Aurora Protocol Specification

SP002 (v1.3) June 21, 2004

XILINX®

The following table shows the revision history for this document.

	Version	Revision
10/16/02	1.0	Initial Xilinx Release.
02/14/03	1.1	Added Flow Control section. Edited text and figures for clarification.
10/17/03	1.2	Miscellaneous edits to text and figures throughout. Clarifications to State Machine Conventions in the preface, and Section 1 Introduction and Overview. Added Overview to Section 2 Data Transmission and Reception. Added sections 3.1.2 Native Flow Control Latency, and 3.1.4 User Flow Control Operation, and added Table 3-2 user flow control SIZE Encoding table to Section 3 Flow Control. Corrected Lane Initialization procedure, and Channel Verification in Section 4 Initialization and Error Handling. Additions to section 5.4.8 Idle Sequence, and section 5.4.11 Multi-Lane Striping and Transmission Rules in Section 5 PCS and PMA Layers. Corrections to Section 6 Electrical Specification.
06/21/04	1.3	Added 64B/66B feature.

Table of Contents

About This Specification

Conventions

This document uses the following conventions.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
| |	Used to denote columns	
CAPS	A word in all CAPITAL letters either refers to a state in a state machine diagram, a function, or a transaction type	State0
Italic font	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
	Italics are also used for clarification, the first time a new function is introduced	The 8B/10B code uses *disparity* to keep...
REG[FIELD]	Abbreviations or acronyms for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets.	REG[11:14]

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current chapter (or file) or in another chapter (or file) in the current document	See the section "Additional Resources" for details. See "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-II Handbook.*
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Numerical

Convention	Meaning or Use	Example
n	A decimal value	
[*n:m*]	Used to express a numerical range from n to m	
x	Unknown value	
z	High impedance	

Values of Literals

Literals are represented by specifying three of their properties as listed and shown in Figure P-1 and in Table P-1, page 11 and Table P-2, page 11:

1. Width in bits
2. Radix (Base)
3. Value

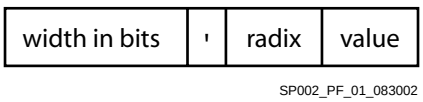


Figure P-1: **Properties of Literals**

Table P-1 shows the Radix specifics:

Table P-1: **Radix Specifics of Literals**

Radix Specifier	Radix
b	Binary
d	Decimal
h	Hexadecimal
o	Octal

All values are extended with zero except those with x or z in the most significant place; they extend with x or z respectively. A list of examples is shown in Table P-2:

Table P-2: **Examples of Extended Values**

Number	Value	Comment
8′b0	00000000	An 8-bit binary number with value of zero. (Zero extended to get 8 bits.)
8′bx	xxxxxxxx	An 8-bit binary number with value unknown. (x extended to get 8 bits.)
8′blx	0000001x	An 8-bit binary number with value of 2 or 3, depending on the value of x.
8′b0x	0000000x	An 8-bit binary number with value of 0 or 1, depending on the value of x.
8′hx	xxxxxxxx	An 8-bit hexadecimal number with value unknown. (x extended to get 8 bits.)
8′hzx	zzzzxxxx	An 8-bit hexadecimal number with the upper four bits not driven and the lower four bits unknown.
8′b1	00000001	An 8-bit binary number with value of one.
8′hzl	zzzz0001	An 8-bit hexadecimal number with the upper four bits not driven and the lower four bits having value one.
8′x1	xxxxxxx1	An 8-bit binary number that is odd.
8′bx0	xxxxxxx0	An 8-bit binary number that is even.
8′hz	zzzzzzzz	An 8-bit hexadecimal number with value not driven. (z extended to get 8 bits.)
8′h0z	0000zzzz	An 8-bit hexadecimal number with upper nibble specified and the lower not driven.
11′d*n*	*n*	An 11-bit decimal number with value *n*.
6′h*n*	*n*	A 6-bit hexadecimal number with value *n*.
w′b101	. . . 101	A binary number with value 5 and an unknown width.

State Diagram Conventions

This section describes the conventions used in the state diagrams for this document. The numbered sections correspond to the call-outs shown in the state machine diagram in Figure 2.

States

1. A state is represented by a rectangle.
2. The name of the state is indicated in bold.

State Transitions

3. State transition is indicated by an arrow annotated in italics.

State Machine Outputs

Outputs are shown in plain text. Outputs can be shown inside of state rectangles or can be part of the annotation associated with a transition arrow. If a signal is not listed in a state rectangle or on a transition arrow, its value at that time is 0 (not asserted). If a registered output does not appear in the state rectangle or transition arrow annotation, then its value is unchanged from the previous value.

Output Types

Outputs are divided into three classes as shown in the examples below.

4. Asserting control signals:
   - go = 1
   - link reset = 1
5. Register initialization:
   - XYZ Register = 78
   - New Counter = 0
   - xmit = /SP/ (an ordered set)
6. Incrementing or decrementing a register:
   - XYZ Register = XYZ Register + 1
   - New Counter = New Counter - 6

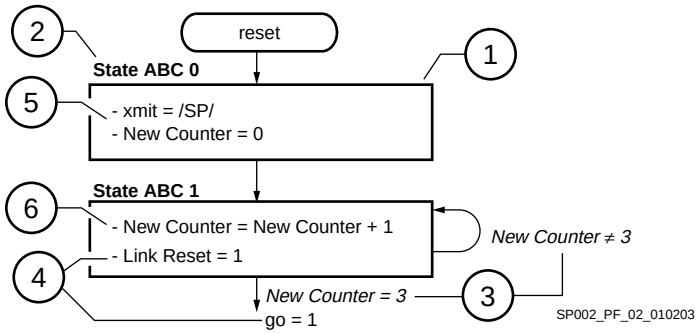


Figure P-2: **State Machine Diagram Conventions**

1 Introduction and Overview

1.1 Introduction

Aurora is a scalable, lightweight, link-layer protocol that can be used to move data point-to-point across one or more high-speed serial *lanes*.

Aurora is protocol independent and can be used to transport industry standard protocols, such as Ethernet and TCP/IP, or proprietary protocols. This allows designers of next-generation communication and computing systems to achieve higher connectivity performance while preserving software infrastructure investment.

While primarily targeted at chip-to-chip and board-to-board applications, Aurora can be used for box-to-box applications with the addition of standard optical interface components.

The Aurora Protocol is an open standard and is available for implementation by anyone without restriction.

1.2 Scope

This section outlines what this specification does and does not define.

The *Aurora Protocol Specification* defines the following:

- Physical layer interface: This includes the electrical levels, the clock encoding, and symbol coding.
- Initialization and error handling: This defines the steps required to prepare *channel partners* for communication across single lane and multi-lane channels. It also describes how the channel partners should behave in the presence of bit errors in the channel.
- Data striping: This describes how data is mapped across a channel of multiple serial lanes.
- Link layer: This describes how the beginning and end of *user protocol data units* (user PDUs) are marked during transmission. This also describes how data pauses may be inserted in data during transmission and how differences in clock rates between the transmitter and receiver are managed.
- Flow control: Aurora defines a link layer flow control mechanism and an expedited mechanism for forwarding higher layer *user flow control messages*.

The Aurora Protocol does *not* define the following; it is assumed that these features will be handled by higher-level protocols:

- Error detection and recovery: Aurora does not define a mechanism for detecting errors in user PDUs, or mechanisms for recovering from them beyond that provided by the 8B/10B or 64B/66B encoding.
- Data switching: Aurora does not define an addressing scheme, therefore cannot support link layer multiplexing or switching.

1.3 Overview

The Aurora Protocol describes the transfer of user data across an *Aurora channel*. An Aurora channel consists of one or more *Aurora lanes*. Each Aurora lane is a full-duplex serial data connection. The devices that communicate across the channel are called *channel partners*. Figure 1-1 illustrates this relationship.

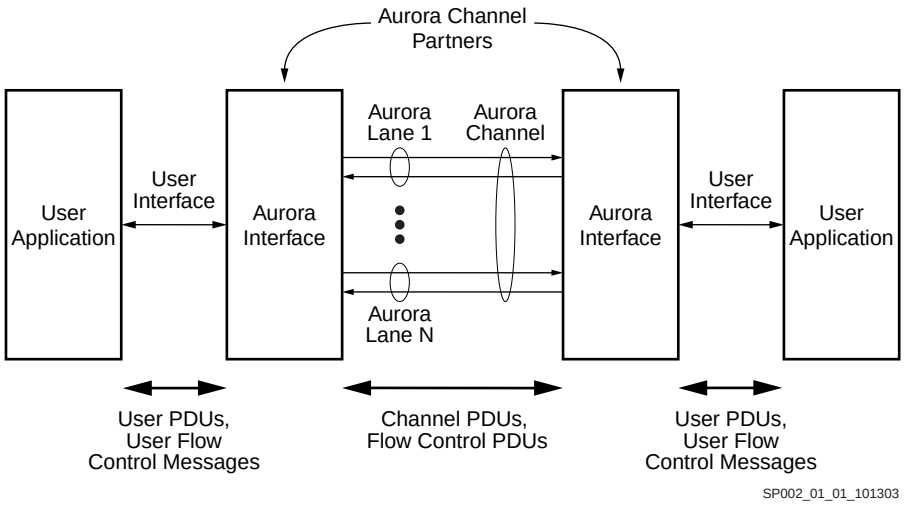


Figure 1-1: **Aurora Channel Overview**

The Aurora interfaces transfer data and control to and from user applications by way of the user interface. The user interface is not defined in this specification.

Data flow consists of the transfer of user PDUs and user flow control messages between the user application and the Aurora interface, and the transfer of channel PDUs, and flow control PDUs across the Aurora channel. User PDUs can be of any length, but their format is not defined in this document. The format of the two types of flow control PDUs are defined in this document.

2 Data Transmission and Reception

2.1 Overview

Section 2.2 8B/10B Data Transmission and Reception and Section 2.3 64B/66B Data Transmission and Reception describe the procedures for transmitting and receiving data across an *Aurora channel*. Aurora 8B/10B uses a symbol-based method, where Aurora 64B/66B uses a block-based method.

2.2 8B/10B Data Transmission and Reception

2.2.1 Symbol-Pairs

The minimum unit of information that is transferred across an Aurora 8B/10B channel is two symbols, called a *symbol-pair*. The information on an Aurora channel (or *lane*) always comprises multiple symbol-pairs.

2.2.2 Data Ordering

Implementations of the Aurora Protocol accept a stream of octets from user applications and transfer them across the Aurora channel as one or more streams of symbol-pairs. In all cases, the ordering of octet streams is preserved. For implementations that have user interfaces that are more than one octet wide, the definition of octet ordering on that interface is implementation specific.

2.2.3 Transmission Scheduling

The following symbol six types of data are transmitted over an Aurora channel:

- **Clock Compensation Sequences:** Sequences of control symbols used to prevent overrun of the receiver due to differences in clock rate between *channel partners*. Clock compensation sequences are described in Section 5.4.9 Clock Compensation, page 54.
- **Initialization Sequences:** Four ordered sets that are used with the *idle sequence* to prepare an Aurora channel for data transmission during initialization. These ordered sets and their use are described in Section 4 Initialization and Error Handling, page 31.
- **Native Flow Control PDUs:** *Link layer flow control* PDUs generated by and interpreted by Aurora interfaces. These PDUs are described in Section 3.1.3 Native Flow Control PDU Format, page 27.

- **User Flow Control PDUs:** Flow control *message*s generated by, and interpreted by user applications, and encapsulated for transmission into user flow control PDUs. These are described in Section 3.1.5 User Flow Control PDU Format, page 28.
- **Channel PDUs:** User PDUs encapsulated for transmission over the Aurora channel. These are described in Section 2.2.4 User PDU Transmission Procedures, page 16.
- **Idle Sequences:** Sequences of control symbols that are transmitted whenever there is no data to send. Idle sequences are described in Section 5.4.8 Idle Sequence, page 52.

In the event that several of the processes that generate these data types are prepared to transmit data at the same time, they are prioritized as shown in Table 2-1.

Table 2-1: **Transmission Priorities**

Data Type	Priority
Clock Compensation Sequences	Highest
Initialization Sequences	
Native Flow Control PDUs	
User Flow Control PDUs	
Channel PDUs	
Idle Sequences	Lowest

2.2.4 User PDU Transmission Procedures

Transmission of *user* PDUs requires the following procedures:

- Padding
- Encapsulation with channel PDU delimiters
- 8B/10B encoding of channel PDU payload
- Serialization and clock encoding

Figure 2-1 illustrates how user PDUs are mapped to channel PDUs by these procedures. Note that this figure illustrates the case where a *pad octet* is required. The remainder of this section describes these procedures in more detail.

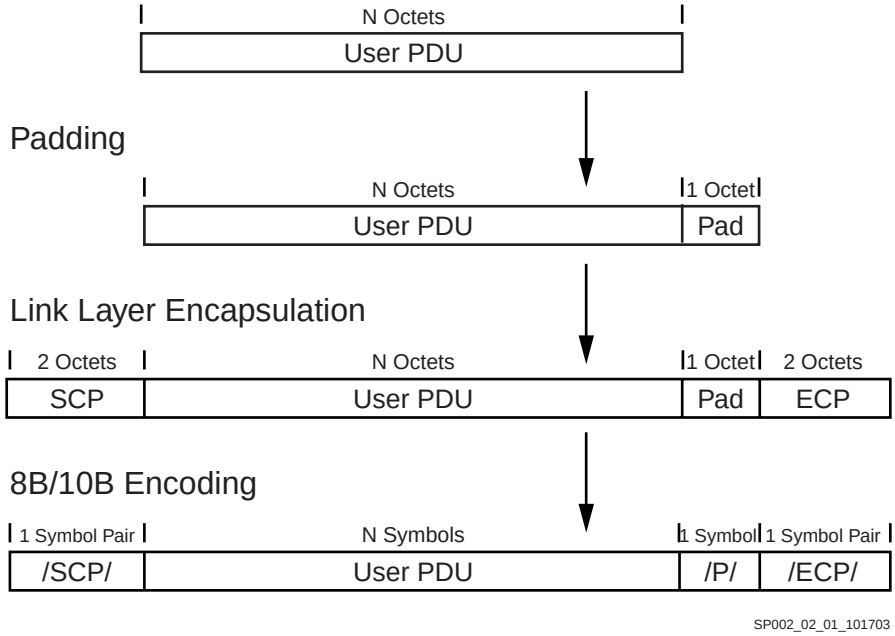


Figure 2-1: **Transmission Procedures**

2.2.4.1 Padding

The Aurora channel requires that all transmissions consist of an even number of symbols. To meet this requirement, all user PDUs that contain an odd number of octets must be padded with a single octet. This pad octet has a value of 0x9C and immediately follows the user PDU.

2.2.4.2 Link Layer Encapsulation

The user PDU is encapsulated with control symbol sequences, called *ordered sets*, to produce the complete channel PDU. These ordered sets demarcate the beginning and end of channel PDUs within the serial data stream. The Aurora Protocol uses an /SCP/ (/K28.2/K27.7/) ordered set to mark the *start of channel* PDUs, and an /ECP/ (/K29.7/K30.7/) ordered set to mark the *end of channel* PDUs. The details of these ordered sequences can be found in Section 5.4.7 Ordered Sets, page 51.

2.2.4.3 8B/10B Encoding

After padding, the resulting data structure is referred to as the *link layer payload*. Prior to transmission the link layer payload is 8B/10B encoded by the *physical coding sublayer* (PCS). All characters, with the exception of the pad octet, are encoded as data symbols. The pad octet is encoded as a /P/ (/K28.4/) control symbol to facilitate stripping at the receiving end. The details of the encoding process are described in Section 5.4.4 8B/10B Encoding, page 50.

2.2.4.4 Serialization and Clock Encoding

After the complete channel PDU has been assembled and encoded, it is serialized for transmission. The details of this process are described in Section 5.4.5 Transmission Order, page 50. The serialized data stream is transmitted in differential *non return to zero* (NRZ) format.

2.2.5 User PDU Reception Procedures

Reception of user PDUs involves the following procedures:

- Deserialization
- 8B/10B decoding of channel PDU payload
- Link layer stripping
- Pad stripping

Figure 2-2 illustrates how user PDUs are mapped to channel PDUs by these procedures. Note that this figure illustrates the case where a pad octet was added to the user PDU data. The remainder of this section describes these procedures in more detail.

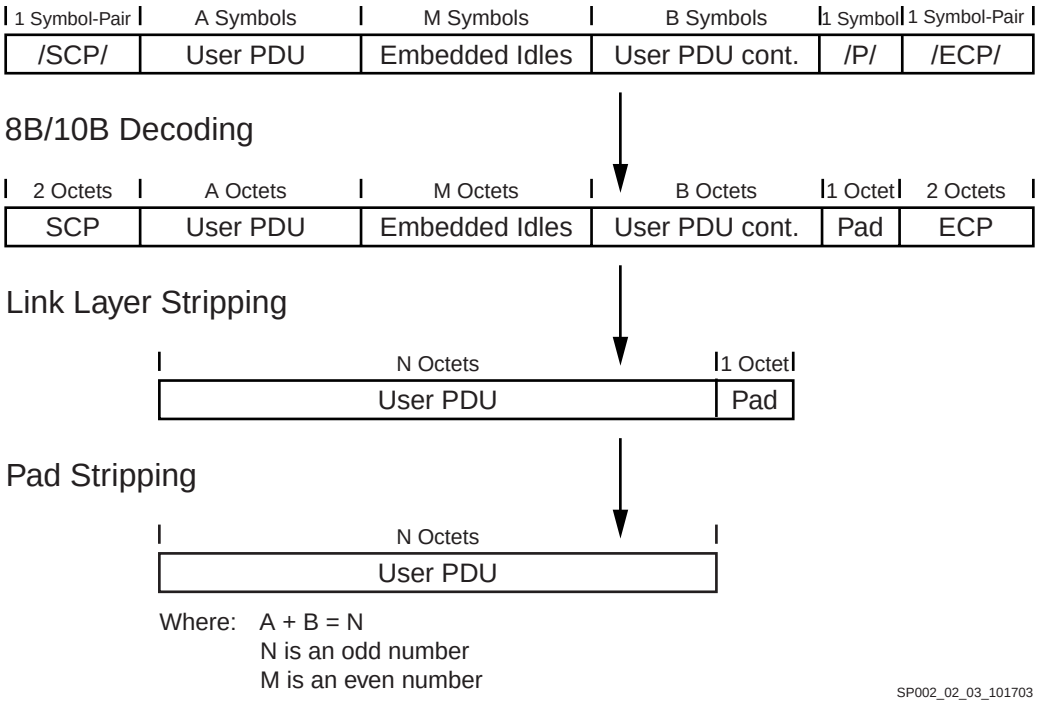


Figure 2-2: **Receive Procedures**

2.2.5.1 Deserialization

The serial data stream is received in differential NRZ format. The receive logic deserializes this data into 10-bit data and control symbols. The details of this process are described in Section 5.4.5 Transmission Order, page 50.

Symbol alignment within the stream is established during the lane initialization procedure described in Section 4.2.1 8B/10B Lane Initialization, page 32 and is not performed again during normal channel operation.

2.2.5.2 8B/10B Decoding

After deserialization, the link layer payload is decoded into a stream of octets. The details of the 8B/10B decoding process are described in Section 5.4.6 8B/10B Decoding, page 51. During the decode process, the presence of a /P/ (/K28.4/) control symbol followed by an/ECP/ (/K29.7/K30.7/) in the data stream must be flagged so that the pad octet can be stripped during the pad stripping procedure.

2.2.5.3 Link Layer Stripping

The link layer stripping procedure removes channel PDU encapsulation and any embedded *idle* ordered sets that may have been inserted during transmission.

Removal of channel PDU encapsulation includes the /SCP/ (/K28.2/K27.7/) ordered set to mark the start of channel PDUs, and an /ECP/ (/K29.7/K30.7/) ordered set to mark the end of channel PDUs. The details of these ordered sequences can be found in Section 5.4.7 Ordered Sets, page 51.

Removal of idle ordered sets involves the removal of /K/ (/K28.5/), /R/ (/K28.0/), and /A/ (/K28.3/) symbols. Any number of these symbols can appear at any point in the channel PDU with the following restrictions:

- An even number of idle symbols must have been inserted
- The start of the idle sequence must begin after an even number of symbols in the channel PDU

2.2.5.4 Pad Stripping

If a /P/ (/K28.4/) control symbol followed by /ECP/ (/K29.7/K30.7/) was detected during the 8B/10B decoding process, a pad octet was post-pended to the user PDU by the transmission process in order to meet channel alignment requirements. This octet, which has a value 0x9C after decoding, shall be stripped from the end of the data stream before passing it to the user application.

2.3 64B/66B Data Transmission and Reception

2.3.1 Transmission Scheduling

There are five block types transmitted over an Aurora 64B/66B channel. They are as follows:

- **Clock Compensation Sequences:** Sequences of control blocks used to prevent overrun of the receiver due to differences in clock rate between channel partners. Clock compensation sequences are described in Section 5.5.4 Clock Compensation Control Block, page 62.
- **Native Flow Control PDUs:** Link layer flow control messages generated by and interpreted by Aurora 64B/66B interfaces. These messages are described in Section 3.1.2 Native Flow Control Message Format, page 19.
- **User Flow Control PDUs:** Higher layer flow control messages generated by and interpreted by user applications. These messages are described in Section 5.5.13 User FC Control Block, page 67.
- **Channel PDUs:** User PDUs encapsulated for transmission over the Aurora 64B/66B channel. These are described in Section 2.2.4 User PDU Transmission Procedures, page 16.
- **Idle Sequences:** Sequences of control blocks that are transmitted whenever there is no data to send. Idle sequences are described in Section 5.5.3 Idle Control Block, page 62.

In the event that several of the processes that generate these data types are prepared to transmit data at the same time, they are prioritized as shown in Table 2-1.

Table 2-2: **Transmission Priorities**

Data Type	Priority
Clock Compensation Sequences	Highest
Native Flow Control PDUs	
User Flow Control PDUs	
Channel PDUs	
Idle Sequences	Lowest

2.3.2 User PDU Transmission Procedures

Transmission of User PDUs requires the following procedures:

- Encapsulation with Channel PDU delimiters
- 64B/66B encoding of Channel PDU payload
- Serialization and clock encoding

Figure 2-3 illustrates how User PDUs are mapped to Channel PDUs by these procedures. The remainder of this section describes these procedures in more detail.

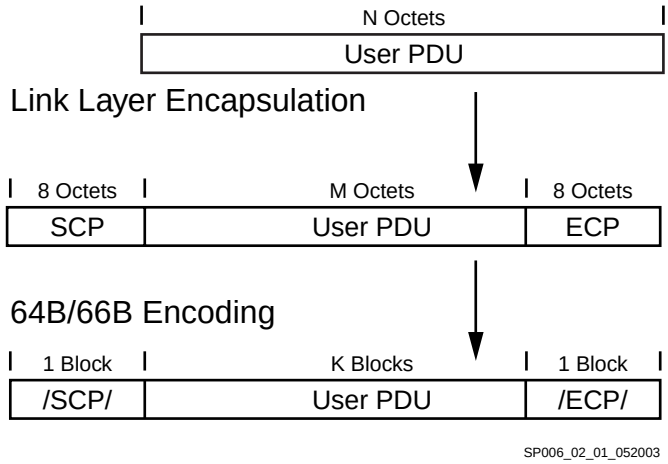


Figure 2-3: **Transmission Procedures**

2.3.2.1 Link Layer Encapsulation

The Link Layer Payload is encapsulated with control characters, to produce the complete Channel PDU. After encapsulation, the resulting data structure is referred to as the Block Code. These block codes demarcate the beginning and end of Channel PDUs within the serial data stream. The Aurora 64B/66B feature uses a Start Channel PDU control block to mark the start of channel PDUs, and an End Channel PDU control block to mark the end of channel PDUs. The Aurora 64B/66B feature requires that all data is transmitted in the form of a block code that consists of 8 bytes of data or mixed data, control, and idle characters. To meet this requirement, the Partial Data control block can be used to transmit data with idles to make up an 8-byte block code. The Partial Data control block can be used between the Start of Channel PDUs and the End of Channel PDUs. The details of these block codes can be found in Section 5.5.2 Block Codes, page 60.

2.3.2.2 64B/66B Encoding

After the Link Layer Payload is encapsulated into block codes, it is 64B/66B coded by the Physical Coding Sublayer (PCS) prior to transmission. The details of the coding process are described in Section 5.5 64B/66B Transmission Code, page 58.

2.3.2.3 Serialization and Clock Encoding

After the complete Channel PDU has been assembled, it is serialized for transmission. The details of this process are described in Section 5.5.1 Transmission and Reception Data Order, page 59. The serialized data stream is transmitted in differential Non Return to Zero (NRZ) format.

2.3.3 User PDU Reception Procedures

Reception of User PDUs involves the following procedures:

- Deserialization
- 64B/66B decoding of Channel PDU payload
- Link layer stripping

Figure 2-4 illustrates how User PDUs are mapped to Channel PDUs by these procedures. The remainder of this section describes these procedures in more detail.

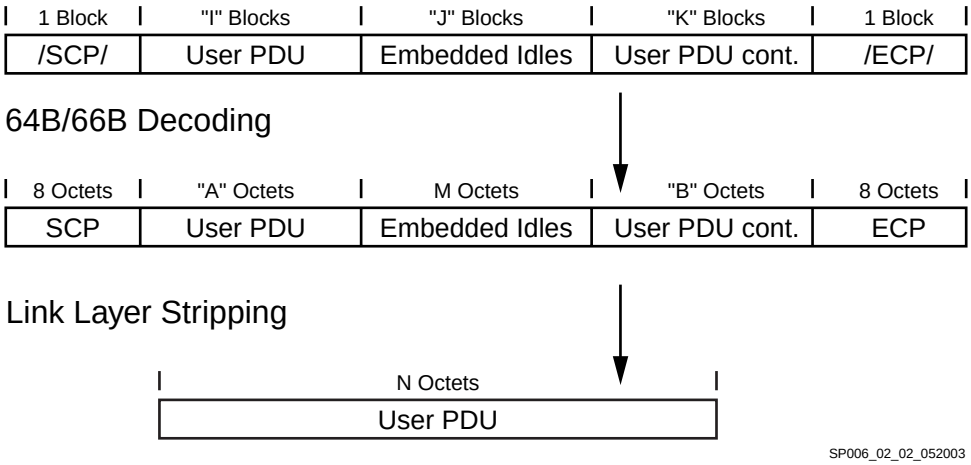


Figure 2-4: **Receive Procedures**

2.3.3.1 Deserialization

The serial data stream is received in differential Non Return to Zero (NRZ) format. The receive logic deserializes this data into 66-bit block codes. The details of this process are described in Section 5.5.1 Transmission and Reception Data Order, page 59.

Block alignment within the stream is established during the lane initialization procedure described in Section 4.3.1 64B/66B Lane Initialization, page 37, and is not performed again during normal channel operation.

2.3.3.2 64B/66B Decoding

After deserialization, the Link Layer Payload is decoded into a stream of octets. The details of the 64B/66B decoding process are described in Section 5.5 64B/66B Transmission Code, page 58.

2.3.3.3 Link Layer Stripping

The link layer stripping procedure removes all control characters, sync header bits, and idle characters that were inserted during transmission. It disassembles all block codes to present only valid data at the user interface. The receiver must also be able to recognize when a pause has been issued. The details of these block codes can be found in Section 5.5.2 Block Codes, page 60.

A summary of the procedure follows:

- Detects the Idle block(s) and informs the interface that there is no useful data.
- Detects the clock compensation blocks and informs the interface that there is no useful data.
- Detects the Start Channel PDU control block, removes the sync header, block type field, byte containing the SPB and any other unused bytes and presents the data at the interface. Since the first unused byte of user data can appear on any position from byte 3 to byte 7, the receiver must be able to handle this movement and properly present data at the user interface.

- The receiver continues to remove the sync header and present the data at the interface until it sees an End Channel PDU control block.
- Upon detecting a Partial Data control block or a Native FC control block, the receiver presents valid bytes of data that are carried by these control blocks, if any, at the interface.
- Upon detecting an End Channel PDU control block, the receiver presents the last bytes of data, if any, at the interface. If the data happened to end on the previous block clock cycle, the user informs the interface that the PDU is complete.
- The receiver processes the next block and starts looking for the next Start Channel PDU control block.
- Upon detecting a Native FC control block, the user must decode the IDLES field and send the appropriate amount of Idle control blocks. If the system was in an XOFF state and an XON is received, the user will restart data transmission. If the system was in an XOFF state and the IDLES field decoded to a value, the user will send the number of idles designed by the field and then restart data transmission.
- Upon detecting a User FC control block, the user will decode the SIZE field and present the appropriate amount of data to the interface. This data must be distinguished from standard data as a user flow control message.

3 Flow Control

3.1 Overview

This chapter describes the optional *flow control* mechanisms supported by Aurora. These mechanisms provide low-latency flow control to prevent data loss due to differences between the rate at which data is sourced and sunk between *channel partners*. Latency is minimized because flow control PDUs can be embedded within *channel* PDUs (see Section 2.2.3 Transmission Scheduling, page 15).

Aurora supports the following two flow control mechanisms:

- **Native Flow Control:** This is a *link-layer* flow control mechanism. Native flow control PDUs are generated by, and interpreted by the Aurora interface.
- **User Flow Control:** This mechanism can be used to implement user-defined flow control schemes at any layer. User flow control *messages* are generated by, and interpreted by the user application. The Aurora interface encapsulates these messages into user flow control PDUs, and provides a low latency mechanism for their transport across the channel.

Figure 3-1, page 26 illustrates how these flow control schemes interrelate.

Native flow control PDUs are typically generated when elasticity storage at the receiver side is being depleted. Note that these PDUs must be generated early enough that the latency between the time that they are issued and when traffic stops does not result in an overflow of elasticity storage.

User flow control messages are forwarded from one user application to the other through the *Aurora channel* and are not interpreted by either Aurora interface.

Some things to note about the flow control mechanisms:

- Assertion of native flow control does not block the forwarding of user flow control PDUs
- User flow control PDUs, once their transmission has started, cannot be interrupted by any other sequence

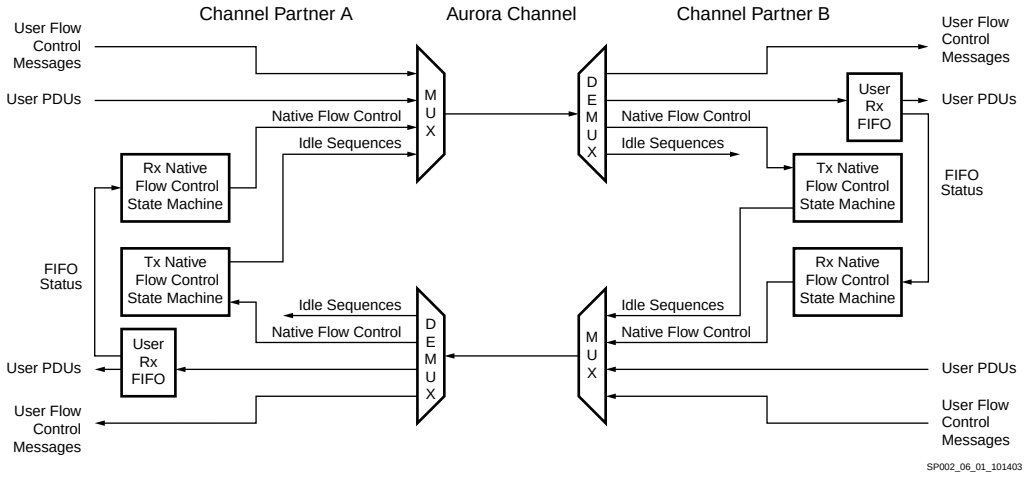


Figure 3-1: **Flow Control Overview**

3.1.1 Native Flow Control Operation

Operation of native flow control is governed by the two state machines shown in Figure 3-1.

The Rx native flow control state machine monitors the state of the Rx FIFO, and generates native flow control PDUs when there is a risk of Rx FIFO overflow. These PDUs request that the channel partner pause the transmission of user PDUs for a specified number of *symbol times*. The format of these native flow control PDUs is described in Section 3.1.3 Native Flow Control PDU Format. The algorithm used to map the state of the FIFO to a specific native flow control PDU is implementation dependent, and is not defined in this specification.

The Tx native flow control state machine responds to native flow control PDUs from the channel partner by pausing the transmission of user PDUs for the requested interval. Note that in addition to *idle sequences*, a pause can include the transmission of native flow control PDUs or user flow control PDUs since neither of these PDUs are stored in the Rx FIFO.

If the Aurora interface is in the process of transmitting a user PDU, the Tx native flow control state machine can respond in one of two ways to native flow control PDUs.

- **Completion Mode:** The user PDU is completed before transmission is paused.
- **Immediate Mode:** The transmission of the user PDU is interrupted by the pause.

All Aurora implementations must be capable of supporting either behavior if they support flow control. The operating mode must be programmable, but this specification does not define how the operating mode is chosen.

3.1.2 Native Flow Control Latency

When operating in *immediate mode*, the maximum round trip delay contribution from the channel partners is defined, in order to bound the depth of the User Rx FIFO. The contribution to latency introduced by the channel is not included in this definition, but must also be taken into account when designing the User Rx FIFO for a particular channel.

For 8B/10B

The round trip delay through the Aurora interfaces between the native flow control PDU request and the first pause arriving at the originating channel partner must not exceed 256 *symbol* times.

For 64B/66B

The round trip delay through the Aurora interfaces between the native flow control PDU request and the first pause arriving at the originating channel partner must not exceed 256 *block* times.

3.1.3 Native Flow Control PDU Format

For 8B/10B

Native flow control PDUs are two octets in length. The first octet is a K28.6 *start of native flow control* (SNF) character and the second octet is a data character called the *command octet*.

The command octet contains the PAUSE field, which specifies the number of *idle characters* the channel partner must send in response to the PDU. Figure 3-2 shows the command octet format, and Table 3-1 shows the encoding of the PAUSE field.

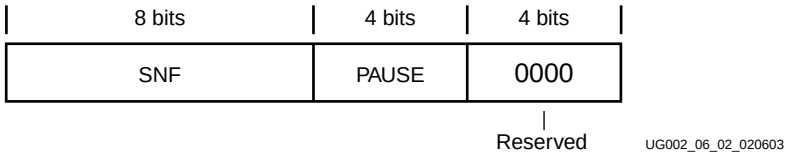


Figure 3-2: **Native Flow Control Command Octet Format**

Table 3-1: **Native Flow Control PAUSE Field Encoding**

PAUSE Field Contents	PAUSE Interval (Symbols)
0000	0 (XON)
0001	2
0010	4
0011	8
0100	16
0101	32
0110	64
0111	128
1000	256

Table 3-1: **Native Flow Control PAUSE Field Encoding *(Continued)***

PAUSE Field Contents	PAUSE Interval (Symbols)
1001 to 1110	Reserved
1111	Infinite (XOFF)

For 64B/66B

Refer to Section 5.5.12 Native FC Control Block, page 65.

3.1.4 User Flow Control Operation

User flow control PDUs are not interruptible by clock compensation sequences, native flow control PDUs, or idle sequences once their transmission has begun. The Aurora Protocol implementations may need to buffer user flow control PDUs to ensure that they are not interrupted, and that the clock compensation rules are met. Any buffering scheme is implementation dependent and is not defined in this specification.

3.1.5 User Flow Control PDU Format

For 8B/10B

User flow control PDUs are 4 to 18 octets in length. The first octet is a K28.4 *start of user flow control* (SUF) character, which is followed by a data character called the *command octet*. This command octet is immediately followed by 2 to 16 octets of the flow control message from the user application.

The K28.4 character is also used as a pad character for channel PDUs. However the pad character can only be followed by a control character, and can never be followed by a data character. This discriminates between a K28.4 used as a pad character and a K28.4 used to mark the start of a user flow control PDU.

Figure 3-3 shows the format of the user flow control PDU. The length of the user flow control message is specified in the SIZE field. The user flow control message size can be any even number of octets from 2 to 16 as shown in Table 3-2, page 29.

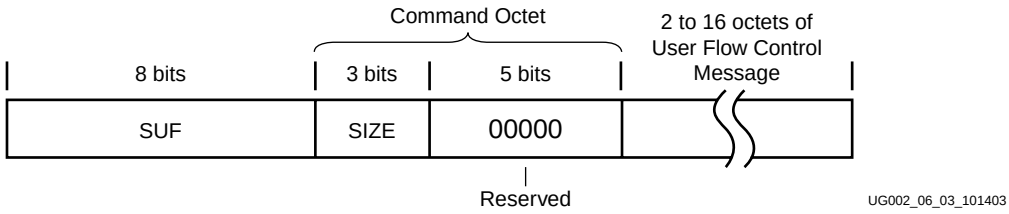


Figure 3-3: **User Flow Control PDU Format**

Table 3-2: **SIZE Encoding**

SIZE Field Contents	User Flow Control Message Size
000	2 octets
001	4 octets
010	6 octets
011	8 octets
100	10 octets
101	12 octets
110	14 octets
111	16 octets

For 64B/66B

Refer to Section 5.5.13 User FC Control Block, page 67.

4 Initialization and Error Handling

4.1 Overview

This chapter describes the initialization procedures needed to prepare an *Aurora channel* for data transmission and reception. Initialization of an Aurora channel is a three-stage process. These stages are:

- **Lane Initialization:** This procedure is performed individually on each lane to activate the link and align the received data to the proper boundaries.
- **Channel Bonding:** This procedure bonds the individual lanes into a single data channel. If the function implemented only uses a single link, channel bonding is not required and will be bypassed. Channel bonding eliminates skew introduced from various sources including the trace lengths, connectors and IC variations.
- **Channel Verification:** This procedure performs any alignment needed to map received data to the user interface and verifies the ability of the channel to transfer valid data.

Figure 4-1 illustrates how these procedures interrelate.

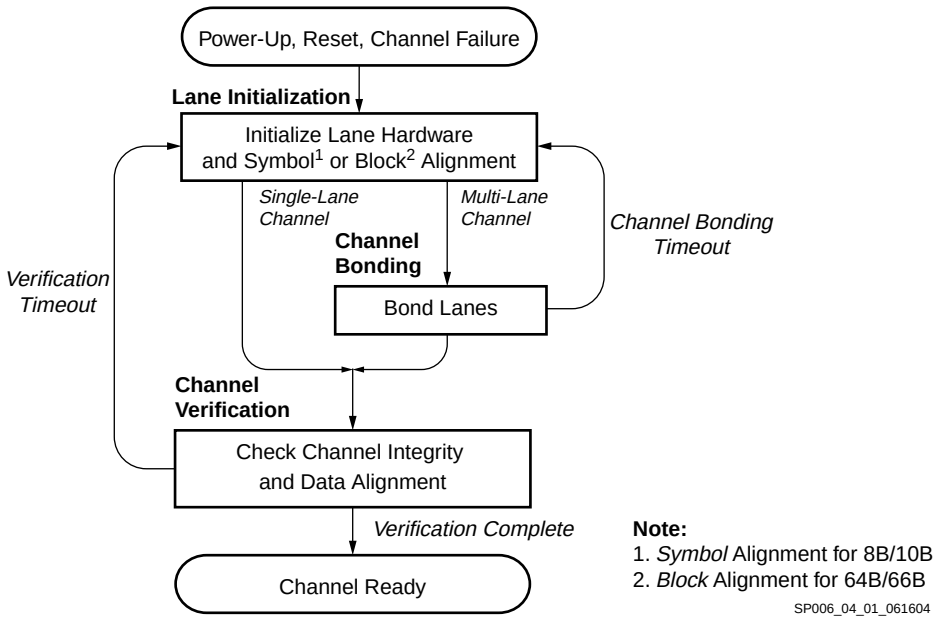


Figure 4-1: **Initialization Overview**

During *channel verification*, all *lanes* must become ready to receive data across the *channel*. When an Aurora interface completes channel verification, it can immediately start to transmit data.

4.2 8B/10B Initialization and Error Handling

4.2.1 8B/10B Lane Initialization

The individual lane initialization procedure synchronizes each lane transceiver with its lane partner upon reset or *channel failure*. A channel failure is defined as any one of the following conditions:

- Excessive channel data errors, as defined in Section 4.2.4 8B/10B Error Handling
- Excessive protocol violations, implementation defined

A state diagram of the lane initialization procedure is shown in Figure 4-2, page 33. See Table 5-1, page 51 for the definition of /SP/, /K/, /SPA/, D10.2, and D12.1. Also, note that D21.5 is the inverse of D10.2 and D19.6 is the inverse of D12.1.

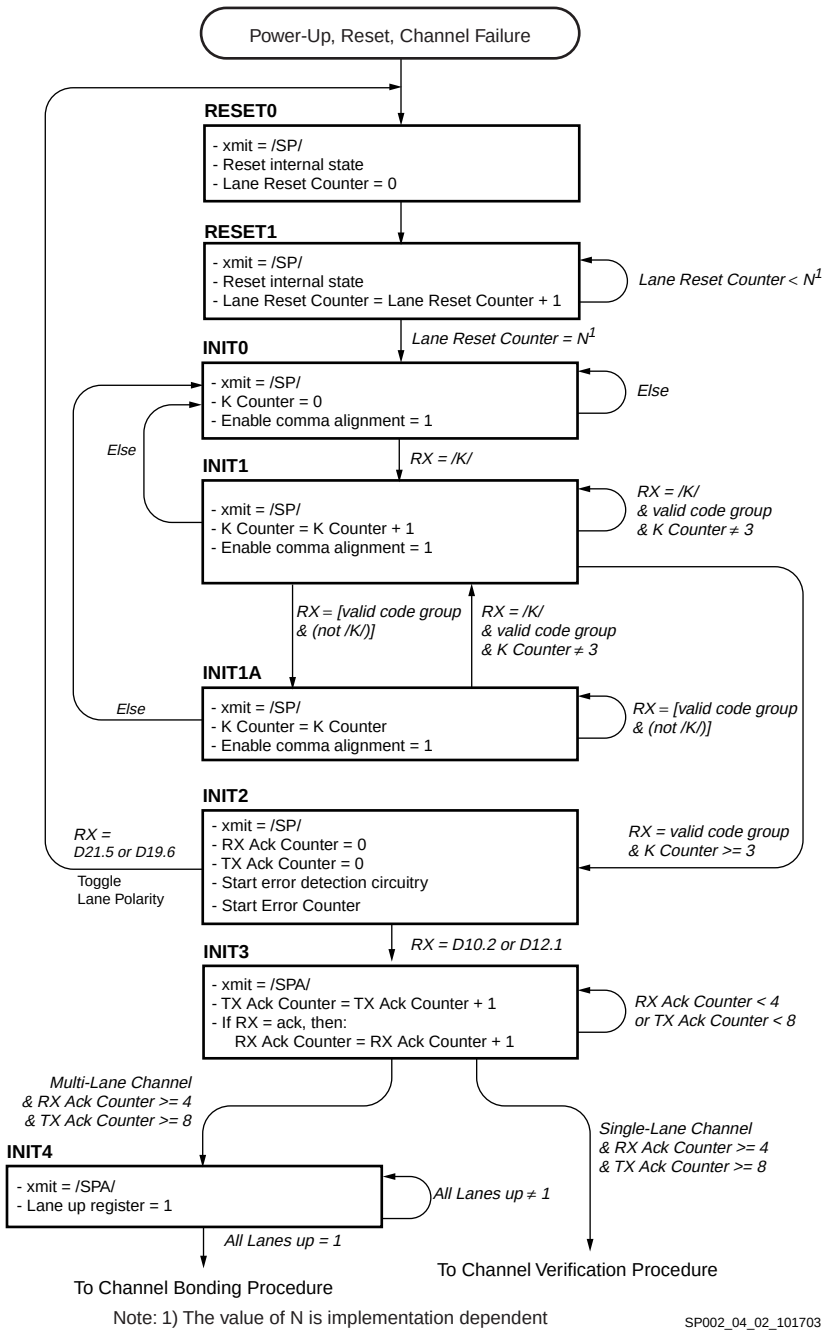


Figure 4-2: **Lane Initialization Procedure**

Note: See State Diagram Conventions, page 12 for conventions used in this diagram.

4.2.2 8B/10B Channel Bonding

The *channel bonding* procedure aligns all data being received by each lane. Channel bonding cannot begin until each lane has completed the lane initialization procedure. Figure 4-3 shows a state diagram of the channel bonding procedure. See Table 5-1, page 51 for the definition of /I/ and /A/.

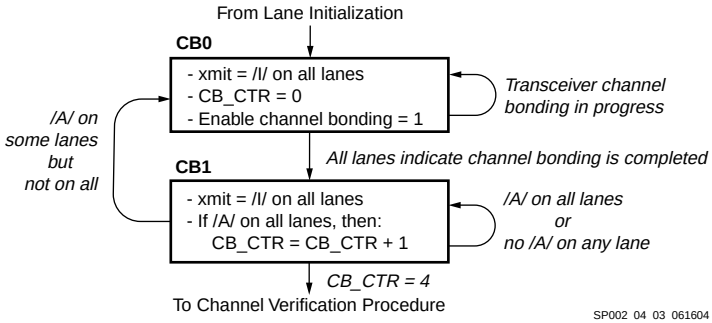


Figure 4-3: **Channel Bonding Procedure**

Note: See State Diagram Conventions, page 12 for conventions used in this diagram.

Channel bonding occurs in two phases. This first phase, which corresponds to State CB0 in Figure 4-3, consists of transceiver specific data alignment. The second phase, which corresponds to State CB1, verifies that the transceivers have aligned correctly and are delivering the /A/ *ordered sets* within the /I/ ordered set at the same time.

4.2.3 8B/10B Channel Verification

The *channel verification* procedure is used to align data across the user interface, and verify channel integrity. It essentially consists of sending the channel verification sequence across all lanes in each direction. Using this known data pattern, the receiving channel partners can correctly align data across the user interface, and verify channel integrity. A diagram of the procedure is shown in Figure 4-4. See Table 5-1, page 51 for the definition of /V/.

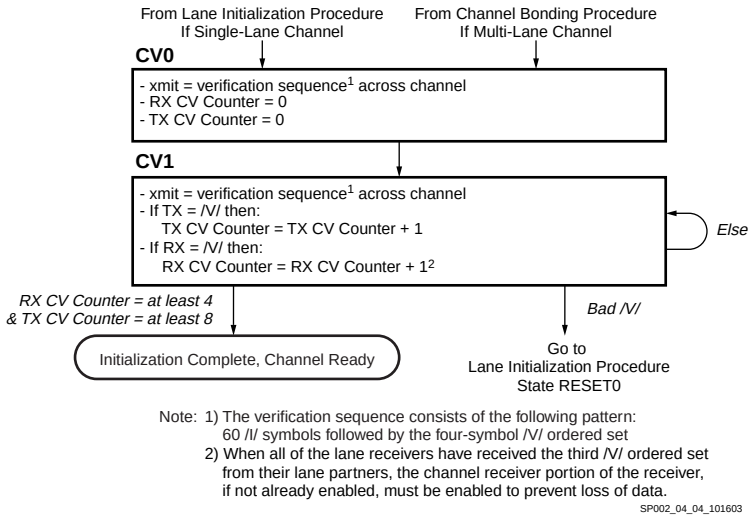


Figure 4-4: **Channel Verification Procedure**

Note: See State Diagram Conventions, page 12 for conventions used in this diagram.

4.2.4 8B/10B Error Handling

During the normal transmission of data, it is possible for the channel to fail. All lanes must be individually monitored for errors. Before the serial transceiver properly byte aligns the data, many errors will be seen on the outputs. Because of this, it is important that none of the error detection circuits is activated until the lane is stable. This stable point is achieved when the K counter reaches 3, as shown in Figure 4-2, page 33. Once the K counter has reached 3, all error detection circuits should be activated.

There are two types of errors: *hard* and *soft*.

A *hard* error occurs when:

- There is an overflow/underflow in either the transmit or receive elastic buffers
- There are too many soft errors, refer to Figure 4-5, page 36
- The channel partner undergoes a reset, indicated by receipt of initialization sequences
- The physical connection between channel partners is broken

Hard errors are considered catastrophic errors, after which the channel should immediately be re-initialized. Any data unit being received when a hard error occurs is corrupt and should be discarded.

Soft errors are typically due to transient bit errors on the lane. Soft errors do not necessarily require the channel to be reset. Aurora does no explicit data error checking; data errors induced by soft errors should be detected and corrected by the user's error handling capability. Examples of soft errors include:

- *Disparity* errors
- *Symbol* errors

In multi-Gbps systems, a bit error can occur in a range between minutes, for a system with poor channel integrity, to days or years for a system with good channel integrity. As a result, Aurora interfaces must implement soft error monitoring logic to verify ongoing channel integrity. This logic consists of a soft error counter and associated control logic for each lane. Figure 4-5, page 36 illustrates the required behavior of the soft error monitoring logic.

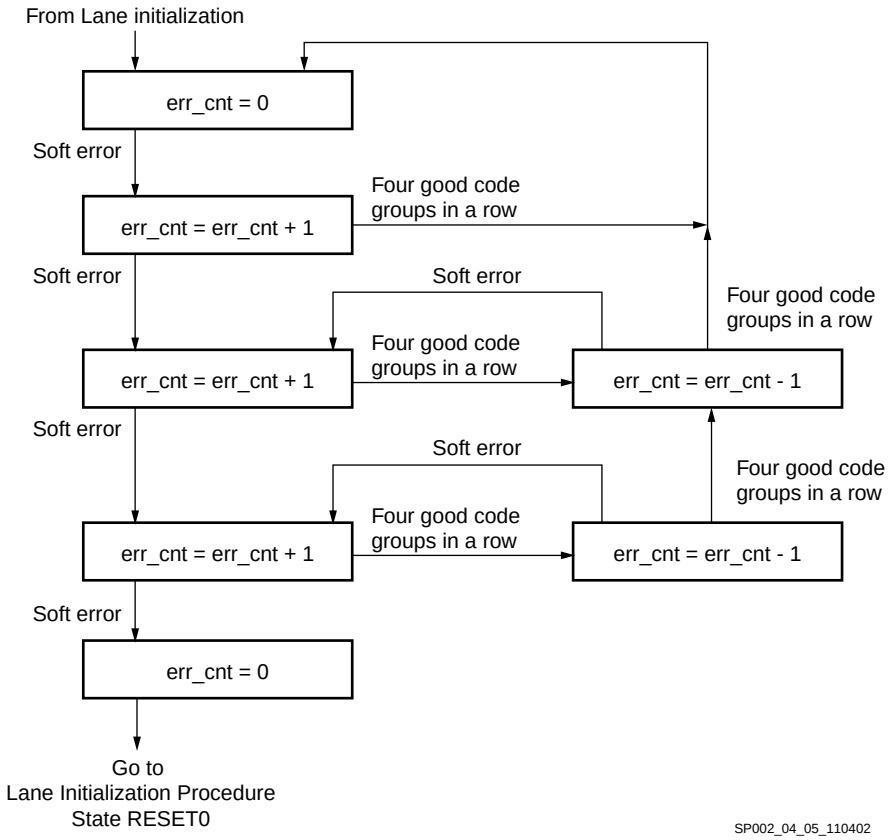


Figure 4-5: **Soft Error Handling**

4.3 64B/66B Initialization and Error Handling

4.3.1 64B/66B Lane Initialization

The individual lane initialization procedure synchronizes each lane transceiver with its lane partner upon reset or channel failure. A channel failure is defined as any one of the following conditions:

- Excessive channel data errors, as defined in Section 4.3.4 64B/66B Error Handling
- Excessive protocol violations, implementation defined

A state diagram of the lane initialization procedure is shown in Figure 4-6. See Table 5-4, page 61 for the definition of 64B/66B block codes.

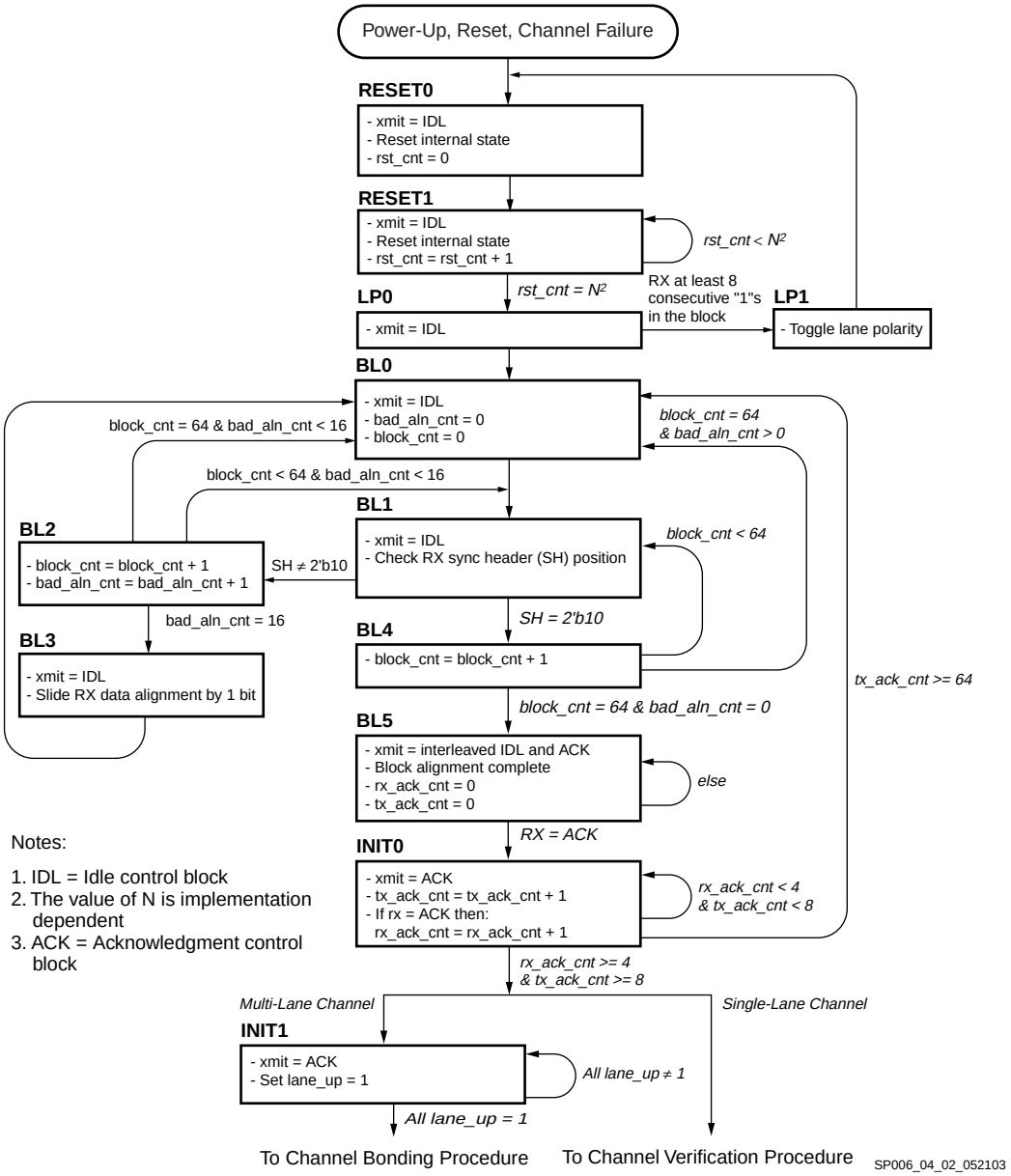


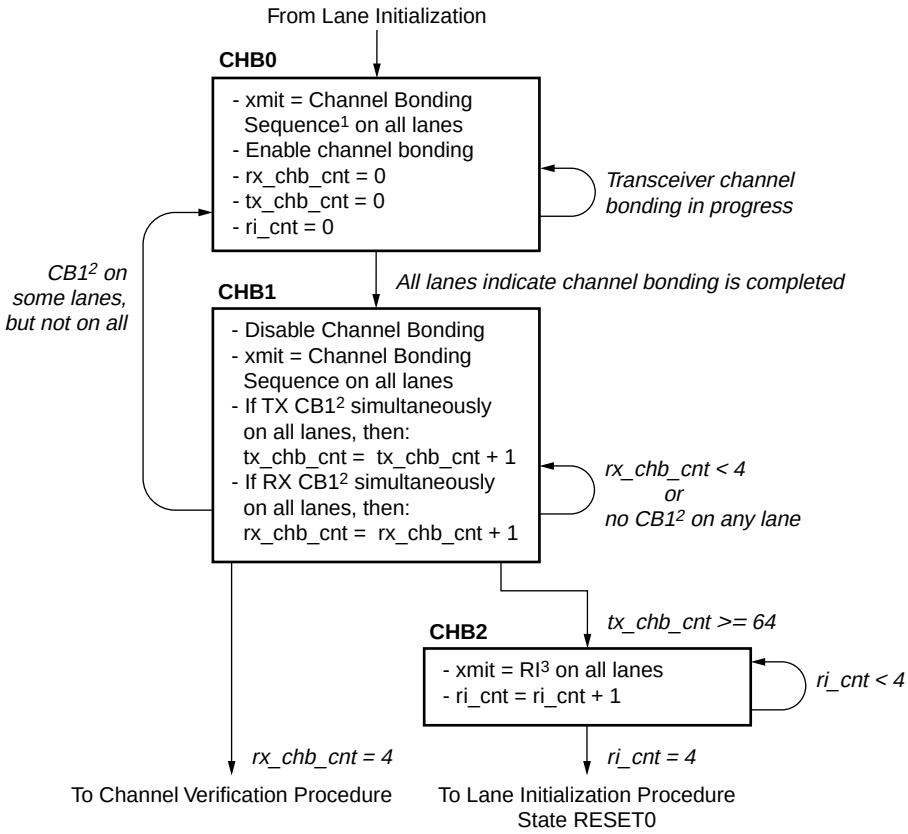
Figure 4-6: **Lane Initialization Procedure**

4.3.1.1 Lane Initialization Procedure

1. Upon reset, power up, or lane initialization restart, the transmitter sends out Idle control blocks for *N* times. The value of *N* is implementation dependent, that is, depending on how long the internal state needs to reset itself.
2. The receiver then checks incoming data for at least eight consecutive ones. If it is true, the receiver will swap its polarity and restart the initialization procedure.
3. The receiver sets the **block_cnt** and **bad_aln_cnt** counters to 0.
4. Upon receipt of enough data to create a single block code, the sync header position is checked to see if it is equal to 2'b10.
   a. If it is, the **block_cnt** is incremented by 1.
   b. If it is not, the **block_cnt** and **bad_aln_cnt** are both incremented by 1.
5. The next block code is checked to see if the sync header position is equal to 2'b10. This procedure continues until either 64 block codes in a row pass the sync header check or **bad_aln_cnt** equal 16.
6. If the **bad_aln_cnt** equals 16, it means that the sync header is in a different position than the one presently being analyzed.
7. The block code should then be shifted and the sampling process should restart after resetting the **block_cnt** and **bad_aln_cnt** counters.
8. The procedure of shifting the received bits is implementation dependent. It is up to the designer of this process to ensure that all bit positions are checked.
9. Once 64 sync headers in a row have been seen without any errors, block alignment is complete.
10. When block alignment is complete, the transmitter starts transmitting the Acknowledgement control blocks interleaved with Idle control blocks.
11. The receiver waits for the reception of the Acknowledgement before it stops transmitting interleaved Acknowledgment and Idle control blocks.
12. A channel cannot stop transmitting the Acknowledgement control block until it has received at least four Acknowledgement control blocks from its channel partners and until it has been transmitting its own Acknowledgement control blocks at least eight times. Once these two requirements are met, lane initialization is complete.

4.3.2 64B/66B Channel Bonding

The channel bonding procedure aligns all data being received by each lane by removing skew imparted by board, IC, and package effects. The algorithm used, will allow greater than 12 ns of skew between channels. Channel bonding cannot begin until every transceiver has completed the lane initialization procedure. Figure 4-7 shows the procedure that controls channel bonding for the receiver and transmitter. A summary of the procedure will be given, but the state machines are the final authority.



Note:
1. Channel Bonding Sequence includes one Channel Bonding 1 control block followed by four Channel Bonding 2 data blocks.
2. CB1 = Channel Bonding 1 control block
3. RI = Restart Initialization control block.

SP006_04_03_061604

Figure 4-7: **Channel Bonding Procedure**

4.3.2.1 Channel Bonding Procedure

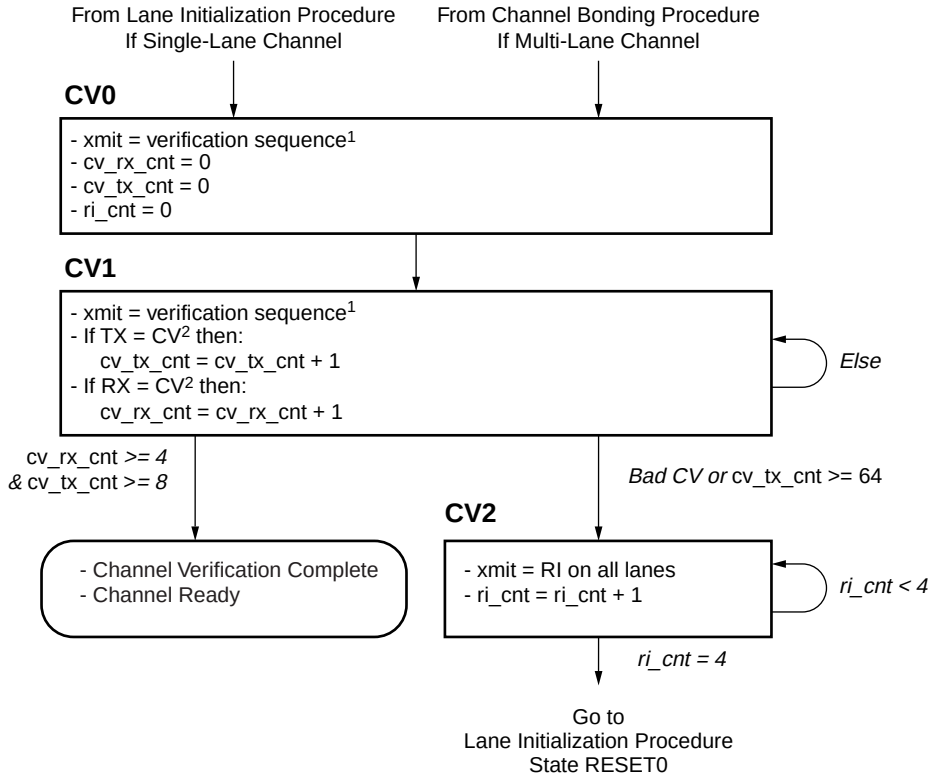
1. Channel bonding will not start until all lanes in the channel have completed lane initialization.
2. Transmit the channel bonding sequence on all lanes, which consists of one Channel Bonding 1 control block and four Channel Bonding 2 control blocks. This makes a pattern of five block codes that will be continuously transmitted.
3. Set the **rx_chb_cnt** and **tx_chb_cnt** to 0.
4. Enable channel bonding.

5. If channel bonding passes on all lanes simultaneously, disable channel bonding and continuously send channel bonding sequence on all lanes.
6. If the Channel Bonding control block is transmitted on all lanes, increment **tx_chb_cnt** by 1.
7. If the Channel Bonding 1 control block is received simultaneously on all lanes, increment **rx_chb_cnt** by 1.
8. If the Channel Bonding 1 control block is not received on all lanes simultaneously, restart the channel bonding procedure.
9. If **rx_chb_cnt** is equal to 4, channel bonding is completed.
10. If 64 Channel Bonding 1 control blocks are transmitted, channel bonding is timed out. Restart the lane initialization procedure by sending four Restart Initialization control blocks to the channel partners.

4.3.3 64B/66B Channel Verification

The channel verification procedure is used to align data transfers across the channel, ensuring that all channel partners can communicate with each other. It essentially consists of sending a small packet, the Channel Verification control blocks, across the channel in each direction. Using this known data pattern, the receiving channel partners can verify channel integrity and correctly align data across the user interface.

Until channel bonding is complete, there is a possibility that alignment can change. After the completion of channel verification, alignment should never change unless an error is induced. A state diagram of the procedure is shown in Figure 4-8, page 41.



Notes:
1. The verification sequence consists of the following pattern: six channel bonding sequences followed by one Channel Verification control block. The channel bonding sequence consists of one Channel Bonding 1 control block and four Channel Bonding 2 data blocks.
2. CV = Channel Verification control block.
3. RI = Restart Initialization control block.

SP006_04_04_041503

Figure 4-8: **Channel Verification Procedure**

4.3.3.1 Channel Verification Procedure

1. The transmitter starts the channel verification procedure by sending six channel bonding sequences, followed by one Channel Verification control block on all lanes.
2. The receiver will look for the receipt of a Channel Verification control block to align the interface. Once aligned, it will stay aligned unless an error occurs.
3. If a bad Channel Verification control block is received, or if verification times out, the transmitter will initiate a reset and restart lane initialization after sending four Restart Initialization control blocks to channel partners.
4. After at least eight Channel Verification control blocks have been sent, and at least four Channel Verification control blocks have been received, the receiver declares the lane up in one of the following ways:
   a. For single-lane designs, the **lane_up** and **channel_up** outputs go high.
   b. For multi-lane designs, the **channel_up** output goes high.

   The channel is now ready to accept user data.

4.3.4 64B/66B Error Handling

During the normal transmission of data, it is possible for the channel to fail. All lanes must be individually monitored for errors. Before the serial transceiver properly aligns the block codes, many errors will be seen on the outputs. Because of this, it is important that none of the error detection circuits is activated until the lane is stable. This stable point is achieved when block alignment is complete, as shown in Figure 4-6, page 37. Once the block alignment has completed, all error detection circuits should be activated.

There are two types of errors: *hard* and *soft*. A hard error can be caused by any of the following conditions:

- Overflow/underflow in either the transmit or receive elastic buffers
- Too many soft errors
- A restart initialization request from the receiver's channel partner.

These are considered catastrophic errors, after which the channel should immediately be reinitialized. The soft errors are typically due to transient bit errors on the lane. Soft errors do not necessarily require the channel to be reset. Aurora does no data error checking; data errors induced by soft errors should be detected and corrected by the user's error handling capability.

Aurora 64B/66B provides for a circuit to detect if the bit error rate gets bad and issue a restart. The bit error rate will be computed by monitoring the sync header during normal operation. The monitoring process is similar to that used during lane initialization. If there are more than two errors every 4,096 block codes in each lane, the bit error will be deemed excessive and lane initialization should begin. In multi-lane implementations, each lane should have its own error detection circuit.

A state machine describing the operation is shown Figure 4-9, page 43. A summary of the procedure is given, but the state machines are the final authority.

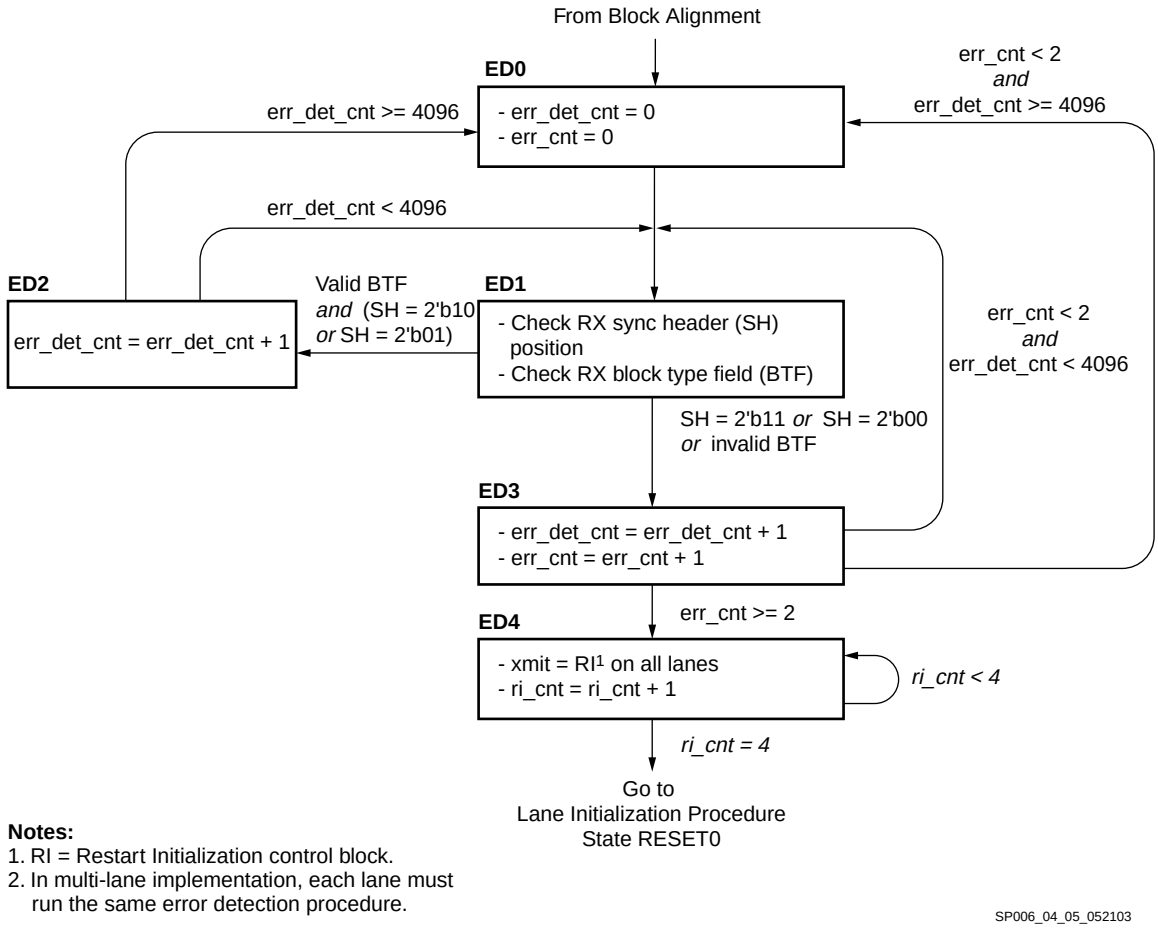


Figure 4-9: **Error Detection Procedure**

4.3.4.1 Error Detection Procedure

1. Block alignment in Lane Initialization is complete.
2. Set counter **err_cnt** and timer **err_det_cnt** to 0
3. Check sync header for correctness.
   a. If correct, check next sync header for correctness.
   b. If incorrect, increment **err_cnt** by 1 and check next sync header for correctness.
4. When **err_det_cnt** is equal to 4,096, and **err_cnt** is less than 2, reset **err_det_cnt** and **err_cnt** to 0.
5. If **err_cnt** is equal to 2, send four Restart Initialization control blocks to channel partners and restart lane initialization procedure.

4.3.4.2 Receiver Error Handling

It is possible for errors to occur upon reception. If a receiver detects an error, it will be signaled at the user interface. The receiver will only indicate the detection of errors. The receiver will not drop user data upon the detection of an error. That task is left to higher layers. Errors will only be indicated at the user interface if the error occurs during the reception of user data. Errors seen during idles will not be indicated at the user interface. Errors will be signaled on the following conditions:

- Sync header value equal to 2'b11 or 2'b00

  The receiver will declare all 8 bytes of the block code invalid. The error will only be indicated at the user interface if the error occurs during the reception of a PDU. Errors seen during idles will not be indicated at the user interface.

- The block type field of a control block does not correspond to one of the 15 possible values

  The receiver will declare all 8 bytes of the block code invalid.

- Idle control block with any non-idle byte from byte 1 through byte 7

  The receiver will declare no error at the user interface.

- Protocol Violation

  The receiver will declare error at the user interface when it detects any protocol violation. For example, if data blocks are received after an End Channel PDU control block and without a Start Channel PDU control block, they will be declared as errors.

5 PCS and PMA Layers

5.1 Overview

This chapter specifies the functions provided by the *physical coding sublayer* (PCS) and *physical medium attachment* (PMA) sublayer (the PCS and PMA terminology is adopted from IEEE Std 802.3®). While the Aurora Protocol does not explicitly implement these sublayers, the physical layer functions that are part of the Aurora Protocol are described using this conceptual model.

The topics include:

- 8B/10B and 64B/66B coding
- Character representation
- Data stream serialization
- Code groups
- Columns
- Channel transmission rules
- Idle sequences
- Channel initialization

This section also describes how data is striped across lanes when the channel is made up of more than one lane. Referring back to Section 1.3 Overview, page 14, an *Aurora channel* is made up of one or more *Aurora lanes*, each of which is a full-duplex serial connection.

5.2 PCS Layer Functions

The physical coding sublayer (PCS) function is responsible for *idle sequence* generation, lane striping, and encoding for transmission. Upon reception, the PCS is responsible for decoding, lane alignment, and destriping. The PCS in a standard implementation uses an 8B/10B encoding for transmission over the channel. See Section References, page 107, Reference 1, for the source of the 8B/10B encoding scheme. The PCS in a 10 Gbps physical layer implementation uses a 64B/66B encoding for transmission over the channel.

The PCS layer also provides mechanisms to detect lane states. It provides for clock difference tolerance between the sender and receiver without requiring *flow control*. The Aurora Protocol does not include mechanisms for determining the number of lanes that make up the channel. Each *channel partner* must be configured for operation over the same number of lanes.

The PCS layer performs the following transmit functions:

For 8B/10B

- Dequeues channel PDUs, *native* flow control PDUs, *user* flow control PDUs and delimited control symbols awaiting transmission as a character stream
- Stripes the transmit character stream across the available lanes
- Generates the idle sequence and inserts it into the transmit character stream for each lane when no PDUs or delimited control symbols are available for transmission
- Encodes the character stream of each lane independently into 10-bit parallel code groups
- Passes the resulting 10-bit parallel code groups to the PMA

The PCS layer performs the following receive functions:

- Decodes the received stream of 10-bit parallel code groups for each lane independently into characters
- Marks characters decoded from invalid code groups as invalid
- Aligns the character streams to eliminate the skew between the lanes and reassembles (destripes) the character stream from each lane into a single character stream, if the channel is using more than one lane
- Delivers the decoded character stream of PDUs and delimited control symbols to the higher layers

For 64B/66B

- Dequeues User Protocol Data Units (User PDUs) and delimited control symbols awaiting transmission as a character stream
- Stripes the transmit character stream across the available lanes
- Generates the Idle sequence and inserts it into the transmit character stream for each lane when no packets or delimited control symbols are available for transmission
- Encodes eight octets in the character stream of each lane independently into 66-bit parallel block codes
- Passes the resulting 66-bit parallel block codes to the PMA

The PCS layer performs the following receive functions:

- Decodes the received stream of 66-bit parallel block codes for each lane independently into characters
- Marks characters decoded from invalid block codes as invalid
- Aligns the character streams to eliminate the skew between the lanes and reassembles (destripes) the character stream from each lane into a single character stream, if the channel is using more than one lane
- Delivers the decoded character stream of packets and delimited control symbols to the higher layers

5.3 PMA Layer Functions

For 8B/10B

The physical medium attachment (PMA) function is responsible for serializing 10-bit parallel code groups to/from a serial bitstream on a lane-by-lane basis. Upon receiving data, the PMA function aligns the received bitstream into 10-bit code group boundaries, independently on a lane-by-lane basis. It then provides a continuous stream of 10-bit code groups to the PCS, one stream for each lane. The 10-bit code groups are not observable by layers higher than the PCS.

For 64B/66B

The Physical Medium Attachment (PMA) function is responsible for serializing 66-bit parallel block codes to/from a serial bitstream on a lane-by-lane basis. Upon receiving data, the PMA function aligns the received bitstream into 66-bit block code boundaries, independently on a lane-by-lane basis. It then provides a continuous stream of 66-bit block codes to the PCS, one stream for each lane. The 66-bit block codes are not observable by layers higher than the PCS.

5.4 8B/10B Transmission Code

The 8B/10B transmission code used by the PCS encodes 9-bit characters (eight bits of information and one control bit) into 10-bit code groups for transmission, and reverses the process on reception. Encodings are defined for 256 data characters and 12 special (control) characters.

The code groups used by the transmission code are either balanced or unbalanced. A balanced code group has an equal number of ones and zeros and an unbalanced code group has an unequal number of zeros. This selection of code groups guarantees a minimum of three transitions, 0 to 1 or 1 to 0, within each code group, while maintaining DC balance.

The 8B/10B code uses *disparity* to keep track of the number of ones and zeros it has sent and received. A code group that has two more ones than zeros is assigned a positive disparity. A code group that has two more zeros than ones is assigned a negative disparity. A code group that has an equal number of ones and zeros is assigned a zero disparity. The encoder maintains DC balance by never transmitting a second code group of the same disparity without first transmitting a code group of the opposite disparity. This rule does not apply to code groups with zero disparity. The complete definition is in Section 5.4.3 Running Disparity Rules, page 49.

The 8B/10B code has the following properties:

- Sufficient bit transition density (three to eight transitions per code group) to allow clock recovery by the receiver
- Special code groups that are used for establishing the receiver synchronization to the 10-bit code group boundaries, delimiting control symbols, and maintaining receiver bit and code group boundary synchronization
- DC balanced
- Detection of single and some multiple-bit errors

5.4.1 Character and Code Group Notation

This section describes the notation for characters, code groups, and their bits used in 8B/10B encoding and decoding.

The information bits [0-7] of an unencoded character are denoted with the letters **A** through **H** where the letter **H** denotes the most significant information bit (bit 0) and the letter **A** denotes the least significant information bit (bit 7). This is shown in Figure 5-1.

Each data character has a representation of the form D*x.y* where *x* is the decimal value of the least significant five information bits **EDCBA**, and *y* is the decimal value of the most significant three information bits **HGF**, as shown in Figure 5-1. Each control character has a similar representation using the form K*x.y*.

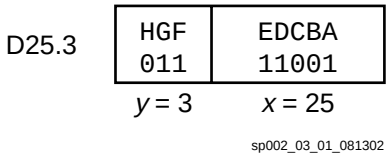


Figure 5-1: **Character Notation Example (D25.3)**

The output of the 8B/10B encoding process is a 10-bit code group. The bits of a code group are denoted with the letters **a** through **j**. The bits of a code group are all of equal significance, there is no most significant or least significant bit. The ordering of the code group bits is shown in Figure 5-2.

The code groups corresponding to the data character D*x.y* is denoted by /D*x.y*/. The code groups corresponding to the special character K*x.y* are denoted by /K*x.y*/.

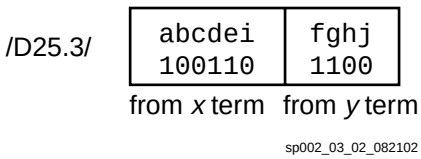


Figure 5-2: **Code Group Notation Example (/D25.3/)**

5.4.2 Running Disparity

The 8B/10B encoding and decoding functions use a binary variable called *running disparity*. The variable can have a value of either positive (RD+) or negative (RD-). The encoder and decoder each have a running disparity variable for each lane, which are all independent of each other.

The primary use of running disparity in the encoding process is to keep track of whether the encoder has output either more ones or more zeros. The current running disparity value is used to select which unbalanced code group will be used when the encoding for a character requires a choice between two unbalanced code groups.

The primary use of running disparity in the decoding process is to detect errors. Bit error(s) will transform a 10-bit code group into either an invalid code group or a different valid code group. For those code groups that were transformed into another valid code group, running disparity will catch all instances caused by a single bit error and some instances caused by multiple bit errors. When a running disparity error is flagged, the error can be either in that code group or in an earlier code group.

5.4.3 Running Disparity Rules

After power-up and before the channel is operational, both the transmitter (encoder) and receiver (decoder) must establish current values of running disparity. The transmitter shall use a negative value as the initial value for the running disparity for each lane.

The receiver may use either a negative or positive initial value of running disparity for each lane.

The following algorithm shall be used for calculating the running disparity for each lane. In the encoder, the algorithm operates on the code group that has just been generated by the encoder. In the receiver, the algorithm operates on the received code group that has just been decoded by the decoder.

Each code group is divided to two sub-blocks as shown in Figure 5-2, page 48, where the first six bits **abcdei** form one 6-bit sub-block and the second four bits **fghj** form a second 4-bit sub-block. The running disparity value at the beginning of the 6-bit sub-block is the value determined at the end of the previous code group. Running disparity at the beginning of the 4-bit sub-block is the running disparity at the end of the 6-bit sub-block. The running disparity value at the end of the code group is the value determined at the end of the 4-bit sub-block.

The sub-block running disparity shall be calculated as follows:

1. The running disparity is positive if at the end of any sub-block the sub-block contains more ones than zeros. It is also positive if at the end of a 4-bit sub-block the sub-block has the value 4'b0011, and if at the end of a 6-bit sub-block the sub-block has the value 6'b000111.
2. The running disparity is negative if at the end of any sub-block, the sub-block contains more zeros than ones. It is also negative if at the end of a 4-bit sub-block, the sub-block has the value 4'b1100, and if at the end of a 6-bit sub-block, the sub-block has the value 6'b111000.
3. In all other cases, the value of the running disparity at the end of the sub-block is the same as the value at the beginning of the sub-block (the running disparity is unchanged).

5.4.4 8B/10B Encoding

The 8B/10B encoding function encodes 9-bit characters into 10-bit code groups. The encodings for the 256 data characters (D*x.y*) are specified in Table A-1, page 81. The encodings or the 12 control characters (K*x.y*) are specified in Table A-2, page 90. Both tables have two columns of encodings, one marked RD- and one marked RD+. When encoding a character, the code group in the RD- column is selected if the current value of encoder running disparity is negative. The code group in the RD+ column is selected if the current value of encoder running disparity is positive.

Data characters (D*x.y*) shall be encoded according to Table A-1, page 81 and the current value of encoder running disparity. Special characters (K*x.y*) shall be encoded according to Table A-1, page 81 and the current value of encoder running disparity. After each character is encoded, the resulting code group shall be used by the encoder to update the running disparity according to Section 5.4.3 Running Disparity Rules, page 49.

5.4.5 Transmission Order

The 10-bit parallel output of the encoder shall be serialized and transmitted with bit **a** transmitted first and a bit ordering of **abcdei fghj** as shown in Figure 5-3.

Figure 5-3 gives an overview of a character passing through the encoding, serializing, transmission, deserializing, and decoding processes. The left side of the figure shows the transmit process of encoding a character stream using 8B/10B encoding and the 10-bit serialization. The right side shows the reverse process of the receiver deserializing and using 8B/10B decoding on the received code groups.

The dashed line shows the functional separation between the PCS layer, that provides 10-bit code groups, and the PMA layer.

The drawing also shows, on the receive side, the bits of a special character containing the comma pattern that is used by the receiver to establish 10-bit code-boundary synchronization.

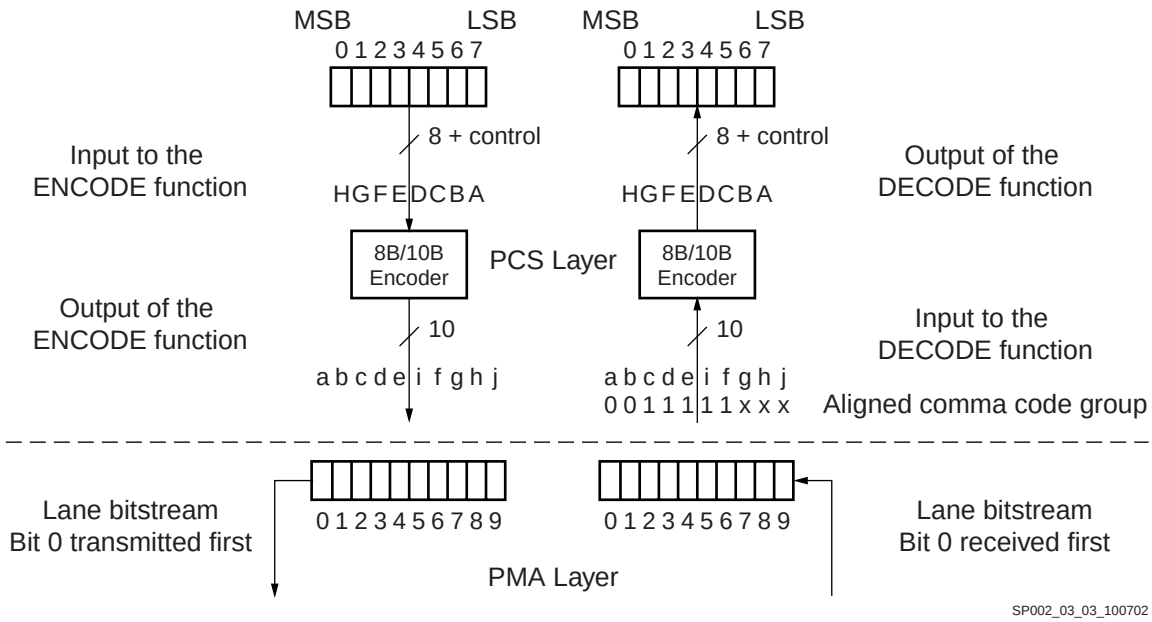


Figure 5-3: **Lane Encoding, Serialization, Deserialization, and Decoding Process**

5.4.6 8B/10B Decoding

The 8B/10B decoding function decodes received 10-bit code groups into 9-bit characters. It also detects received code groups that have no defined decoding and marks the resulting characters in the output stream invalid.

The decoding function uses Table A-1, page 81, Table A-2, page 90, and the current value of the decoder running disparity. To decode a received code group, the decoder shall select the RD- column of Table A-1, page 81 and Table A-2, page 90 if the current value of the decoder running disparity is negative or the RD+ column if the value is positive. The decoder shall then compare the received code group with the code groups in the selected column of both tables. If a match is found, the code group is decoded to the associated character. If no match is found, the code group is decoded to a character that is flagged as invalid. After each code group is decoded, it shall be used by the decoder to update the decoder running disparity according to the rules in Section 5.4.3 Running Disparity Rules.

5.4.7 Ordered Sets

Table 5-1 defines the ordered sets of special characters used by Aurora channel partners. These ordered sets are used for the following functions:

- Alignment to code group (10-bit) boundaries on lane-by-lane basis
- Alignment of the receive data stream across a channel
- Synchronization between channel partners
- Polarity checking
- Clock rate compensation between receiver and transmitter
- PDU delineation

Table 5-1: **Aurora Ordered Sets**

Ordered Set	Designator	Encoding
Idle	/I/	/K/, /R/, /A/ sequence
Sync and Polarity	/SP/	/K28.5/D10.2/D10.2/D10.2/
Sync and Polarity Acknowledge	/SPA/	/K28.5/D12.1/D12.1/D12.1/
Verification	/V/	/K28.5/D8.7 /D8.7/D8.7/
Start of Channel PDU	/SCP/	/K28.2/K27.7/
End of Channel PDU	/ECP/	/K29.7/K30.7/
Pad or Start of User Flow Control PDU	/P/ or /SUF/	/K28.4/
Comma	/K/	/K28.5/
Skip	/R/	/K28.0/
Channel Bonding	/A/	/K28.3/
Clock Compensation	/CC/	/K23.7/K23.7/
Start of Native Flow Control PDU	/SNF/	/K28.6/

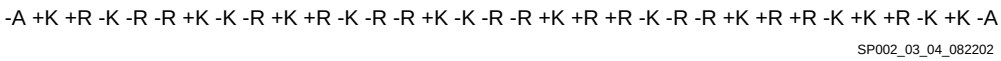
5.4.8 Idle Sequence

The *idle ordered set* (idle), shown in Table 5-1, page 51, is used to perform word-boundary alignment and channel bonding during initialization. During operation, idles are used to indicate that there is no data. Idle insertion occurs during wait-states and in between channel PDUs. The idle ordered set consists of three code groups: /A/, /K/, and /R/. The /K/ and /R/ code groups must be applied in a pseudo-random sequence (between /A/ code groups) to reduce EMI, by not producing a discrete spectrum. The following conditions apply:

- /A/ spacing is randomized with a minimum of 16 code groups but no more than 32 code groups between any two /A/ code groups.
- /K/s and /R/s are placed between the /A/s in a random fashion.
- The minimum transmit pattern is one *symbol-pair*. Any of the three characters can be sent, as long as the preceding two rules are obeyed. If the /A/ is sent, the spacing rule must still be obeyed.
- For multi-lane channels, the same idle symbol-pair must be transmitted simultaneously on all lanes that require an idle at that time.
- No discrete spectrum is created.

The example in Figure 5-4 shows an /A/ code group having a separation of 31 code groups. It also shows the current running disparity after the character has been applied.

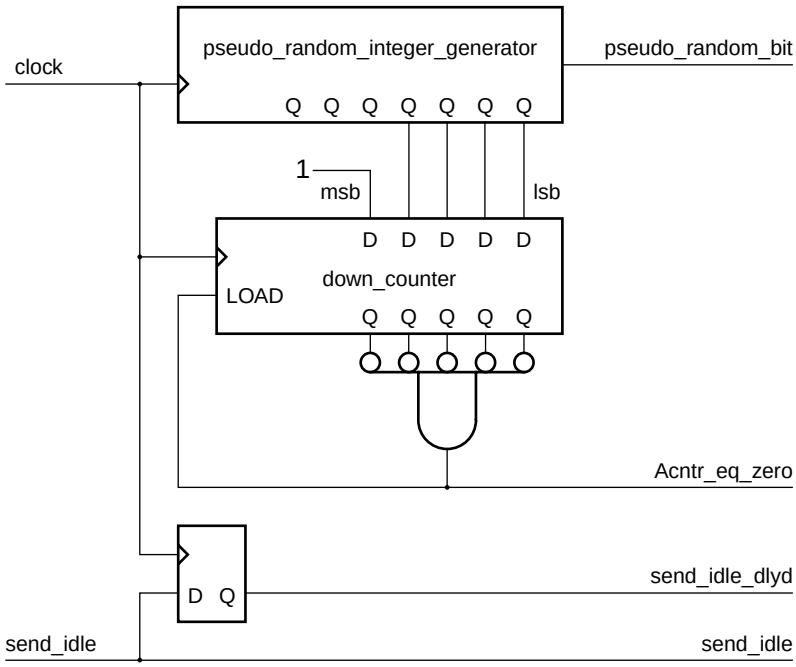
Note: /A/ and /K/ will change the current running disparity. /R/ will keep the disparity the same.



SP002_03_04_082202

Figure 5-4: **Example of an Idle Sequence**

Figure 5-5 shows an example implementation of idle generation logic which meets the requirements.




```
send_K = send_idle & (!send_idle_dlyd | send_idle_dlyd & !Acntr_eq_zero & pseudo_random_bit)
send_A = send_idle & send_idle_dlyd & Acntr_eq_zero
send_R = send_idle & send_idle_dlyd & !Acntr_eq_zero & !pseudo_random_bit
```

SP002_03_04_082102

Figure 5-5: **Example of a Pseudo-Random Idle Code Group Generator**

5.4.9 Clock Compensation

The Aurora Protocol provides a compensating mechanism for clock rate differences between the transmitter and receiver. This mechanism, called *clock compensation*, can accommodate up to a 200 ppm clock rate differential between the transmitter and the receiver. The Aurora Protocol implements clock compensation by periodically inserting clock compensation sequences into idle patterns or user data. Clock compensation sequences should not be inserted into user flow control PDUs, see Section 3.1.4 User Flow Control Operation, page 28.

The clock compensation sequence consists of six copies of the clock compensation ordered set, /CC/. The clock compensation sequence shall be transmitted at least every 10,000 code groups even when there are PDUs or other code groups available for transmission. For multi-lane channels, the complete clock compensation sequence is transmitted simultaneously over each lane that makes up the channel.

5.4.10 Single-Lane Transmission Rules

A single-lane channel has a single differential pair in each direction. A single-lane channel shall be encoded and shall transmit the character stream of control ordered sets and PDUs received from the upper layers over the differential pair in the order the characters were received from the upper layers.

When neither control ordered sets nor PDUs are available from the upper layers for transmission, the idle sequence shall be fed to the input of the encoder for encoding and transmission.

On reception, the code group stream is decoded and passed to the upper layers. Figure 5-6, page 55 shows the encoding and transmission order for a channel PDU transmitted over a single-lane channel. The data stream shown in Figure 5-6, page 55 illustrates many of the cases defined in the protocol. The key features to note in the figure are:

- The first example shows how a channel PDU with an odd number of payload octets is padded to maintain alignment of the /ECP/ symbol
- The third example shows a channel PDU interrupted by a clock compensation sequence, and idles

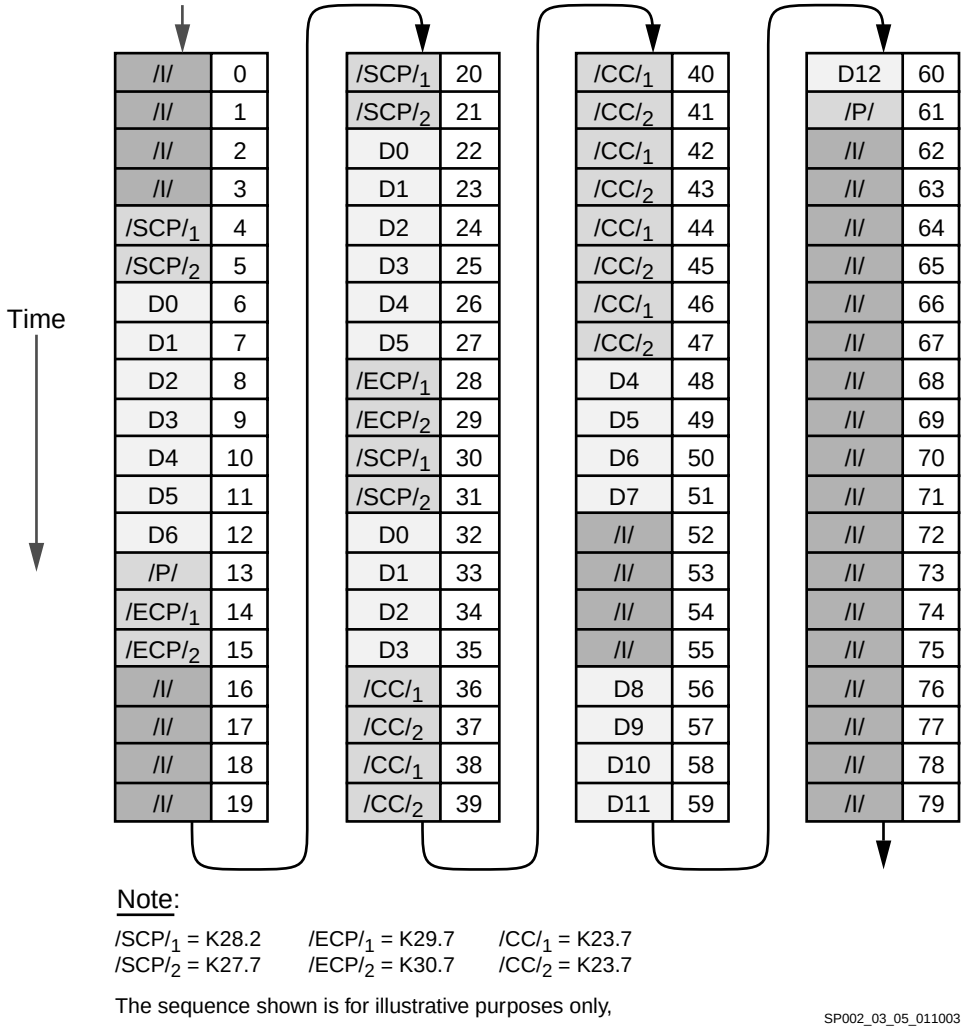


Figure 5-6: **Single-Lane Channel Typical Data Flow**

5.4.11 Multi-Lane Striping and Transmission Rules

The Aurora Protocol defines the striping of user data and control ordered sets across channels consisting of an arbitrary number of lanes. The striping scheme balances lane efficiency and implementation simplicity. Figure 5-7 is an overview of the striping scheme.

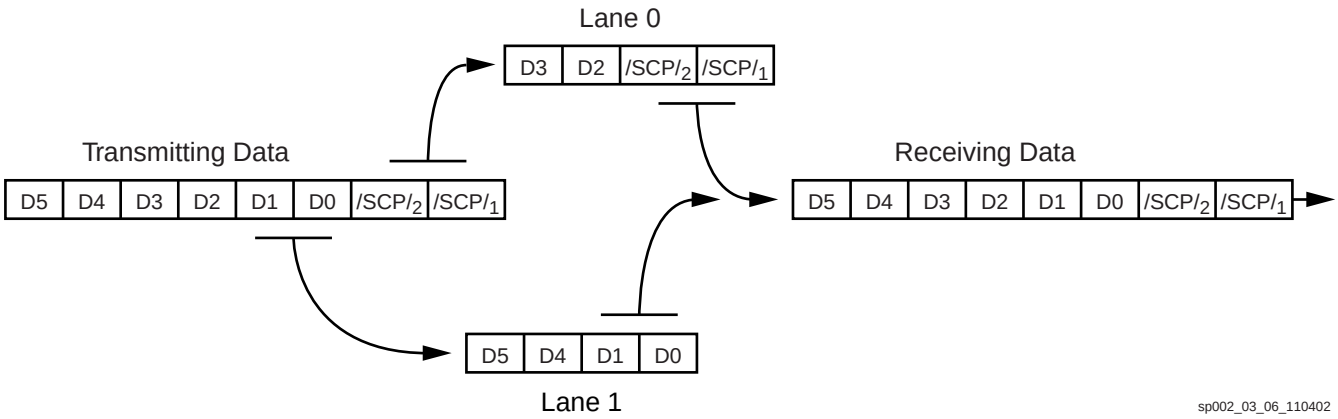


Figure 5-7: **Channel Striping Overview**

Striping allocates symbol-pairs across multiple lanes. Striping is the method used to send data simultaneously across all *n* lanes of a multi-lane channel. The symbol stream is striped across the lanes on a symbol-pair by symbol-pair basis. Striping may begin in any lane and proceeds lane by lane. For example, the first symbol-pair is striped onto lane 0, the second symbol-pair onto lane 1, and the *n*th symbol-pair onto lane *n*-1. The *n*th+1 symbol-pair is striped onto lane 0.

The only special requirements are as follows:

- The individual symbols in the symbol pairs /SCP/, /ECP/, /SNF/, /SUF/ must not be split between lanes, but can be transmitted and received on any lane
- When /I/ sequences are transmitted, the same data pattern must be transmitted over each lane in the channel that requires an /I/ sequence

Figure 5-8, page 57 shows how the same channel sequence shown in Figure 5-6, page 55 would be transmitted over a channel consisting of three lanes.

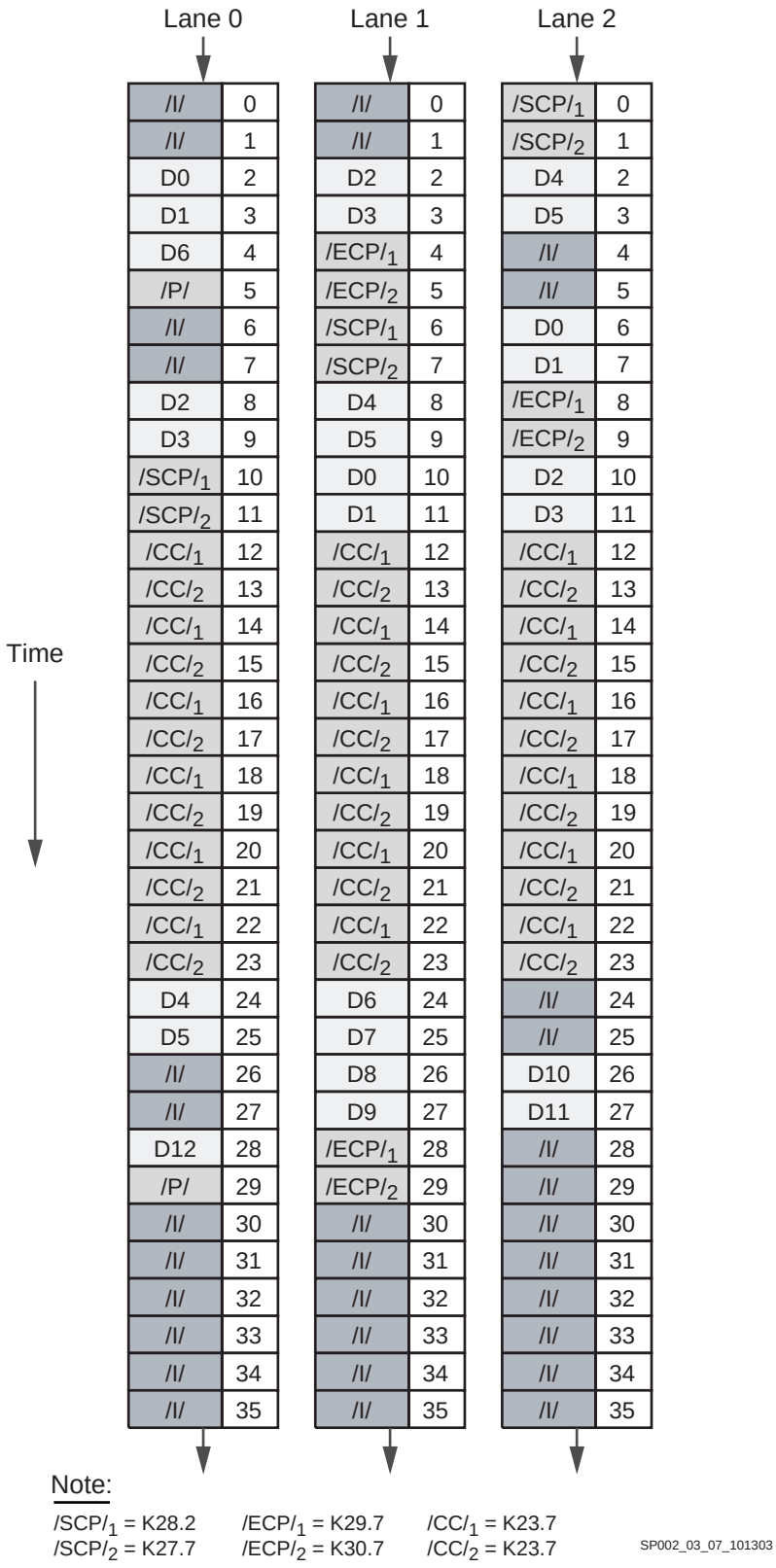


Figure 5-8: **Typical Data Flow for a Triple-Lane Channel**

5.5 64B/66B Transmission Code

Most transmission media use some type of bit encoding. 64B/66B is a block code that is used by 10 Gigabit Ethernet and 10 Gigabit Fibre Channel to decode and encode data. The 64B/66B code supports data and control characters while maintaining robust error detection. 64B/66B takes 64 bits of data (8 bytes) along with a two-bit value called a *sync header* (SH). The 64-bits of data can either be user data, control symbols, or a mix of user data and control symbols. The sync header can have a value of either 2'b10 or 2'b01. Values of 2'b00 and 2'b11 are illegal and will be signaled as errors. The combination of the 64 data bits and two sync header (SH) bits is called a block code. The 64 bits of transmit data are scrambled using a self-synchronizing scrambler with a polynomial of $1 + X^{39} + X^{58}$. The two-bit sync header is never scrambled. All transmitted data will be in the form of a block code. There are no exceptions. If the sync header is 2'b01, all 8 bytes are user data. This is referred to as a *data block*. If the sync header is 2'b10, a control function is designated. This is referred to as a *control block*. For a control block, the type of control function is denoted by byte 0. Control blocks include Idles, Start Channel PDU, and End Channel PDU designators. A control block carries either control information or a mix of control information and data, or a mix of control information and idles or pauses. A data block only carries data. Aurora 64B/66B has 19 possible block codes. Of these, 17 are control blocks and two are data blocks. Aurora 64B/66B defines 11 control blocks and two data blocks. The other six control blocks are reserved for future use.

Table 5-2 shows the big-endian bit ordering for a single byte of the block code when transmitting and receiving data. Table 5-3 shows the byte ordering and position of the SH of a block code. This specification uses little-endian notation for character definition. It is the responsibility of the implementer to ensure the proper bit positions for transmitting and receiving block codes.

Table 5-2: **Bit Ordering Within a Byte of Block Code**

bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7

Table 5-3: **Byte Ordering and SH Position Within a Block Code**

SH	byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7

A *gearbox* may be necessary when using 64B/66B. If the transmitter operates on a multiple of eight-bits, the gearbox performs the rate conversion between sending 66 bits instead of 64 bits. Since 66 bits are being sent, the transmitter will actually be operating at 10.3125 Gbps to compensate for the extra two bits, giving a data throughput of 10 Gbps.

5.5.1 Transmission and Reception Data Order

Block codes must be transmitted and received in a particular order. Figure 5-9 shows ordering for the transmitter given a 32-bit interface. Data flow is from top to bottom. Bit 0 of byte 0 is the first bit into the scrambler. The *sync header* (SH) is not scrambled, but is merged into the data stream after the scrambler. After the merging, bit 0 of SH is the first bit into the gearbox, if a gearbox is used. Bit 0 of SH is transmitted first, followed by bit 1 of SH, and the scrambled data.

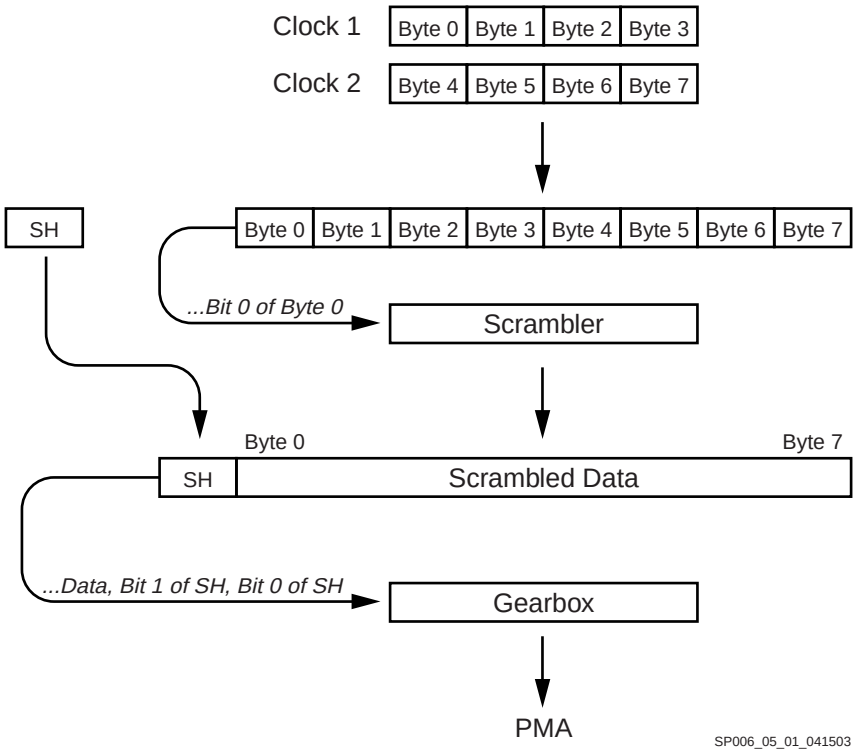


Figure 5-9: **Data Transmission Ordering**

Figure 5-10 shows data ordering for the receiver, given a 32-bit interface. This is the reverse process of the transmitter ordering. Since bit 0 of SH was the first bit sent, it must be the first bit received. SH is stripped from the data stream so that only the scrambled data is left. This allows bit 0 of byte 0 to be the first bit into the descrambler, ensuring the proper decoding of the bits. After descrambling, the data is presented at the interface.

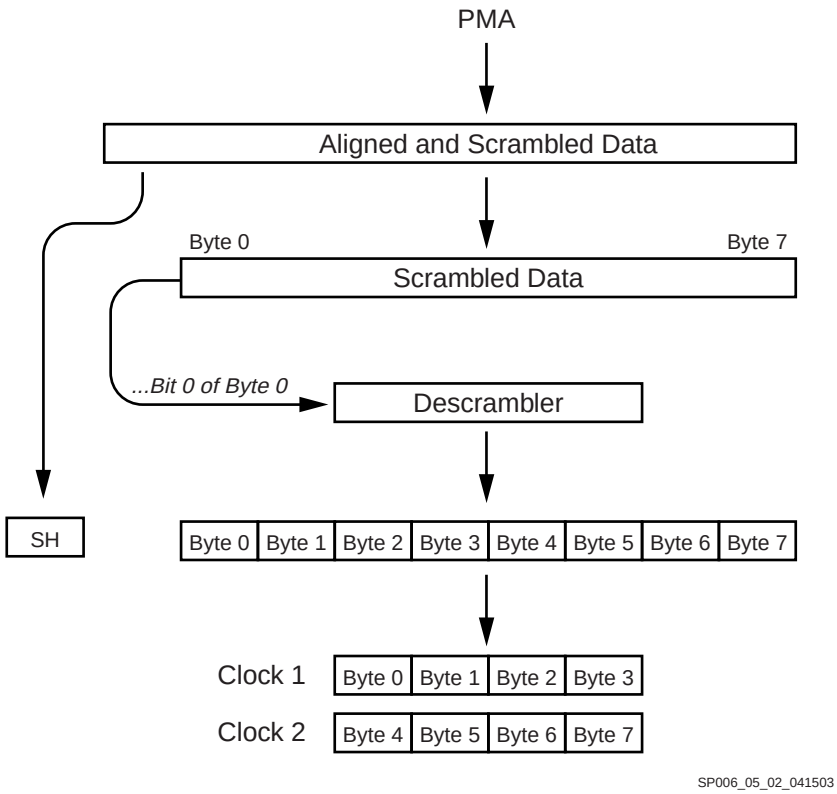


Figure 5-10: **Data Reception Ordering**

5.5.2 Block Codes

Block codes regulate the operation of an Aurora 64B/66B protocol engine. Control blocks carry either control information or a mix of control information and data. Data blocks carry only data. Without exception, all data is transmitted in the form of a block code.

Aurora 64B/66B defines 11 control blocks and two data blocks, as shown in Table 5-4, page 61. Of the 15 possible control block type field codes available in 64B/66B encoding, Aurora 64B/66B uses nine. One of the type field codes serves multiple purposes. The Idle, Clock Compensation, and Channel Bonding 1 block codes use type field code 8'h1e.

Table 5-4: **Aurora 64B/66B Block Codes**

Function	Desig-nator	Sync Header (Binary)	Block Payload (Hexadecimal)							
			Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Idle	IDL	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
Clock Compensation	CC	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
Channel Bonding 1	CB1	2'b10	8'h1e	8'hf0	8'hf0	8'hf0	8'hf0	8'hf0	8'hf0	8'hf0
Channel Bonding 2	CB2	2'b01	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
Acknowledgement	ACK	2'b10	8'h55	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
Restart Initialization	RI	2'b10	8'h4b	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
Start Channel PDU	SCP	2'b10	8'h78	N	D/8'h00	D/8'h00	D/8'h00	D/8'h00	D/8'h00	D/8'h00
End Channel PDU	ECP	2'b10	8'h87	P	D/8'h00	D/8'h00	D/8'h00	D/8'h00	D/8'h00	D/8'h00
Partial Data	PD	2'b10	8'h99	Q	D	D	D/8'h00	D/8'h00	D/8'h00	D/8'h00
Native FC	SNF	2'b10	8'haa	R	D/8'h00	D/8'h00	D/8'h00	D/8'h00	D/8'h00	D/8'h00
User FC	SUF	2'b10	8'hb4	S	M	M	M/8'h00	M/8'h00	M/8'h00	M/8'h00
Channel Verification	CV	2'b10	8'h2d	8'he8	8'he8	8'he8	8'he8	8'he8	8'he8	8'he8
Reserved	n/a	2'b10	8'h33	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Reserved	n/a	2'b10	8'h66	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Reserved	n/a	2'b10	8'hcc	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Reserved	n/a	2'b10	8'hd2	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Reserved	n/a	2'b10	8'he1	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Reserved	n/a	2'b10	8'hff	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Data	DAT	2'b01	D	D	D	D	D	D	D	D

Notes: Block Type Field Codes

1. Byte 0 contains the block type field codes for the control blocks

Notes: Byte-Wide Control Fields

1. D denotes a byte of user data
2. M denotes a byte in a user flow control message

Notes: Partial-Byte Control Fields

1. N denotes the first 3 bits as SPB, which indicates the last data byte. The last 5 bits are reserved.
2. P denotes the first 3 bits as EPB, which indicates the last data byte. The last 5 bits are reserved.
3. Q denotes the first 3 bits as PB, which indicates the last data byte before idles begin. The last 5 bits are reserved.
4. R denotes the first 3 bits as NFB, which indicates the last valid byte of data. R also denotes the last 4 bits as IDLES, which indicates how many idles to send. The remaining bit is reserved.
5. S denotes the first 4 bits as SIZE, which indicates the number of bytes in the flow control message. The maximum message length is 16 bytes. The last 4 bits are reserved.

5.5.3 Idle Control Block

Block type field 8'h1e denotes the Idle control block if bytes 1 through 7 are all 8'h00. The Idle control block will be transmitted during link initialization when there is a need for clock compensation (clock correction), or when there is no user data to be sent, or when a PDU is paused. No data occurs between PDUs and during the period when a PDU is paused.

5.5.4 Clock Compensation Control Block

Block type field 8'h1e denotes the clock compensation control block if bytes 1 through 7 are all 8'hff. Aurora 64B/66B provides a compensating mechanism for clock rate differences between the transmitter and receiver. This mechanism, called Clock Compensation, can accommodate up to a 200 ppm clock rate differential between the transmitter and the receiver. Aurora 64B/66B implements clock compensation by periodically inserting Clock Compensation sequences into user data. The Clock Compensation sequence consists of three instances of the Clock Compensation control block that must be transmitted every 10,000 clock cycles. The actual implementation period is left to the user, but there can be no more than 10,000 clock cycles between the transmissions of idle control blocks used for clock compensation. For clock compensation, a user is allowed to either delete or add all eight bytes of the clock compensation control block. For implementations that use more than one Aurora lane, three instances of the clock compensation shall simultaneously be sent down each individual lane that composes the Aurora 64B/66B channel.

5.5.5 Channel Bonding

The Channel Bonding 1 control block and the Channel Bonding 2 data block are used for Aurora 64B/66B implementations that have multiple lanes. The receiver will use these Channel Bonding blocks to eliminate any skew between channels. Skew can be caused by mismatched traces, connector pin differences and IC variations in the transmitter and receiver. Channel bonding (lane alignment) will allow greater than 12 ns of skew between channels.

5.5.5.1 Channel Bonding 1 Control Block

Block type field 8'h1e denotes the Channel Bonding 1 control block if bytes 1 through 7 are all 8'hf0. The Channel Bonding 1 block code is seen as a control block because its sync header is equal to 2'b10.

5.5.5.2 Channel Bonding 2 Data Block

The Channel Bonding 2 block code is a data block where the sync header is equal to 2'b01 and bytes 0 through 7 are all 8'h00.

The Channel Bonding 2 data block is distinguished as a special data block because of its use during channel bonding. It holds no other significance. During normal operation, if this data block is transmitted, it will not be treated differently from any other data block.

5.5.6 Acknowledgement Control Block

Block type field 8'h55 denotes the Acknowledgement control block. This control block is only used during initialization. Once an initialization step has been completed, the Acknowledgement control block will be transmitted to one's channel partner to signify that it has completed the present step.

5.5.7 Channel Verification Control Block

Block type field 8'h2d denotes the Channel Verification control block if bytes 1 through 7 are all 8'he8. The Channel Verification control block is transmitted during channel verification. Using this known data pattern, the receiving channel partners can verify channel integrity and correctly align data across the user interface.

5.5.8 Restart Initialization Control Block

Block type field 8'h4b denotes the Restart Initialization control block. If there is a need to reset, the Restart Initialization control block is set to one's channel partner to inform the need to reinitialize the link. Upon receipt of this control block, an Aurora 64B/66B channel must restart the initialization process.

5.5.9 Start Channel PDU Control Block

Block type field 8'h78 denotes the Start Channel PDU control block. This control block indicates the start of a PDU transmission. SPB, contained in the three most significant bits of byte 1, designates among bytes 2 through 7 as to which one contains the first valid byte prior to any unused bytes in this control block. The last five bits of byte 1 are reserved. For transmission, all bytes, if any following the last valid data byte shall be 8'h00 (idles). For reception, all bytes, if any following the last valid data byte shall be deemed idles and discarded. The Start Channel PDU control block can only carry odd numbers of bytes if the entire PDU can fit into this control block. Otherwise, it can only carry even numbers of bytes. Figure 5-11 shows the bit ordering for the Start Channel PDU control block.

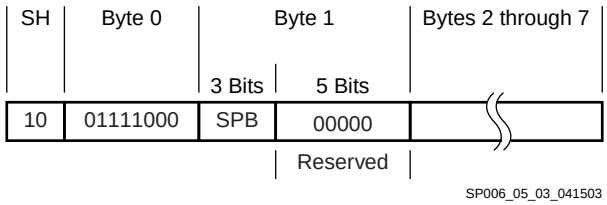


Figure 5-11: **Start Channel PDU Bit Ordering**

Table 5-5 shows the encoding of the SPB field.

Table 5-5: **SPB Encoding**

SPB Field Contents	Valid Byte End Point
000	Contains no data
001	Reserved
010	Byte 2
011	Byte 3
100	Byte 4
101	Byte 5
110	Byte 6
111	Byte 7

5.5.10 End Channel PDU Control Block

Block type field 8'h87 denotes the End Channel PDU control block. This control block indicates the end of a user PDU. EPB in byte 1 designates which byte in the control block is the actual end of data. EPB is contained in the three most significant bits of byte 1. The last five bits of byte 1 are reserved. Note that if EPB is equal to 3'b000, the PDU ended on the previous block. In the majority of cases, this will be a Start Channel PDU control block or a data block. The implementation must handle both data and control blocks. This allows the transmission of PDUs less than seven bytes in length. For transmission, all bytes, if any after the last valid data byte shall be 8'h00 (designating idles). For reception, all bytes, if any after the last valid data byte shall be deemed idles and discarded. Figure 5-12 shows the bit ordering for the End Channel PDU.

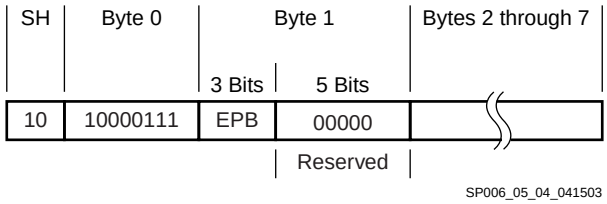


Figure 5-12: **End Channel PDU Bit Ordering**

Table 5-6 shows the encoding of the EPB field.

Table 5-6: **EPB Encoding**

EPB Field Contents	PDU End Point
000	Previous block
001	Reserved
010	Byte 2
011	Byte 3
100	Byte 4
101	Byte 5
110	Byte 6
111	Byte 7

5.5.11 Partial Data Control Block

Block type field 8'h99 denotes the Partial Data control block. This control block is used to transmit data with idles in the middle of a PDU transmission. The Partial Data control block is provided for transmission of data less than seven bytes between the Start of Channel PDUs and the End of Channel PDUs. In the majority of cases, this will happen when users wish to pause transmission, but cannot fit the rest of data into an 8-byte data block. The Partial Data control block is also necessary in a transmission of the last seven bytes of a PDU that cannot be fit into an End Channel PDU control block. The Partial Data control block can be used many times during a PDU transmission as long as it contains valid PDU data. PB in byte 1 designates which byte in the control block is the last valid byte of data before the idles begin. PB is contained in the first three bits of byte 1. The last five bits of byte 1 are reserved. For transmission, all bytes, if any after the last valid data byte

shall be 8'h00 (idles). For reception, all bytes, if any after the last valid data byte shall be deemed idles and discarded. The Partial Data control block can only carry even bytes of PDU data. Figure 5-13 shows the bit ordering for the Partial Data control block.

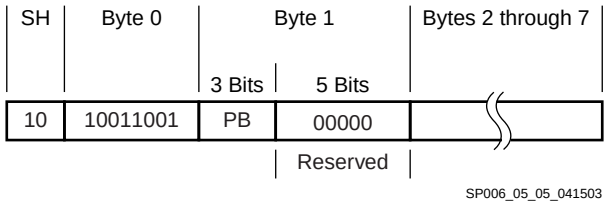


Figure 5-13: **Partial Data Bit Ordering**

Table 5-7 shows the encoding of the PB field.

Table 5-7: **PB Encoding**

PB Field Contents	Valid Byte End Point
000 - 001	Reserved
010	Reserved
011	Byte 3
100	Reserved
101	Byte 5
110	Reserved
111	Byte 7

5.5.12 Native FC Control Block

Block type field 8'haa denotes the Native FC control block. A Native FC control block can be embedded within a PDU or transmitted between PDUs. This control block is used for native flow control. This block code is transmitted by a user across the Aurora 64B/66B channel when its receive FIFO is in danger of overflow. Upon reception, the receiver will send the amount of idles specified within the IDLES field. Out of the 10 possible values of the IDLES field, two are special. If IDLES equals 4'b0000, an XON (zero idles) is called and transmission of new data can begin at any time. If IDLES equals 4'b1111, an XOFF is called and idles are sent until receipt of another native flow control message containing an XON or an IDLES field less than or equal to 256. Bytes 2 through 7 are used to carry data, if any, in a PDU. However, the Native FC control block can only carry even bytes of PDU data. NFB in byte 1 designates which byte in the control block is the last valid byte of data before idles begin. The NFB designator is contained in the first three bits of byte 1. Note that if EPB is equal to 3'b000, it indicates that no PDU data is carried in this Native FC control block. The IDLES designator is contained in the last four bits of byte 1. The remaining bit is reserved. For transmission, all bytes, if any after the last valid data byte shall be 8'h00 (idles). For reception, all bytes, if any after the last valid data byte shall be deemed idles and discarded.

Figure 5-14 shows the bit ordering for the Native FC control block.

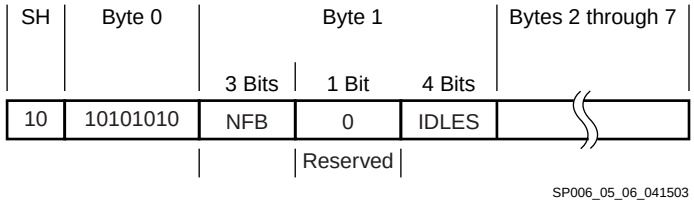


Figure 5-14: **Native FC Bit Ordering**

Table 5-8 shows the encoding of the NFB field.

Table 5-8: **NFB Encoding**

NFB Field Contents	Valid Byte End Point
000	No valid byte
001	Reserved
010	Reserved
011	Byte 3
100	Reserved
101	Byte 5
110	Reserved
111	Byte 7

Table 5-9 shows the encoding of the IDLES field.

Table 5-9: **IDLES Encoding**

IDLES Field Contents	Idles Sent
0000	0 (XON)
0001	2
0010	4
0011	8
0100	16
0101	32
0110	64
0111	128
1000	256
1001 to 1110	Reserved
1111	Infinite (XOFF)

5.5.13 User FC Control Block

Block type field 8'hb4 denotes the User FC control block. This control block is used for layers above Aurora 64B/66B to transmit flow control messages across the Aurora 64B/66B channel. Since layers above Aurora 64B/66B generate the message, the message is simply transmitted across the Aurora 64B/66B channel. The User FC message can be any even number of bytes from 2 to 16.

Byte 1 in the User FC control block contains a 4-bit field labeled "SIZE" that indicates the size of the message. If the message is greater than the six remaining bytes in the block, Data blocks and Partial Data control blocks containing the remainder of the message will be sent. For example, one User FC control block, one Data block, and one Partial Data control block are required to send a 16-byte message. All unused bytes in the User FC control block will contain 8'h00.

Figure 5-15 shows the bit ordering for the User FC control block.

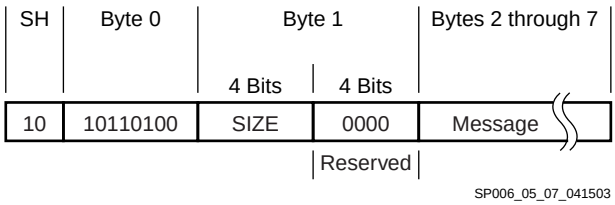


Figure 5-15: **User FC Bit Ordering**

Table 5-10 shows the encoding of the SIZE field.

Table 5-10: **SIZE Encoding**

SIZE Field Contents	Message Size
0000	Reserved
0001	2 bytes
0010	Reserved
0011	4 bytes
0100	Reserved
0101	6 bytes
0110	Reserved
0111	8 bytes
1000	Reserved
1001	10 bytes
1010	Reserved
1011	12 bytes
1100	Reserved
1101	14 bytes
1110	Reserved
1111	16 bytes
Other values	Reserved

5.5.14 Data Block

A data block is designated by the sync header value of 2'b01. Only data blocks are permitted to have this sync header value. Each data block contains eight bytes of data. It is not permissible for a data block to contain control characters. Note that the data block for Channel Bonding 2 is a subset of this group.

5.6 Data Transmission for 64B/66B

Data transmission of a PDU begins with a Start Channel PDU control block and ends with an End Channel PDU control block. The Start Channel PDU control block requires the first byte of data to be on byte 2. If PDUs of less than seven bytes are to be transmitted, data could end on any byte from 2 to 7 in a Start Channel PDU control block. An End Channel PDU control block allows data to end on any byte from 2 to 7. The End Channel PDU control block also has a special code that allows a PDU to end on the previous block. This allows transmission of PDUs of six bytes or less. PDU transmission can be paused with a Partial Data control block and/or Idle control block, and can be restarted with a Data block or Partial Data control block. There is no limit on how many times a PDU can be paused.

The smallest PDU must contain at least one byte and requires a Start Channel PDU control block and an End Channel PDU control block. The EPB field in the End Channel PDU control block uses the special value of 3'b000 indicating that the PDU finished on the *previous* block. This means that for the smallest transfer, the PDU starts and ends on byte 2 of the Start Channel PDU control block. An example of the smallest transfer is shown in Table 5-11.

Table 5-11: **Minimum PDU Transfer**

Sync Header	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
2'b10	8'h78	8'h40	D	8'h00	8'h00	8'h00	8'h00	8'h00
2'b10	8'h87	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00

Idles are transmitted when there is no data to send. This occurs between PDUs and when transmission of a PDU is temporarily paused. The transmitter does not have to separate user PDUs with Idle control blocks. It is perfectly acceptable to send back-to-back PDUs.

Native FC and User FC control blocks may be transmitted at any time. They can be embedded within PDUs or sent between PDUs. Native FC control blocks can also carry user data.

A transmitter must also send three clock compensation control blocks at least once every 10,000 clock cycles for clock compensation. If the Aurora 64B/66B channel uses multiple lanes, the three Clock Compensation control blocks must be sent across each lane at the same time.

5.6.1 Single-Lane Transmission Rules

Table 5-12, page 69 shows an example of data transmission across a single lane, beginning with the transmission of idles. A summary of functions noted in the table are as follows:

- The PDU control block starts transmission with a Start Channel PDU
- The PDU is paused and idles are inserted

- A restart of the PDU begins, interrupted by clock compensation control blocks before the last PDU is transmitted
- Idles are again transmitted until the next Start Channel PDU is issued
- Another PDU is started and a native flow control message is embedded within the PDU, which asks that 256 idles be sent before resuming data transmission
- The PDU is then completed, clock compensation occurs, and idles are transmitted
- A user flow control message is transmitted containing 16 bytes and requiring the transmission of three user flow control blocks
- The user flow control message is followed by idles and a Start Channel PDU
- Back-to-back PDUs are transmitted, the second of which starts with 2 bytes rather than 6 bytes in Start Channel PDU
- The second PDU is interrupted by clock compensation before it is completed, followed by End Channel PDU and idles

Note: This example is for illustrative purposes only and not intended to represent an actual transfer.

Table 5-12: **Single-Lane Channel Typical Data Flow**

Function	Sync Header	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
Start Channel PDU	2'b10	8'h78	8'he0	D	D	D	D	D	D
PDU Data	2'b01	D	D	D	D	D	D	D	D
Partial Data	2'b10	8'h99	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
PDU Data	2'b01	D	D	D	D	D	D	D	D
Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
PDU Data	2'b01	D	D	D	D	D	D	D	D
End Channel PDU (Total 38 bytes)	2'b10	8'h87	8'he0	D	D	D	D	D	D
Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
Start Channel PDU	2'b10	8'h78	8'he0	D	D	D	D	D	D
PDU Data	2'b01	D	D	D	D	D	D	D	D
Native Mode Flow Ctrl	2'b10	8'haa	8'he8	D	D	D	D	D	D
End Channel PDU (Total 22 bytes)	2'b10	8'h87	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff

Table 5-12: **Single-Lane Channel Typical Data Flow *(Continued)***

Function	Sync Header	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
User Flow Ctrl	2'b10	8'hb4	8'hf0	M	M	M	M	M	M
Data	2'b01	M	M	M	M	M	M	M	M
Partial Data	2'b10	8'h99	8'h60	M	M	8'h00	8'h00	8'h00	8'h00
Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
Start Channel PDU	2'b10	8'h78	8'he0	D	D	D	D	D	D
PDU Data	2'b01	D	D	D	D	D	D	D	D
End Channel PDU (Total 16 bytes)	2'b10	8'h87	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
Start Channel PDU	2'b10	8'h78	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
PDU Data	2'b01	D	D	D	D	D	D	D	D
Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
PDU Data	2'b01	D	D	D	D	D	D	D	D
End Channel PDU (Total 20 bytes)	2'b10	8'h87	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00

5.6.2 Multi-Lane Striping and Transmission Rules

Data striping is used to distribute data across multiple lanes in an Aurora 64B/66B channel. Data striping is the process of taking user data, segmenting it, and then sending it down multiple lanes. The receiver reassembles the data from the multiple lanes for presentation at the output.

Data striping is done in block code increments. Each block code is transmitted over a different Aurora 64B/66B lane. The first block code to be transmitted begins on Lane 0. The next block code is sent down Lane 1. This progression continues until a block code has been sent down all lanes. Each time all lanes have transmitted a block code, the sequence restarts on Lane 0, striping across each successive lane until all lanes have a block code to transmit.

PDUs and user flow control messages can start and end on any lane. Likewise, native flow control blocks can be sent down any lane. Clock compensations, however, have stricter requirements. They must be sent simultaneously sent down all lanes using the same pattern. Any deviation in this will cause a loss of channel alignment.

Table 5-13 is based on the same data stream shown in the single-lane example in Table 5-12, but shows how it is striped over an Aurora 64B/66B channel that has three lanes. In the single-lane example, 36 block codes were transmitted. Because clock compensation control block must be striped across all lanes, an additional 18 block codes are required to transmit all the data. Another difference from the single-lane example is that the Idle block code is sent ahead of the clock compensation that occurs in clocks 9, 10, and 11. This is necessary to ensure that clock compensation happens simultaneously across all lanes.

Table 5-13: **Multi-Lane Channel Typical Data Flow**

Lane	Clock	Function	Sync Header	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0	1	Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
1	1	Start Channel PDU	2'b10	8'h78	8'he0	D	D	D	D	D	D
2	1	PDU Data	2'b01	D	D	D	D	D	D	D	D
0	2	Partial Data	2'b10	8'h99	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
1	2	Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
2	2	PDU Data	2'b01	D	D	D	D	D	D	D	D
0	3	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
1	3	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
2	3	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
0	4	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
1	4	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
2	4	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
0	5	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
1	5	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
2	5	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
0	6	PDU Data	2'b01	D	D	D	D	D	D	D	D
1	6	End Channel PDU (Total 38 bytes)	2'b10	8'h87	8'he0	D	D	D	D	D	D
2	6	Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
0	7	Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
1	7	Start Channel PDU	2'b10	8'h78	8'he0	D	D	D	D	D	D
2	7	PDU Data	2'b01	D	D	D	D	D	D	D	D
0	8	Native Flow Ctrl	2'b10	8'haa	8'he8	D	D	D	D	D	D
1	8	End Channel PDU (Total 22 bytes)	2'b10	8'h87	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
2	8	Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00

Table 5-13: **Multi-Lane Channel Typical Data Flow *(Continued)***

Lane	Clock	Function	Sync Header	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0	9	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
1	9	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
2	9	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
0	10	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
1	10	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
2	10	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
0	11	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
1	11	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
2	11	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
0	12	User Flow Ctrl	2'b10	8'hb4	8'hf0	M	M	M	M	M	M
1	12	Data	2'b01	M	M	M	M	M	M	M	M
2	12	Partial Data	2'b10	8'h99	8'h60	M	M	8'h00	8'h00	8'h00	8'h00
0	13	Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00
1	13	Start Channel PDU	2'b10	8'h78	8'he0	D	D	D	D	D	D
2	13	PDU Data	2'b01	D	D	D	D	D	D	D	D
0	14	End Channel PDU (Total 16 bytes)	2'b10	8'h87	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
1	14	Start Channel PDU	2'b10	8'h78	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
2	14	PDU Data	2'b01	D	D	D	D	D	D	D	D
0	15	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
1	15	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
2	15	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
0	16	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
1	16	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
2	16	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
0	17	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
1	17	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
2	17	Clock Compensation	2'b10	8'h1e	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff	8'hff
0	18	PDU Data	2'b01	D	D	D	D	D	D	D	D
1	18	End Channel PDU (Total 20 bytes)	2'b10	8'h87	8'h60	D	D	8'h00	8'h00	8'h00	8'h00
2	18	Transmit Idles	2'b10	8'h1e	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00	8'h00

6 Electrical Specifications

6.1 Overview

The AC specifications cover both single-lane and multi-lane implementations. The specifications define two types of transmitters and one type of receiver with baud rates of 1.25, 2.5, and 3.125 Gbps. The Aurora Protocol does not preclude the use of other baud rates, but only defines timing for these by way of example.

This chapter specifies differential signaling in quantities that represent the voltage difference between the true and complement signals. This is known as the peak-to-peak voltage. The peak-to-peak voltage is twice that of the peak voltage of either the true or the complement signal. Specific definitions are given Table 6-2, page 77.

To ensure interoperability between drivers and receivers of different vendors and technologies, AC coupling at the receiver input is required.

6.2 Signal Definition

The Aurora Protocol uses differential signaling between ports. This section specifies signals using *peak-to-peak* differential voltages. Figure 6-1 shows how the signals are defined. The figure shows waveforms for either a transmitter (TD and $\overline{TD}$) or a receiver (RD and $\overline{RD}$). Each signal swings between A volts and B volts. Using these waveforms, the definitions are as follows:

1. The transmitter or receiver has a peak-to-peak range of A - B
2. The differential signal ranges from + | A−B | to - | A-B |
3. The differential peak-to-peak signal is 2 * (A−B)

Peak-to-peak is the difference between the most positive and the most negative readings of a particular signal. In this case (A−B) −(−(A−B)) = 2*(A−B).

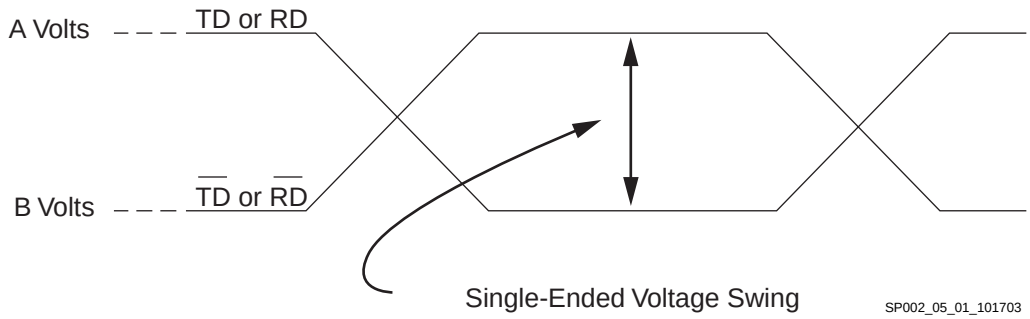


Figure 6-1: **Differential Peak-To-Peak Voltage of Transmitter or Receiver**

To illustrate this concept using real values, consider the case where a current mode logic (CML) transmitter has a termination voltage of 2.5V and has a swing between 2.5V and 2.0V. Using these values, the peak-to-peak range is 500 mV. The differential signal ranges between 500 mV and -500 mV. The peak differential signal is 500 mV. The differential peak-to-peak signal is 1000 mV.

6.3 Equalization

With the use of high-speed serial transceivers, the interconnect media causes degradation of the signal at the receiver. Effects such as inter-symbol interference (ISI) or data-dependent jitter are produced. This loss can be large enough to degrade the eye opening at the receiver beyond that which is allowed in this specification. To negate a portion of these effects, equalization techniques can be used, such as:

Pre-emphasis: Applied to the transmitter

Passive equalization: A passive high pass filter network placed at the receiver

Adaptive equalization: The use of active circuits in the receiver

6.4 Transmitter Specifications

Driver AC timing specifications are displayed in the tables below.

Table 6-1: **Transmitter AC Timing Specifications - 1.25 Gbps Baud Rate**

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential output voltage	V_{DIFF}	800	1600	mV	Peak-to-peak differential
Rise/fall time	TM_{RF}	60		ps	At driver output
Deterministic jitter	J_D		0.17	UI	
Total jitter	J_T		0.35	UI	
Output skew	S_O		25	ps	Skew at a transmitter output between the two signals comprising a differential pair
Multiple output skew	S_{MO}		1000	ps	Skew at the transmitter output between lanes of a multi-lane *channel*
Unit interval	UI	800	800	ps	+/- 100 ppm

Note: AC coupling is required to guarantee interoperability between vendors.

Table 6-2: **Transmitter AC Timing Specifications - 2.5 Gbps Baud Rate**

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential output voltage	V_{DIFF}	800	1600	mV	Peak-to-peak differential
Rise/fall time	TM_{RF}	40		ps	At driver output
Deterministic jitter	J_D		0.17	UI	
Total jitter	J_T		0.35	UI	
Output skew	S_O		20	ps	Skew at a transmitter output between the two signals comprising a differential pair
Multiple output skew	S_{MO}		1000	ps	Skew at the transmitter output between lanes of a multi-lane channel
Unit interval	UI	400	400	ps	+/- 100 ppm

Note: AC coupling is required to guarantee interoperability between vendors.

Table 6-3: **Transmitter AC Timing Specifications - 3.125 Gbps Baud Rate**

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential output voltage	V_{DIFF}	800	1600	mV	Peak-to-peak differential
Rise/fall time	TM_{RF}	30		ps	At driver output
Deterministic jitter	J_D		0.17	UI	
Total jitter	J_T		0.35	UI	
Output skew	S_O		15	ps	Skew at a transmitter output between the two signals comprising a differential pair
Multiple output skew	S_{MO}		1000	ps	Skew at the transmitter output between lanes of a multi-lane channel
Unit interval	UI	320	320	ps	+/- 100 ppm

Note: AC coupling is required to guarantee interoperability between vendors.

6.5 Receiver Specifications

Receiver AC timing specifications are displayed in the tables below.

Table 6-4: **Receiver AC Timing Specifications - 1.25 Gbps Baud Rate**

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential input voltage	V_{IN}	200	1600	mV	Peak-to-peak differential input voltage
Deterministic jitter	J_D		0.37	UI	Measured at receiver.
Total jitter	J_T		0.65	UI	Measured at receiver
Input skew	S_I		75	ps	Skew at a receiver input between the two signals comprising a differential pair
Multiple input skew	S_{MI}		24	ns	Skew at the receiver input between lanes of a multi-lane channel
Bit error rate	BER		10^{-12}		
Unit interval	UI	800	800	ps	+/- 100 ppm

Note: AC coupling is required to guarantee interoperability between vendors.

Table 6-5: **Receiver AC Timing Specifications - 2.5 Gbps Baud Rate**

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential input voltage	V_{IN}	200	1600	mV	Peak-to-peak differential input voltage
Deterministic jitter	J_D		0.37	UI	Measured at receiver.
Total jitter	J_T		0.65	UI	Measured at receiver
Input skew	S_I		75	ps	Skew at a receiver input between the two signals comprising a differential pair
Multiple input skew	S_{MI}		24	ns	Skew at the receiver input between lanes of a multi-lane channel
Bit error rate	BER		10^{-12}		
Unit interval	UI	400	400	ps	+/- 100 ppm

Note: AC coupling is required to guarantee interoperability between vendors.

Table 6-6: **Receiver AC Timing Specifications - 3.125 Gbps Baud Rate**

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential input voltage	V_{IN}	200	1600	mV	Peak-to-peak differential input voltage
Deterministic jitter	J_D		0.37	UI	Measured at receiver.
Total jitter	J_T		0.65	UI	Measured at receiver
Input skew	S_I		75	ps	Skew at a receiver input between the two signals comprising a differential pair
Multiple input skew	S_{MI}		24	ns	Skew at the receiver input between lanes of a multi-lane channel
Bit error rate	BER		10^{-12}		
Unit interval	UI	320	320	ps	+/- 100 ppm

Note: AC coupling is required to guarantee interoperability between vendors.

6.6 Receiver Eye Diagrams

The following receiver eye openings are required to ensure proper operation.

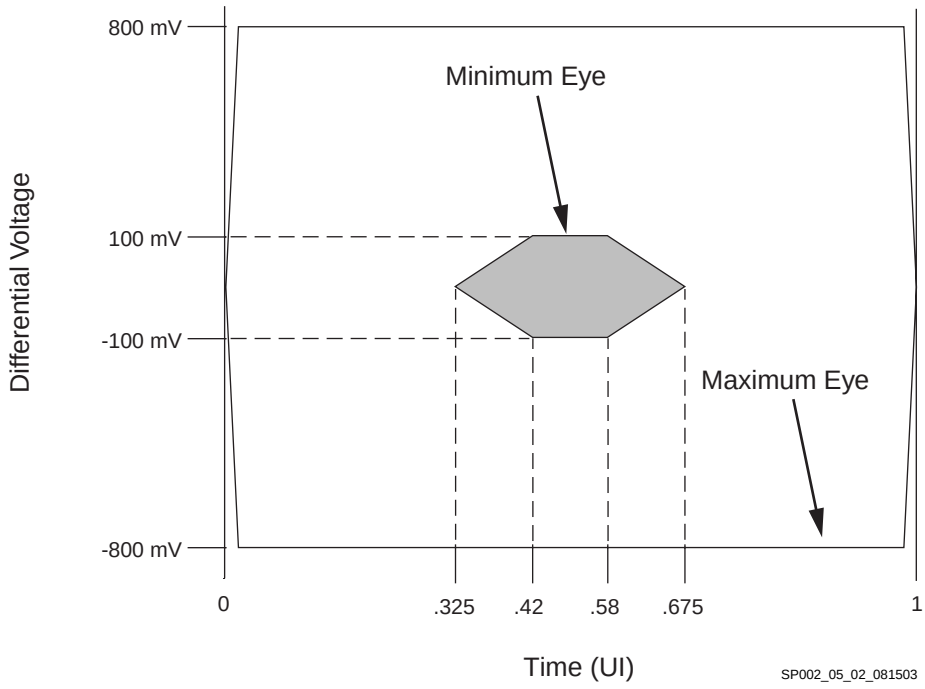


Figure 6-2: **1.25 Gbps Baud Rate Receiver Eye Diagram**

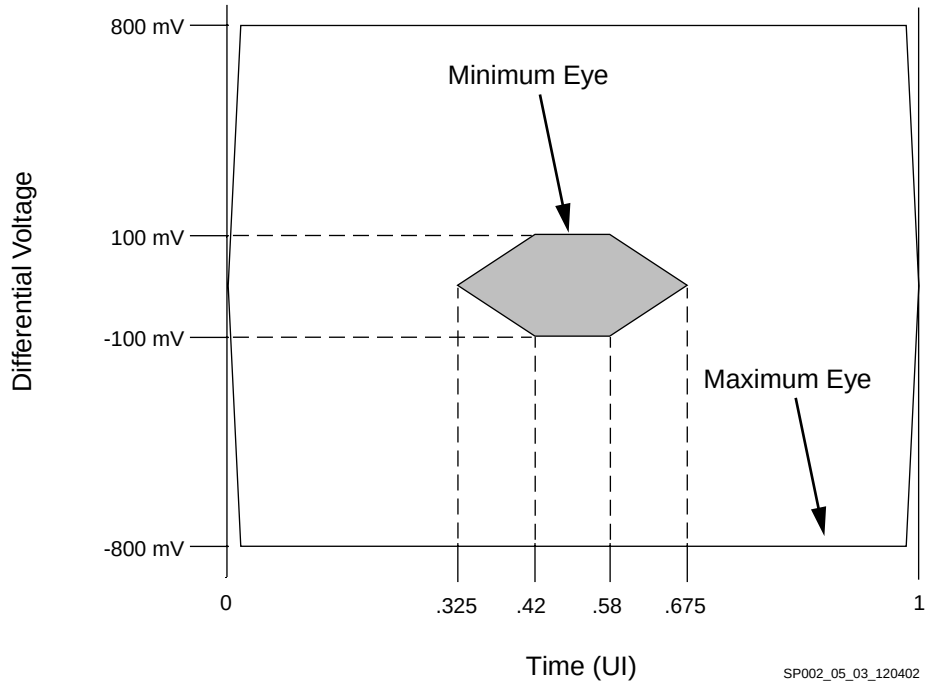


Figure 6-3: **2.5 Gbps Baud Rate Receiver Eye Diagram**

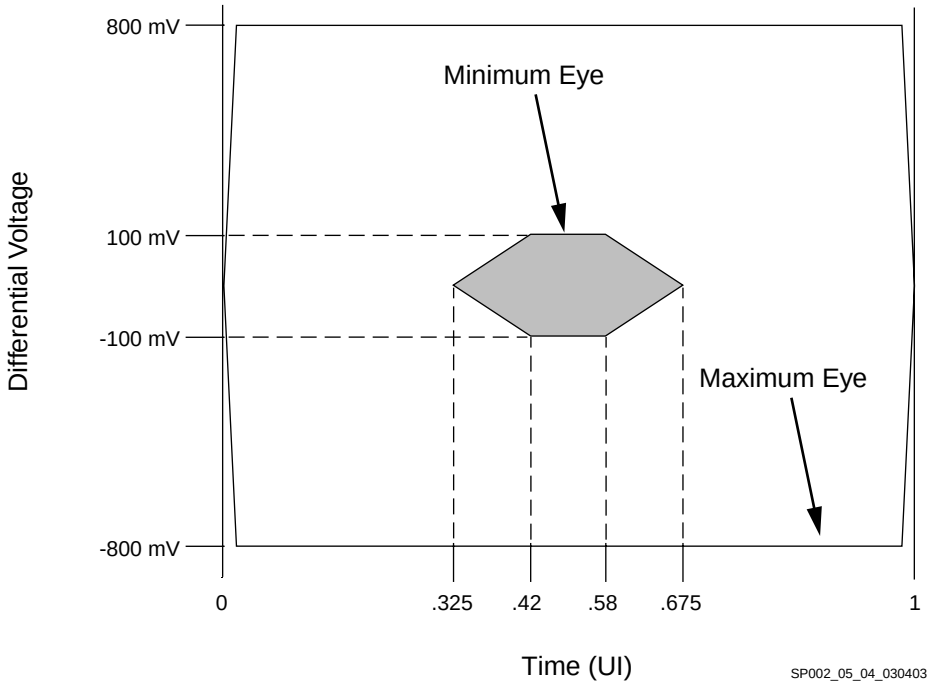


Figure 6-4: **3.125 Gbps Baud Rate Receiver Eye Diagram**

XILINX®

Appendix A

8B/10B Coding Reference

A.1 8B/10B Encoding

Aurora uses 8B/10B line coding, which includes *Data* characters and *K* characters. The 8-bit values are coded into 10-bit values, keeping the serial line DC balanced and providing sufficient line transitions for reliable clock recovery. K characters are special Data characters used to construct the ordered sets defined in Table 5-1, page 51. Table A-1, and Table A-2, page 90 show the tables of valid Data and K characters.

Table A-1: **Valid Data Characters**

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D0.0	000 00000	100111 0100	011000 1011
D1.0	000 00001	011101 0100	100010 1011
D2.0	000 00010	101101 0100	010010 1011
D3.0	000 00011	110001 1011	110001 0100
D4.0	000 00100	110101 0100	001010 1011
D5.0	000 00101	101001 1011	101011 0100
D6.0	000 00110	011001 1011	011001 0100
D7.0	000 00111	111000 1011	000111 0100
D8.0	000 01000	111001 0100	000110 1011
D9.0	000 01001	100101 1011	011010 0100
D10.0	000 01010	010101 1011	010101 0100
D11.0	000 01011	110100 1011	110100 0100
D12.0	000 01100	001101 1011	001101 0100
D13.0	000 01101	101100 1011	101100 0100
D14.0	000 01110	011100 1011	011100 0100
D15.0	000 01111	010111 0100	101000 1011
D16.0	000 10000	011011 0100	100100 1011
D17.0	000 10001	100011 1011	100011 0100
D18.0	000 10010	010011 1011	010011 0100

Table A-1: **Valid Data Characters *(Continued)***

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D19.0	000 10011	110010 1011	110010 0100
D20.0	000 10100	001011 1011	001011 0100
D21.0	000 10101	101010 1011	101010 0100
D22.0	000 10110	011010 1011	011010 0100
D23.0	000 10111	111010 0100	000101 1011
D24.0	000 11000	110011 0100	001100 1011
D25.0	000 11001	100110 1011	100110 0100
D26.0	000 11010	010110 1011	010110 0100
D27.0	000 11011	110110 0100	001001 1011
D28.0	000 11100	001110 1011	001110 0100
D29.0	000 11101	101110 0100	010001 1011
D30.0	000 11110	011110 0100	100001 1011
D31.0	000 11111	101011 0100	010100 1011
D0.1	001 00000	100111 1001	011000 1001
D1.1	001 00001	011101 1001	100010 1001
D2.1	001 00010	101101 1001	010010 1001
D3.1	001 00011	110001 1001	110001 1001
D4.1	001 00100	110101 1001	001010 1001
D5.1	001 00101	101001 1001	101011 1001
D6.1	001 00110	011001 1001	011001 1001
D7.1	001 00111	111000 1001	000111 1001
D8.1	001 01000	111001 1001	000110 1001
D9.1	001 01001	100101 1001	011010 1001
D10.1	001 01010	010101 1001	010101 1001
D11.1	001 01011	110100 1001	110100 1001
D12.1	001 01100	001101 1001	001101 1001
D13.1	001 01101	101100 1001	101100 1001
D14.1	001 01110	011100 1001	011100 1001
D15.1	001 01111	010111 1001	101000 1001
D16.1	001 10000	011011 1001	100100 1001
D17.1	001 10001	100011 1001	100011 1001

Table A-1: **Valid Data Characters *(Continued)***

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D18.1	001 10010	010011 1001	010011 1001
D19.1	001 10011	110010 1001	110010 1001
D20.1	001 10100	001011 1001	001011 1001
D21.1	001 10101	101010 1001	101010 1001
D22.1	001 10110	011010 1001	011010 1001
D23.1	001 10111	111010 1001	000101 1001
D24.1	001 11000	110011 1001	001100 1001
D25.1	001 11001	100110 1001	100110 1001
D26.1	001 11010	010010 1001	010110 1001
D27.1	001 11011	110110 1001	001001 1001
D28.1	001 11100	001110 1001	001110 1001
D29.1	001 11101	101110 1001	010001 1001
D30.1	001 11110	011110 1001	100001 1001
D31.1	001 11111	101011 1001	010100 1001
D0.2	010 00000	100111 0101	011000 0101
D1.2	010 00001	011101 0101	100010 0101
D2.2	010 00010	101101 0101	010010 0101
D3.2	010 00011	110001 0101	110001 0101
D4.2	010 00100	110101 0101	001010 0101
D5.2	010 00101	101001 0101	101011 0101
D6.2	010 00110	011001 0101	011001 0101
D7.2	010 00111	111000 0101	000111 0101
D8.2	010 01000	111001 0101	000110 0101
D9.2	010 01001	100101 0101	011010 0101
D10.2	010 01010	010101 0101	010101 0101
D11.2	010 01011	110100 0101	110100 0101
D12.2	010 01100	001101 0101	001101 0101
D13.2	010 01101	101100 0101	101100 0101
D14.2	010 01110	011100 0101	011100 0101
D15.2	010 01111	010111 0101	101000 0101
D16.2	010 10000	011011 0101	100100 0101

Table A-1: **Valid Data Characters *(Continued)***

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D17.2	010 10001	100011 0101	100011 0101
D18.2	010 01010	010011 0101	010011 0101
D19.2	010 10011	110010 0101	110010 0101
D20.2	010 10100	001011 0101	001011 0101
D21.2	010 10101	101010 0101	101010 0101
D22.2	010 10110	011010 0101	011010 0101
D23.2	010 10111	111010 0101	000101 0101
D24.2	010 11000	110011 0101	001100 0101
D25.2	010 11001	100110 0101	100110 0101
D26.2	010 11010	010010 0101	010110 0101
D27.2	010 11011	110110 0101	001001 0101
D28.2	010 11100	001110 0101	001110 0101
D29.2	010 11101	101110 0101	010001 0101
D30.2	010 11110	011110 0101	100001 0101
D31.2	010 11111	101011 0101	010100 0101
D0.3	000 00000	100111 0011	011000 1100
D1.3	011 00001	011101 0011	100010 1100
D2.3	011 00010	101101 0011	010010 1100
D3.3	011 00011	110001 1100	110001 0011
D4.3	011 00100	110101 0011	001010 1100
D5.3	011 00101	101001 1100	101011 0011
D6.3	011 00110	011001 1100	011001 0011
D7.3	011 00111	111000 1100	000111 0011
D8.3	011 01000	111001 0011	000110 1100
D9.3	011 01001	100101 1100	011010 0011
D10.3	011 01010	010101 1100	010101 0011
D11.3	011 01011	110100 1100	110100 0011
D12.3	011 01100	001101 1100	001101 0011
D13.3	011 01101	101100 1100	101100 0011
D14.3	011 01110	011100 1100	011100 0011
D15.3	011 01111	010111 0011	101000 1100

Table A-1: **Valid Data Characters *(Continued)***

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D16.3	011 10000	011011 0011	100100 1100
D17.3	011 10001	100011 1100	100011 0011
D18.3	011 10010	010011 1100	010011 0011
D19.3	011 10011	110010 1100	110010 0011
D20.3	011 10100	001011 1100	001011 0011
D21.3	011 10101	101010 1100	101010 0011
D22.3	011 10110	011010 1100	011010 0011
D23.3	011 10111	111010 0011	000101 1100
D24.3	011 11000	110011 0011	001100 1100
D25.3	011 11001	100110 1100	100110 0011
D26.3	011 11010	010110 1100	010110 0011
D27.3	011 11011	110110 0011	001001 1100
D28.3	011 11100	001110 1100	001110 0011
D29.3	011 11101	101110 0011	010001 1100
D30.3	011 11110	011110 0011	100001 1100
D31.3	011 11111	101011 0011	010100 1100
D0.4	100 00000	100111 0010	011000 1101
D1.4	100 00001	011101 0010	100010 1101
D2.4	100 00010	101101 0010	010010 1101
D3.4	100 00011	110001 1101	110001 0010
D4.4	100 00100	110101 0010	001010 1101
D5.4	100 00101	101001 1101	101011 0010
D6.4	100 00110	011001 1101	011001 0010
D7.4	100 00111	111000 1101	000111 0010
D8.4	100 01000	111001 0010	000110 1101
D9.4	100 01001	100101 1101	011010 0010
D10.4	100 01010	010101 1101	010101 0010
D11.4	100 01011	110100 1101	110100 0010
D12.4	100 01100	001101 1101	001101 0010
D13.4	100 01101	101100 1101	101100 0010
D14.4	100 01110	011100 1101	011100 0010

Table A-1: **Valid Data Characters *(Continued)***

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D15.4	100 01111	010111 0010	101000 1101
D16.4	100 10000	011011 0010	100100 1101
D17.4	100 10001	100011 1101	100011 0010
D18.4	100 10010	010011 1101	010011 0010
D19.4	100 10011	110010 1101	110010 0010
D20.4	100 10100	001011 1101	001011 0010
D21.4	100 10101	101010 1101	101010 0010
D22.4	100 10110	011010 1101	011010 0010
D23.4	100 10111	111010 0010	000101 1101
D24.4	100 11000	110011 0010	001100 1101
D25.4	100 11001	100110 1101	100110 0010
D26.4	100 11010	010010 1101	010110 0010
D27.4	100 11011	110110 0010	001001 1101
D28.4	100 11100	001110 1101	001110 0010
D29.4	100 11101	101110 0010	010001 1101
D30.4	100 11110	011110 0010	100001 1101
D31.4	100 11111	101011 0010	010100 1101
D0.5	101 00000	100111 1010	011000 1010
D1.5	101 00001	011101 1010	100010 1010
D2.5	101 00010	101101 1010	010010 1010
D3.5	101 00011	110001 1010	110001 1010
D4.5	101 00100	110101 1010	001010 1010
D5.5	101 00101	101001 1010	101001 1010
D6.5	101 00110	011001 1010	011001 1010
D7.5	101 00111	111000 1010	000111 1010
D8.5	101 01000	111001 1010	000110 1010
D9.5	101 01001	100101 1010	011010 1010
D10.5	101 01010	010101 1010	010101 1010
D11.5	101 01011	110100 1010	110100 1010
D12.5	101 01100	001101 1010	001101 1010
D13.5	101 01101	101100 1010	101100 1010

Table A-1: **Valid Data Characters *(Continued)***

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D14.5	101 01110	011100 1010	011100 1010
D15.5	101 01111	010111 1010	101000 1010
D16.5	101 10000	011011 1010	100100 1010
D17.5	101 10001	100011 1010	100011 1010
D18.5	101 01010	010011 1010	010011 1010
D19.5	101 10011	110010 1010	110010 1010
D20.5	101 10100	001011 1010	001011 1010
D21.5	101 10101	101010 1010	101010 1010
D22.5	101 10110	011010 1010	011010 1010
D23.5	101 10111	111010 1010	000101 1010
D24.5	101 11000	110011 1010	001100 1010
D25.5	101 11001	100110 1010	100110 1010
D26.5	101 11010	010010 1010	010110 1010
D27.5	101 11011	110110 1010	001001 1010
D28.5	101 11100	001110 1010	001110 1010
D29.5	101 11101	101110 1010	010001 1010
D30.5	101 11110	011110 1010	100001 1010
D31.5	101 11111	101011 1010	010100 1010
D0.6	110 00000	100111 0110	011000 0110
D1.6	110 00001	011101 0110	100010 0110
D2.6	110 00010	101101 0110	010010 0110
D3.6	110 00011	110001 0110	110001 0110
D4.6	110 00100	110101 0110	001010 0110
D5.6	110 00101	101001 0110	101011 0110
D6.6	110 00110	011001 0110	011001 0110
D7.6	110 00111	111000 0110	000111 0110
D8.6	110 01000	111001 0110	000110 0110
D9.6	110 01001	100101 0110	011010 0110
D10.6	110 01010	010101 0110	010101 0110
D11.6	110 01011	110100 0110	110100 0110
D12.6	110 01100	001101 0110	001101 0110

Table A-1: **Valid Data Characters *(Continued)***

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D13.6	110 01101	101100 0110	101100 0110
D14.6	110 01110	011100 0110	011100 0110
D15.6	110 01111	010111 0110	101000 0110
D16.6	110 10000	011011 0110	100100 0110
D17.6	110 10001	100011 0110	100011 0110
D18.6	110 01010	010011 0110	010011 0110
D19.6	110 10011	110010 0110	110010 0110
D20.6	110 10100	001011 0110	001011 0110
D21.6	110 10101	101010 0110	101010 0110
D22.6	110 10110	011010 0110	011010 0110
D23.6	110 10111	111010 0110	000101 0110
D24.6	110 11000	110011 0110	001100 0110
D25.6	110 11001	100110 0110	100110 0110
D26.6	110 11010	010010 0110	010110 0110
D27.6	110 11011	110110 0110	001001 0110
D28.6	110 11100	001110 0110	001110 0110
D29.6	110 11101	101110 0110	010001 0110
D30.6	110 11110	011110 0110	100001 0110
D31.6	110 11111	101011 0110	010100 0110
D0.7	111 00000	100111 0001	011000 1110
D1.7	111 00001	011101 0001	100010 1110
D2.7	111 00010	101101 0001	010010 1110
D3.7	111 00011	110001 1110	110001 0001
D4.7	111 00100	110101 0001	001010 1110
D5.7	111 00101	101001 1110	101011 0001
D6.7	111 00110	011001 1110	011001 0001
D7.7	111 00111	111000 1110	000111 0001
D8.7	111 01000	111001 0001	000110 1110
D9.7	111 01001	100101 1110	011010 0001
D10.7	111 01010	010101 1110	010101 0001
D11.7	111 01011	110100 1110	110100 1000

Table A-1: **Valid Data Characters *(Continued)***

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D12.7	111 01100	001101 1110	001101 0001
D13.7	111 01101	101100 1110	101100 1000
D14.7	111 01110	011100 1110	011100 1000
D15.7	111 01111	010111 0001	101000 1110
D16.7	111 10000	011011 0001	100100 1110
D17.7	111 10001	100011 0111	100011 0001
D18.7	111 10010	010011 0111	010011 0001
D19.7	111 10011	110010 1110	110010 0001
D20.7	111 10100	001011 0111	001011 0001
D21.7	111 10101	101010 1110	101010 0001
D22.7	111 10110	011010 1110	011010 0001
D23.7	111 10111	111010 0001	000101 1110
D24.7	111 11000	110011 0001	001100 1110
D25.7	111 11001	100110 1110	100110 0001
D26.7	111 11010	010110 1110	010110 0001
D27.7	111 11011	110110 0001	001001 1110
D28.7	111 11100	001110 1110	001110 0001
D29.7	111 11101	101110 0001	010001 1110
D30.7	111 11110	011110 0001	100001 1110
D31.7	111 11111	101011 0001	010100 1110

Table A-2: **Valid K (Control) Characters**

Special Code Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
K23.7	111 10111	111010 1000	000101 0111
K27.7	111 11011	110110 1000	001001 0111
K28.0	000 11100	001111 0100	110000 1011
K28.2	010 11100	001111 0101	110000 1010
K28.3	011 11100	001111 0011	110000 1100
K28.4	100 11100	001111 0010	110000 1101
K28.5	101 11100	001111 1010	110000 0101
K28.6	110 11100	001111 0110	110000 1001
K29.7	111 11101	101110 1000	010001 0111
K30.7	111 11110	011110 1000	100001 0111

Appendix B

Aurora Serial Simplex Operation

B.1 Overview

This appendix provides an addendum to the Aurora Protocol that specifies the use of the protocol in *Aurora serial simplex*(1) *operation*, or *simplex operation*. Except where indicated otherwise in this appendix, simplex operation is wholly defined by the *Aurora Protocol Specification*.

Simplex operation preserves all of the Aurora Protocol features for channel initialization, data transmission, flow control, and error handling across a single unidirectional channel, called the Aurora simplex channel. An Aurora simplex channel comprises one or more serial lanes carrying data from the transmit partner to the receive partner, as shown in Figure B-1. Channel PDUs and user flow control (UFC) PDUs are transferred across the Aurora simplex channel.

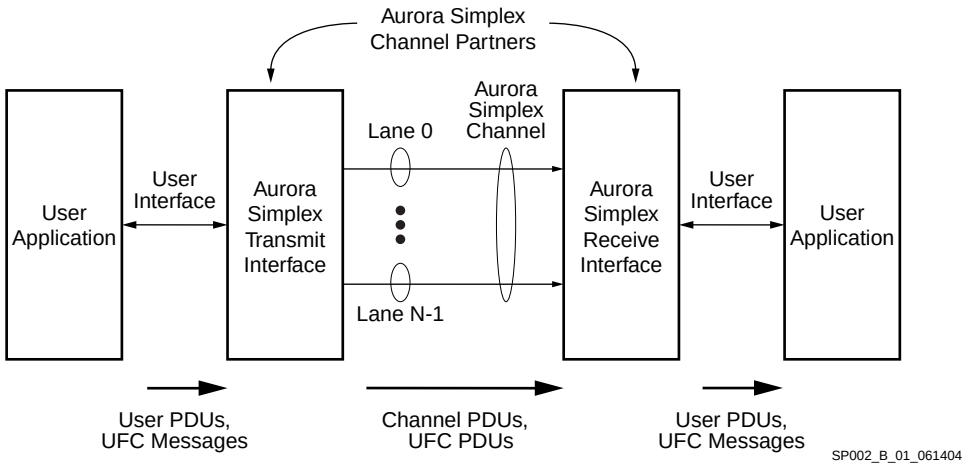


Figure B-1: **Aurora Simplex Channel Overview**

1. Simplex is used in the technical sense, defined as communication across a channel in a single direction. Full-duplex Aurora is used in this appendix to differentiate between Aurora and Aurora Simplex.

B.2 Data Transmission and Reception

The transmit user interface passes user PDUs and user flow control messages to the Aurora transmit interface for encapsulation and transmission across the Aurora simplex channel, as defined by the *Aurora Protocol Specification*. At the Aurora receive interface, the channel PDUs and user flow control PDUs are passed to the receiving user application as user PDUs and user flow control messages.

The rules governing data transmission from the transmit partner and reception at the receive partner are completely defined by Section 2 of the *Aurora Protocol Specification*, with the exception of the transmission of native flow control PDUs. Native flow control PDUs have no relevance when transferred across the Aurora simplex channel.

B.3 Flow Control

User flow control messages can be sent from the transmit partner to the receive partner, as defined in Section 3.1 Overview, Section 3.1.4 User Flow Control Operation, and Section 3.1.5 User Flow Control PDU Format of the *Aurora Protocol Specification*. The transmission of user flow control PDUs from the receive partner to the transmit partner is not specified by the Aurora serial simplex definition.

B.4 Initialization and Error Handling

Lane initialization, channel bonding for multi-lane channels, and lane verification are performed between the Aurora simplex channel partners using a modification of the procedures specified in Section 4 of the *Aurora Protocol Specification*. These modified procedures are illustrated in Figures B-2 through B-8, which show initialization state machines for the transmit and receive partners. The state machines include new inputs and outputs to support simplex operation.

B.5 Serial Simplex versus Full-Duplex Operation

This section details the differences between Aurora simplex operation and Aurora full-duplex operation. Refer to the State Diagram Conventions, page 12 for Figures B-2 through B-8 in the following sections.

B.5.1 Transmit Interface Lane initialization

Each lane, whether in Aurora full-duplex operation or in simplex operation, first undergoes lane initialization. However, simplex operation requires different state machines than full-duplex operation in the transmit and receive interfaces.Table B-1 shows the inputs and outputs that are necessary for simplex operation. These signals are shown in uppercase in the state diagrams that follow.

Table B-1: **Signals for Simplex Operation**

Signal	Direction	Description
TX_ALIGNED	Input	Transmit Lane Initialization state machine input. Moves the transmit interface past the alignment procedure.
RX_ALIGNED	Output	Receive Lane Initialization output. Asserted when alignment is reached at the receiver.
TX_BONDED	Input	Transmit Channel Bonding state machine input. Moves the transmit interface past the channel bonding procedure.
RX_BONDED	Output	Receive Channel Bonding state machine output. Asserted when channel bonding condition is met.
TX_VERIFIED	Input	Transmit Channel Verification state machine input. Moves the transmit interface past the channel verification procedure.
RX_VERIFIED	Output	Receive Channel Verification state machine output. Asserted when the channel verification criteria are met.
TX_RESET	Input	Transmit Lane Initialization, Channel Bonding, and Channel Verification Procedures state machine input. Returns transmit interface to TXRESET0.
RX_RESET	Output	Receive Lane Initialization, Channel Bonding, Channel Verification, and Hard Error Procedures state machine input. Asserted when an error condition occurs causing the receive interface to return to RXRESET0.

Figure B-2 shows the state diagram for the lane initialization procedure of the transmit interface. If the RESET input is asserted at any point in the initialization procedure, then the state of the transmit interface must return to TXRESET0, as indicated in Figure B-2.

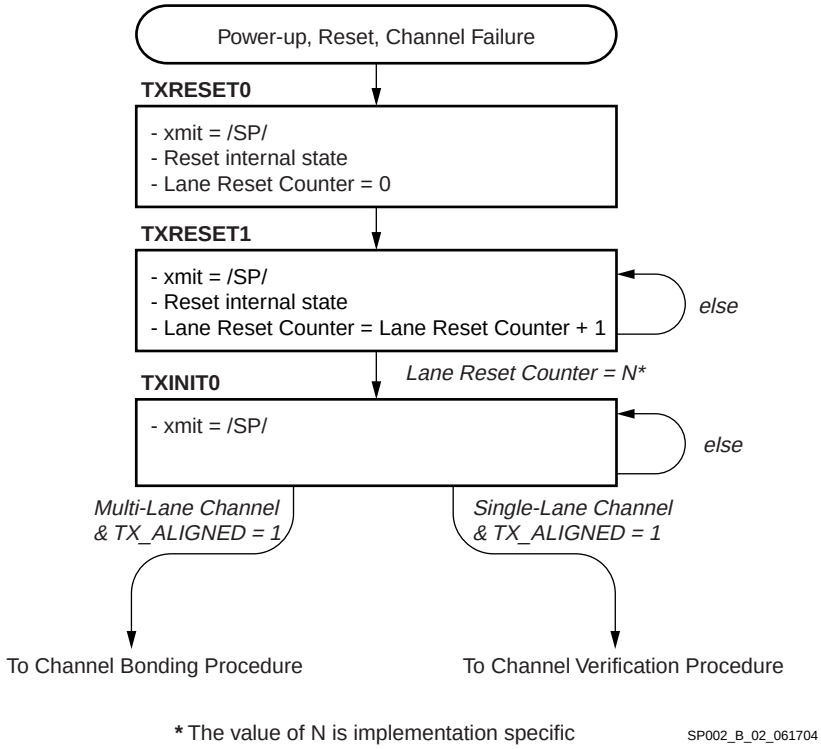


Figure B-2: **Transmit Interface Lane Initialization Procedure**

B.5.2 Receive Interface Lane Initialization

Figures B-3 shows the state diagram for the lane initialization procedure of the transmit interface.

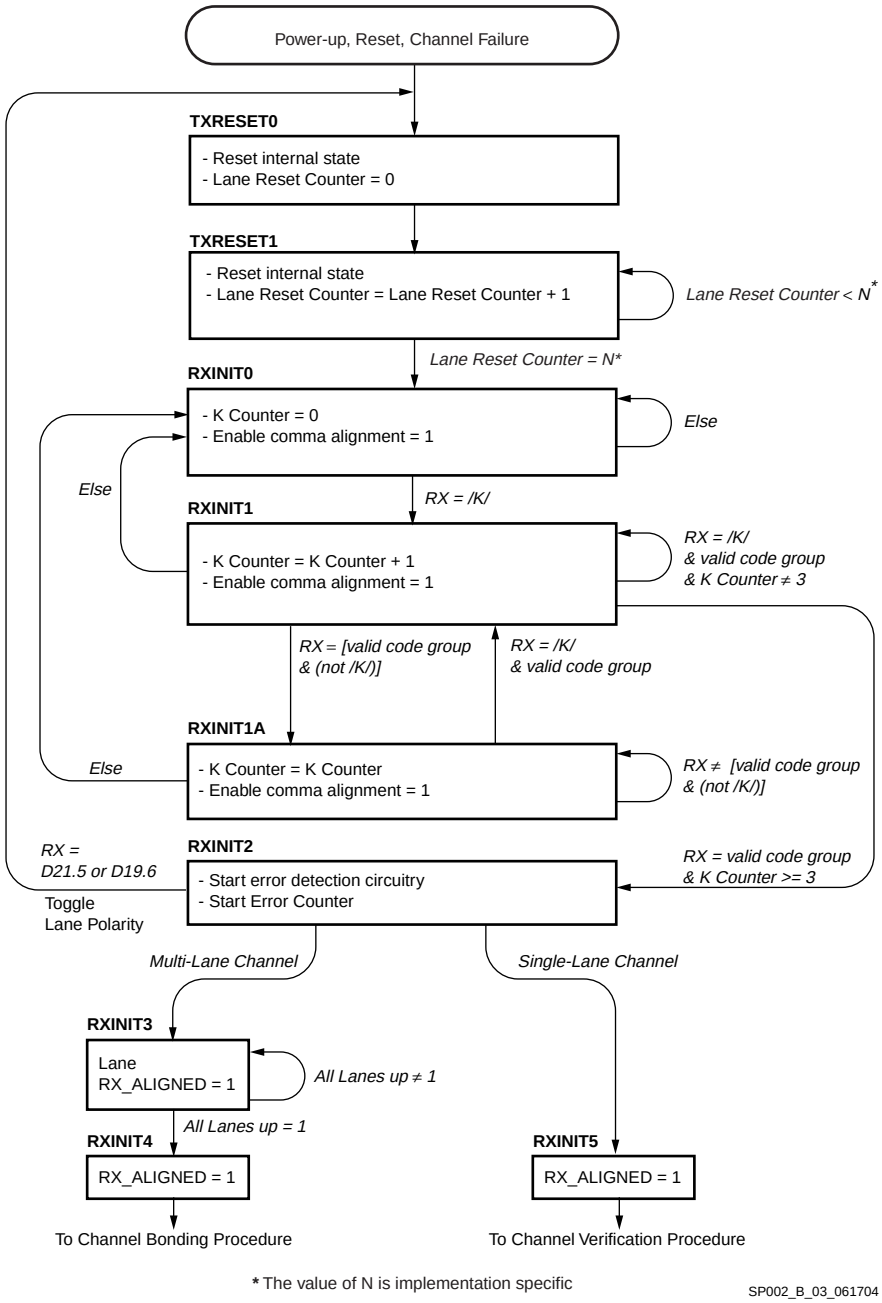


Figure B-3: **Receive Interface Lane Initialization**

B.5.3 Transmit Interface Channel Bonding Procedure

Figure B-4 shows the transmit interface procedure for channel bonding.

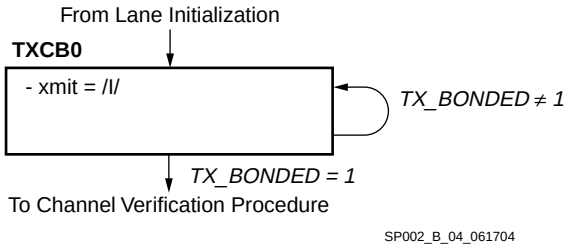


Figure B-4: **Transmit Interface Channel Bonding Procedure**

B.5.4 Receive Interface Channel Bonding Procedure

Figure B-5 shows the state diagram for the receive interface channel bonding procedure.

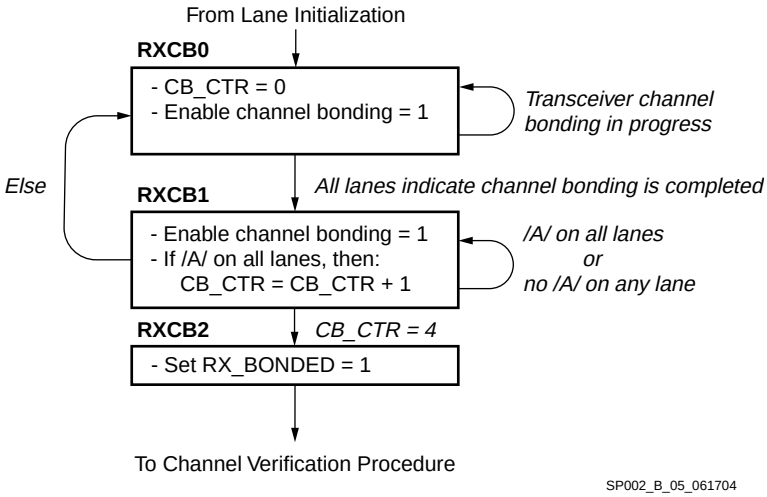


Figure B-5: **Receive Interface Channel Bonding Procedure**

B.5.5 Transmit Interface Channel Verification Procedure

Figure B-6 shows the transmit interface procedure for channel verification.

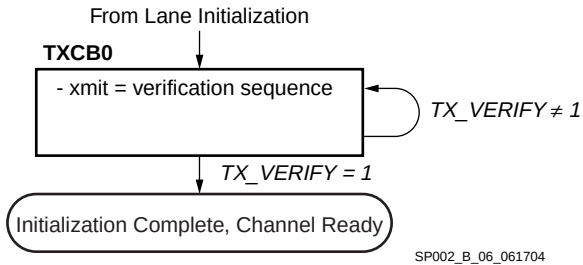


Figure B-6: **Transmit Interface Channel Verification Procedure**

B.5.6 Receive Interface Channel Verification Procedure

Figure B-7 shows the receive interface procedure for channel verification.

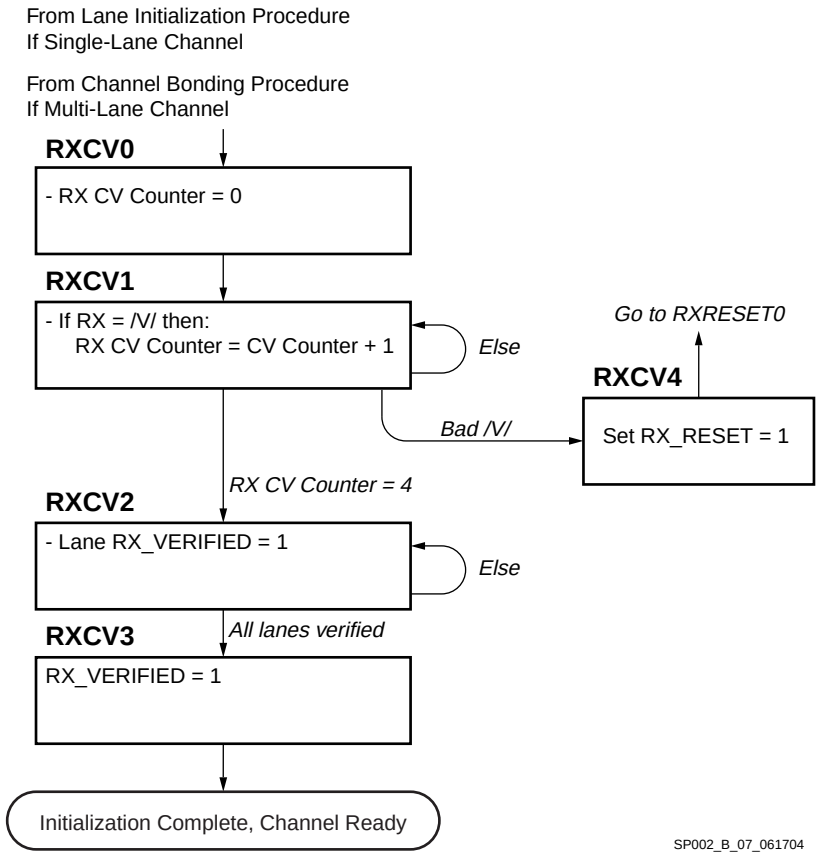


Figure B-7: **Receive Interface Channel Verification Procedure**

B.5.7 Receive Interface Hard Error Procedure

If a hard error is encountered in the receiver at any point during the initialization or data transmission procedures, then the RX_RESET output is asserted at the receive interface, and the receiver returns to the reset state RXRESET0. Figure B-8 shows the receive interface hard error procedure.

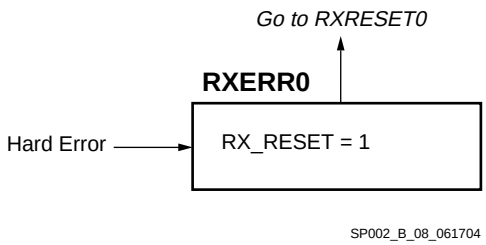


Figure B-8: **Receiver Interface Hard Error Procedure**

B.5.8 Use Models for Aurora Serial Simplex

Two use models are envisaged for Aurora serial simplex.

The first use model includes a user-defined back channel that conveys the receive interface status, as represented by the receive interface outputs defined in this appendix, back to the transmit interface. The details of the back channel implementation is highly application specific and is outside of the scope of this specification.

The second use-model does not include a back channel. In this second case, the control of the additional inputs to the receive interface as defined in this appendix is user-defined. The interface designer must ensure that this control guarantees the Aurora simplex channel operation during normal operation.

Glossary

Click on a letter, or scroll down to view the entire glossary.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

Aurora Interface

An implementation of the Aurora Protocol.

B

C

Channel

The communication link between two Aurora interfaces, comprising one or more lanes.

Character

A 9-bit entity comprised of an information octet and a control bit that indicates whether the information octet contains data or control information. The control bit has the value *D* or *K* indicating that the information octet contains data or control information respectively.

Clock Compensation

A compensating mechanism provided by the Aurora Protocol for clock rate differences between the transmitter and receiver.

Clock Encoding

A method used to transmit a clock in band with data.

Code group

A 10-bit entity that is the result of 8B/10B encoding a character. A *code group* is also called a *symbol*.

Column

A group of characters that are transmitted simultaneously on a multi-lane channel.

Comma

A seven-bit encoded 8B/10B sequence that is either 1100000 or 0011111 which is used for alignment. The seven-bits are the most significant bits of a 10-bit code group. A comma is contained in the 8B/10B symbol designated as /K28.5/.

Control Character

A character whose control bit has the value *K*.

Current Mode Logic

Current Mode Logic (CML) is a differential logic operation using devices that are commonly grounded through a current source.

D

Data Character

A character whose control bit has the value *D*.

Data Striping

This describes how data is mapped across a channel consisting of multiple lanes.

Destriping

The opposite of striping; reverse of striping.

Deterministic Jitter

Deterministic jitter (DJ) is a jitter with a non-Gaussian probability density function. Deterministic jitter is always bounded in amplitude and has specific causes. If a sufficient amount of data is taken over a complete cycle of each periodic element, the total amount of DJ will remain constant.

Disparity

The difference between the number of ones and zeros in a symbol.

Disparity Error

A received code group whose value is inconsistent with the current running disparity.

E

F

G

H

I

Idle Sequence

The sequence of characters (code groups after encoding) that is transmitted when a PDU is not being transmitted. The idle sequence allows the receiver to maintain bit synchronization and code group alignment.

J

K

K Character

A character whose control bit has the value *K*. Also referred to as a special character.

L

Lane

A full duplex physical serial connection.

Lane Alignment

The process of eliminating the skew between the lanes of a multi-lane Aurora channel such that the characters transmitted as a column by the sender are output as a column by the alignment process of the receiver. Without lane alignment, the characters transmitted as a column might be scattered across several columns output by the receiver. During initialization, all lanes continuously transmit columns of the idle sequence. The alignment process in the receiver uses the /A/ ordered sets to realign the columns of received data.

Link Layer

This describes how the beginning and end of user protocol data units (user PDUs) are marked during transmission. It also describes how data pauses may be inserted in data during transmission and how differences in clock rates between the transmitter and receiver are managed. Equivalent to the OSI Link Layer.

Link Layer Flow Control

A mechanism for throttling the flow of user PDU in the link layer.

Link Layer Payload

The resulting data structure after padding.

M

N

Not-in-table Error

A received code group that does not exist in the 8B/10B table. Same as a *symbol error*.

O

Octet

An 8-bit unit of information. Each bit of a octet has the value 0 or 1.

P

PCS

See Physical Coding Sublayer.

Physical Coding Sublayer (PCS)

The physical coding sublayer (PCS) function is responsible for idle sequence generation, lane striping, and encoding for transmission. It is also responsible for decoding, lane alignment, and destriping upon reception. The PCS uses an 8B/10B encoding for transmission over the channel. The PCS layer also provides mechanisms to detect lane states. It provides for clock difference tolerance between the sender and receiver without requiring flow control.

Physical Layer Interface

This consists of the electrical levels, the clock encoding, and symbol coding.

Physical Medium Attachment (PMA)

The physical medium attachment (PMA) function is responsible for serializing 10-bit parallel code groups to/from a serial bitstream on a lane-by-lane basis.

PMA

See Physical Medium Attachment.

Q

R

Running Disparity

The cumulative disparity of a sequence of symbols. The binary variable used by the 8B/10B encoding and decoding functions.

S

SNF

See Start of Native Flow control.

Start of Native Flow Control (SNF)

The /SNF/ ordered set is used in the protocol to mark the start of a native flow control PDU. A native flow control PDU is sent in response to a user's request. A native flow control PDU is always comprised of exactly two symbols.

Start of User Flow Control (SUF)

The SUF ordered set is used in the protocol to mark the start of a user flow control PDU. A user flow control PDU is comprised of the /SNF/ ordered set plus a data symbol that defines the length of the PDU, followed by up to 16 symbols of user data.

Striping

Striping allocates byte-pairs across multiple lanes. The method used on multi-lane channels to send data simultaneously across all n lanes that make up the channel . The byte stream is striped across the lanes, on a byte-pair by byte-pair basis, starting with the first byte-pair on lane 0, to the second byte-pair on lane 1, to the third byte-pair on lane

2, to the nth byte-pair on lane n-1, and wrapping back with the nth+1 byte-pair on lane 0.

Stripping

The method to remove data from a PDU.

SUF

See Start of User Flow control.

Symbol

A 10-bit entity that is the result of 8B/10B encoding a character. A *symbol* is also called a *code group*.

Symbol Error

A received code group that does not exist in the 8B/10B table. Same as a *not-in-table error*.

Symbol Time

Equivalent to 10 unit times.

T

Total Jitter

The total deviation from the ideal timing of a clock or data signal as a result of noise, patterns, or other causes.

U

User Application

An implementation of a higher level function that transports data across an Aurora channel.

User Interface

An implementation-specific interface provided to the user application by an Aurora interface.

User PDUs

User Protocol Data Units.

Unit Interval

The unit interval (UI) is the total time of one ideal clock cycle. For example, an Aurora channel operating at a data rate of 3.125 Gbps has a UI of 320 ps.

V

Valid Code Group

A received code group that when decoded has neither a *disparity* error nor a *not-in-table* error. A disparity error is defined as a received code group whose value is inconsistent with the current running disparity. A not-in-table error is defined as a received code group that does not exist in the 8B/10B table.

W

X

XON

A command that resumes transmission in a single direction used by native flow control

XOFF

A command that suspends transmission in a single direction used by native flow control

Y

Z

References

1. *BYTE ORIENTED DC BALANCED (0,4) 8B/10B PARTITIONED BLOCK TRANSMISSION CODE* (Patent Number: 4,486,739). Inventors: Peter A. Franaszek and Albert X. Widmer (IBM)