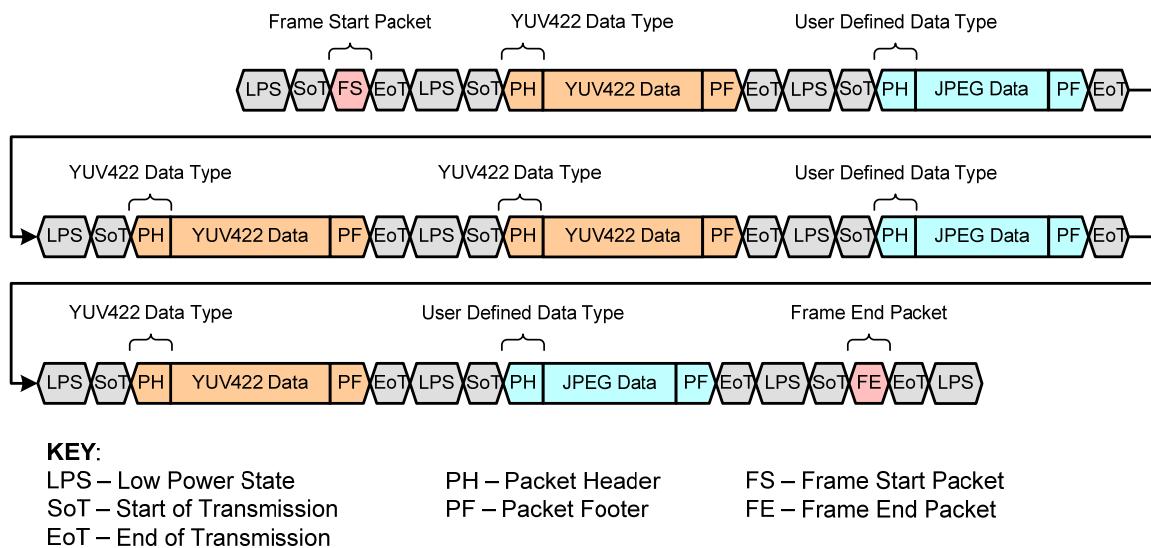3524   and Virtual Channel Identifiers.

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

3527   **Figure 144 Data Type Interleaving: Concurrent JPEQ and YUV Image Data**

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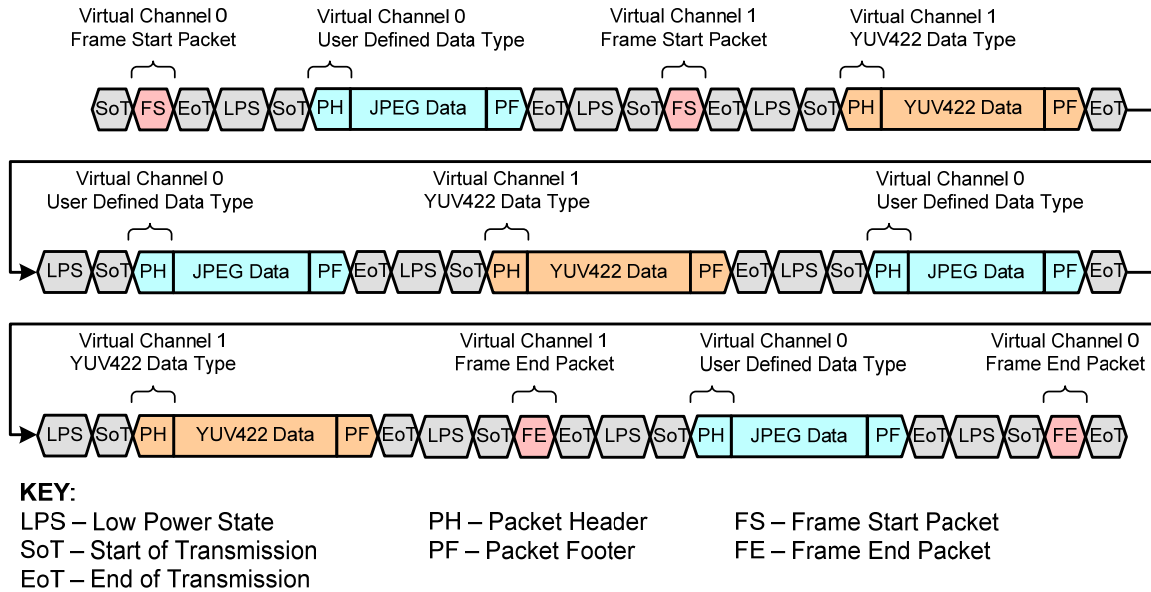EoT – End of Transmission

3528

3529   **Figure 145 Virtual Channel Interleaving: Concurrent JPEQ and YUV Image Data**

3530 Both Figure 144 and Figure 145 can be similarly extended to the interleaving of JPEG image data with any
3531 other type of image data, e.g. RGB565.

3532 Figure 146 illustrates the use of Virtual Channels to support three different JPEG interleaving usage cases:

3533 • Concurrent JPEG and YUV422 image data.

3534 • Alternating JPEG and YUV422 output - one frame JPEG, then one frame YUV

3535 • Streaming YUV22 with occasional JPEG for still capture

3536 Again, these examples could also represent interleaving JPEG data with any other image data type.

Use Case 1: Concurrent JPEG output with YUV data
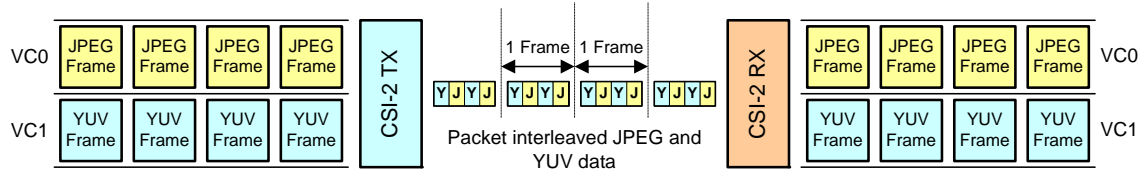
Use Case 2: Alternating JPEG and YUV output – one frame JPEG, then one frame YUV
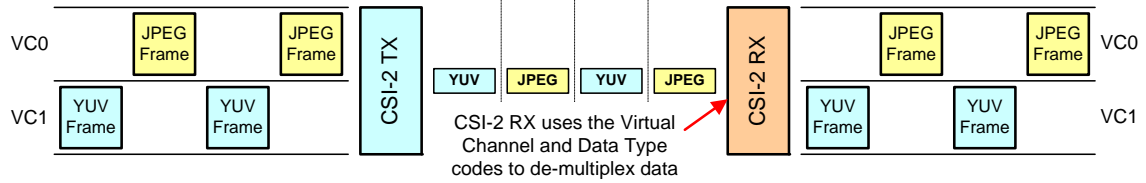
Use Case 3: Streaming YUV with occasional JPEG still capture

3537
3538
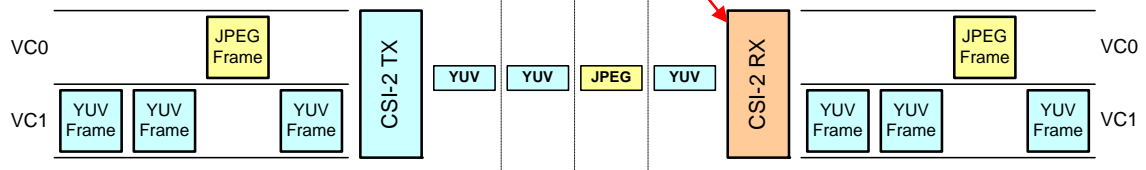
Figure 146 Example JPEG and YUV Interleaving Use Cases

3540