# Aurora 64B/66B Protocol Specification

SP011 (v1.3) October 1, 2014

XILINX®

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 10/01/2014 | 1.3 | Revised Figure 4-2 and Figure 5-7.<br>Updated Section 4.2.4 "Channel Ready," Section 6.4 "Frame Data Striping," Section 6.5 "Strict-Alignment Frame Data Striping" and Section 6.7 "Strict-Alignment User Flow Control Striping".<br>Added Section 6.9 "Strict-Alignment User K-Block Striping". |
| 07/23/2010 | 1.2 | Updated Section 1.2 "Scope" and Section 8.1 "Overview"<br>Deleted Sections 8.4 Transmitter Specifications, 8.5 Receiver Specifications, and 8.6 Receiver Eye Diagrams. |
| 09/19/2008 | 1.1 | Minor typographical edits. Changed *block codes* to *blocks*. Removed *Not Ready blocks* from Simplex in Table 4-1, page 36. Clarified simplex Aurora channel bonding in Section 4.2.2 "Channel Bonding," page 36. Added Appendix 1, "References." |
| 03/31/2008 | 1.0 | Initial Xilinx release. |

# Table of Contents

# Schedule of Figures

# Schedule of Tables

# About This Specification

This specification describes the Aurora 64B/66B protocol. Aurora is a lightweight link-layer protocol that can be used to move data point-to-point across one or more high-speed serial lanes. Aurora 64B/66B is a version of the protocol using 64B/66B encoding instead of 8B/10B.

## Specification Contents

This manual contains the following:

- Section 1, "Introduction and Overview"provides an overview of the Aurora 64B/66B protocol.
- Section 2, "Data Transmission and Reception"describes the procedures for transmitting and receiving data using an Aurora 64B/66B Channel.
- Section 3, "Flow Control"describes the optional flow control features in the Aurora 64B/66B protocol.
- Section 4, "Initialization and Error Handling"describes the procedure used to prepare an Aurora channel for operation.
- Section 5, "PCS Layer"specifies the functions performed in the physical coding sub-layer (PCS) of the Aurora 64B/66B protocol.
- Section 6, "Channel Control"defines the striping rules for using multi-lane channels.
- Section 7, "PMA Layer"specifies the functions performed in the PMA layer of the Aurora 64B/66B Protocol.
- Section 8, "Electrical Specifications"describes the AC specifications, covering both single- and multi-lane implementations.

## Conventions

This document uses the following conventions.

### Typographical

The following typographical conventions are used in this document:

| Convention | Meaning or Use | Example |
|---|---|---|
| *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. |
| | To emphasize a term the first time it is used | The state machine uses *one-hot* encoding. |
| 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 document | See the section "Additional Resources" for details.<br>Refer to "Title Formats" in Section 1 for details. |
| Red text | Cross-reference link to a location in another document | See Figure 2-5 in the *Virtex-II Platform FPGA User Guide.* |
| Blue, underlined text | Hyperlink to a website (URL) | Go to http://www.xilinx.com for the latest speed files. |

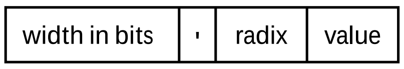
### Numerical

| Convention | Meaning or Use |
|---|---|
| *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 and Table P-2:

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



SP000_PF_01_031408

*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'b1x` | `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'hz1` | `zzzz0001` | An 8-bit hexadecimal number with the upper four bits not driven and the lower four bits having value of one. |
| `8'bx1` | `xxxxxxx1` | An 8-bit binary number that is odd. |
| `8'bx0` | `xxxxxxx0` | An 8-bit binary number that is even. |

*Table P-2:* **Examples of Extended Values *(Cont'd)***

| Number | Value | Comment |
|---|---|---|
| `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 P-2, page 15.

### 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

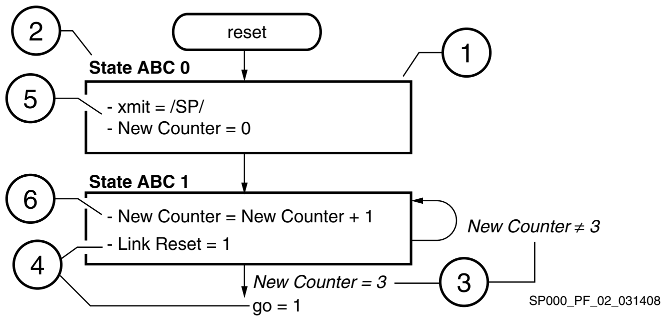


*Figure P-2:* **State Machine Diagram Conventions**

XILINX®

Section 1

# Introduction and Overview

## 1.1 Introduction

Aurora is a lightweight link-layer protocol that can be used to move data point-to-point across one or more high-speed serial lanes. Aurora 64B/66B is a version of the protocol using 64B/66B encoding instead of 8B/10B.

## 1.2 Scope

The *Aurora 64B/66B Protocol Specification* defines the following:

- **Electrical specifications:** This includes signaling levels for an Aurora serial link.
- **PMA layer:** This includes specification for serialization bit ordering and byte ordering.
- **Physical coding sub-layer (PCS):** This includes specification for data encoding and decoding, data scrambling, the 64B/66B gearbox, clock compensation and channel bonding.
- **Channel control:** This includes specifications for multi-lane striping and for scheduling the transmission of data and control information.
- **Cyclic redundancy check (CRC)**: The Aurora protocol recommends a CRC mechanism compatible with the standard 64B/66B scrambling algorithm.

## 1.3 Overview

The Aurora protocol (Figure 1-1, page 18) describes the transfer of user data across an Aurora channel, consisting of one or more Aurora lanes. Each Aurora lane is a serial data connection, either full-duplex or simplex. Devices communicating across the channel are called channel partners.

Aurora interfaces allow user applications to transfer data through the Aurora channel. The user interface on each Aurora interface is not defined in this specification and can be decided independently for each implementation of the protocol.

Aurora channels have the following properties:

- Data is transferred through the Aurora channel in frames.
- Frames share the channel with control information such as flow control messages, clock compensation sequences and idles.
- Frames can be of any length, and can have any format. Only the delineation of frames is defined in this specification.
- Frames in Aurora do not have to be contiguous — they can be interrupted at any time by flow control messages or idles.

There is no gap required between frames in Aurora.

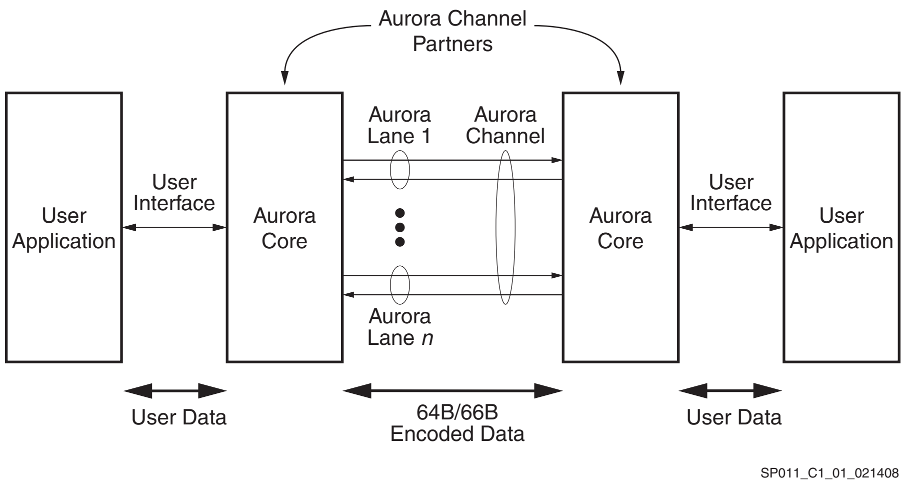


*Figure 1-1:* **Aurora Protocol Overview**

Figure 1-2 shows a simplex connection between a pair of Aurora lanes, depicting the functional blocks comprising the PCS and PMA layers of an Aurora connection. These blocks are specified in detail in this document.

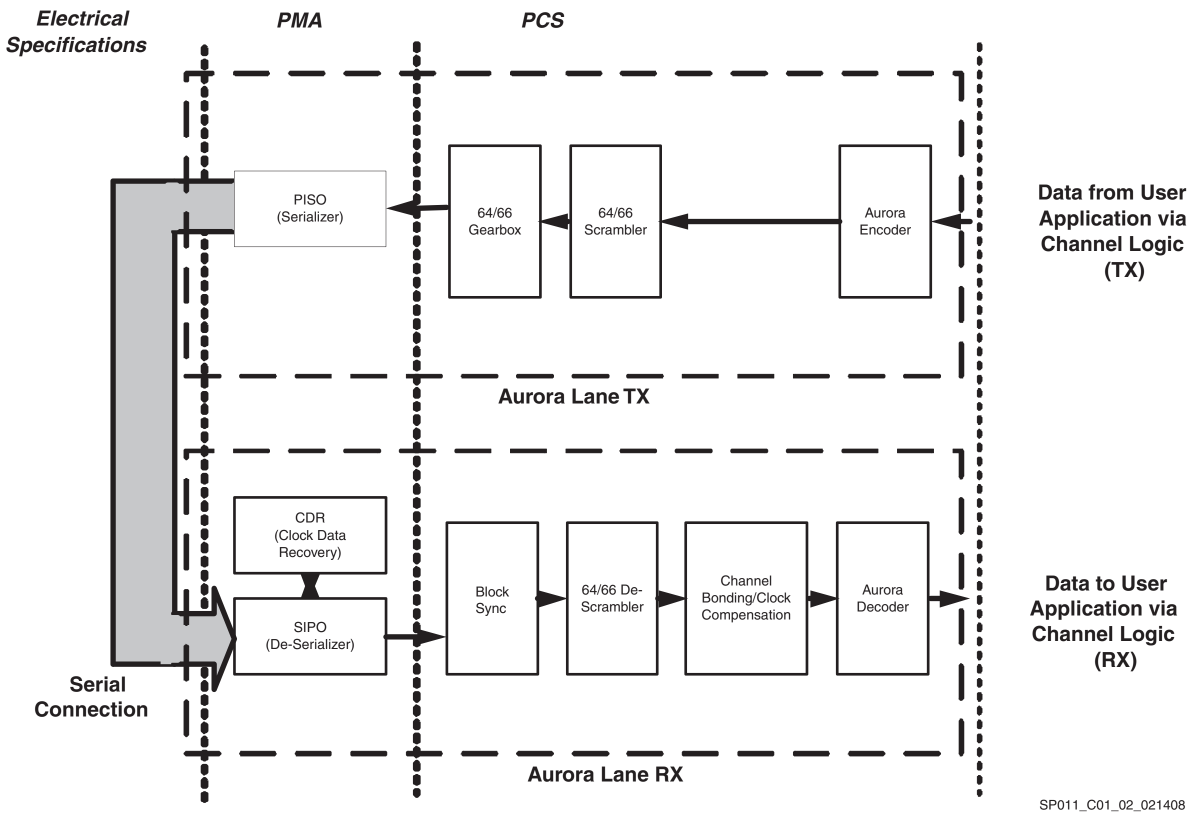


*Figure 1-2:* **A Simplex Connection Between a Pair of Aurora Lanes**

Aurora interfaces allow applications to communicate using Aurora channels. Aurora interfaces are made up of one or more Aurora lanes, either simplex or full-duplex. The four possible configurations of Aurora interfaces are shown in Figure 1-3, Figure 1-4, Figure 1-5, page 20, and Figure 1-6, page 20.

Figure 1-3 shows a single-lane, simplex Aurora interface transmitting to another single-lane, simplex Aurora interface. In this configuration, each interface uses a single lane to transmit or receive from the Aurora channel. Channel control in each interface initializes the channel passing control to the user application.

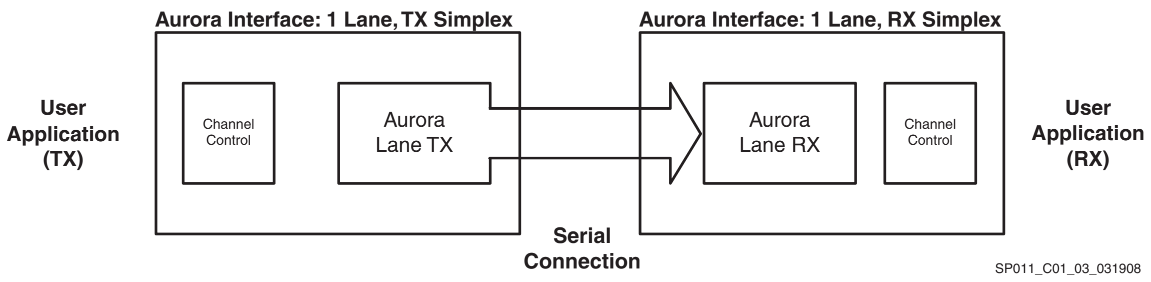


*Figure 1-3:* **A Single-Lane, Simplex Aurora Channel**

Figure 1-4 shows a multi-lane, simplex Aurora interface transmitting to another multi-lane, simplex Aurora interface. In multi-lane configurations, the channel control bonds the lanes to eliminate skew between channels as a part of the channel initialization procedure. During normal operation, the channel control logic distributes data and control information across all the lanes in the channel.

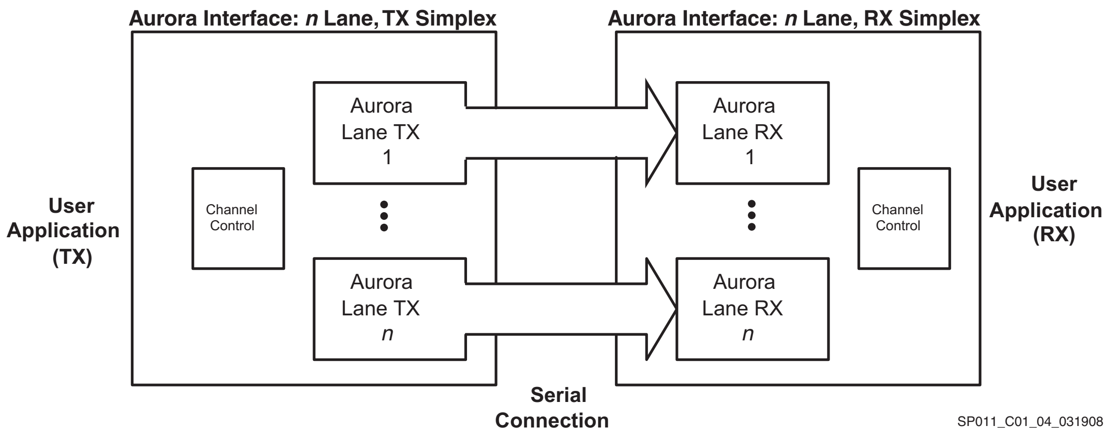


*Figure 1-4:* **A Multi-Lane, Simplex Aurora Channel**

Figure 1-5 shows a pair of single-lane, full-duplex Aurora interfaces. Full-duplex Aurora interfaces can transmit and receive data from the Aurora channel — their lanes have both TX and RX functional blocks. The channel control logic performs the same initialization procedure as for a simplex lane, but in both directions. Additionally, control information is used to allow each Aurora interface to detect whether the other side is prepared to receive data.

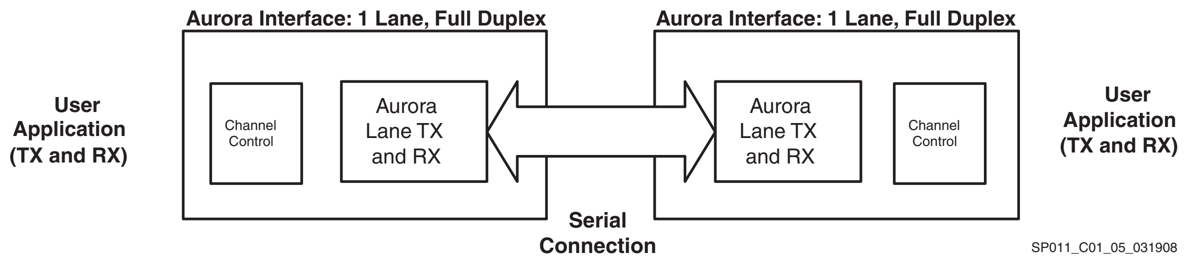

Aurora Interface: 1 Lane, Full Duplex
Aurora Interface: 1 Lane, Full Duplex
User Application (TX and RX)
Channel Control
Aurora Lane TX and RX
Aurora Lane TX and RX
Channel Control
User Application (TX and RX)
Serial Connection
SP011_C01_05_031908

*Figure 1-5:* **A Single-Lane, Full-Duplex Aurora Channel**

Figure 1-6 shows a pair of multi-lane, full-duplex Aurora interfaces. Full-duplex Aurora channels can also be made up of any number of lanes, bonded during initialization to create a single logical channel with higher throughput.

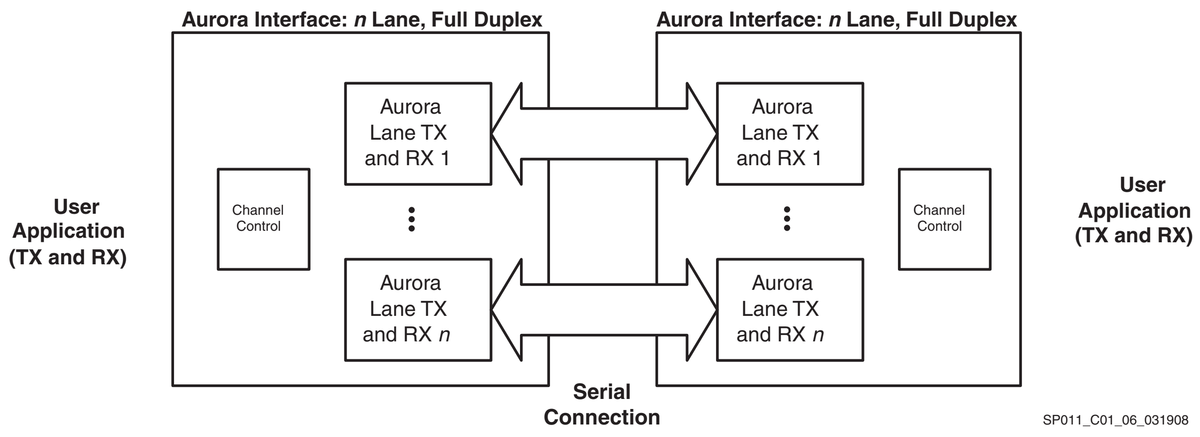

Aurora Interface: *n* Lane, Full Duplex
Aurora Interface: *n* Lane, Full Duplex
User Application (TX and RX)
Channel Control
Aurora Lane TX and RX 1
Aurora Lane TX and RX *n*
Aurora Lane TX and RX 1
Aurora Lane TX and RX *n*
Channel Control
User Application (TX and RX)
Serial Connection
SP011_C01_06_031908

*Figure 1-6:* **A Multi-Lane, Full-Duplex Aurora Channel**

# Data Transmission and Reception

## 2.1 Overview

This section describes the procedures for transmitting and receiving data using an Aurora 64B/66B channel.

## 2.2 Block Codes

All transmissions in Aurora 64B/66B are performed using 64-bit codes called a *block*. There are ten types of blocks that can be transmitted through an Aurora channel. On a given cycle, only one block can be transmitted through a lane. For this reason, the blocks in Aurora are prioritized — when two or more blocks need to be sent on the same cycle on the same lane in a channel, the highest priority block is always selected. Table 2-1 summarizes the blocks available in Aurora 64B/66B.

*Table 2-1:* **Aurora 64B/66B Blocks Summary**

| Block Name | Description |
|---|---|
| Clock Compensation | A special idle block that can be deleted or replicated by the Aurora interface that receives it. Clock Compensation blocks are used for asynchronous channels and are not needed for channels with shared clocks. See Section 5.6 "Clock Compensation." |
| Not Ready | Full-duplex Aurora interfaces send this special idle block while attempting to align data from the channel and perform channel bonding. See Section 4, "Initialization and Error Handling." |
| Channel Bonding | A special idle block used for channel bonding. Aurora interfaces transmit Channel Bonding blocks on every lane in the channel simultaneously. The lanes on the receiver are required to adjust their channel bonding FIFOs to receive these blocks simultaneously, thus correcting for skew between the lanes. See Section 5.5 "Channel Bonding." |
| Native Flow Control | This block requests native flow control from the Aurora interface on the other side of the channel. When an Aurora interface receives this message, it begins a native flow control countdown during which Data, Separator and Separator-7 blocks have the lowest priority. See Section 3.2 "Native Flow Control Operation." |
| User Flow Control | This block is used to start a user flow control message and contains a field indicating how many octets that follow are a part of the message (up to 256). While a user flow control message is in progress, Data, Separator and Separator-7 blocks drop to priority 8 (lowest priority), and idles rise to priority 7. See Section 3.5 "User Flow Control Operation." |

Table 2-1: **Aurora 64B/66B Blocks Summary *(Cont'd)***

| Block Name | Description |
|---|---|
| User K-Blocks | Aurora 64B/66B includes several control blocks not decoded by the Aurora interface, but are instead passed directly to the user. These blocks can be used to implement application-specific control functions. There are nine available User K-Blocks. See Section 5.2.9 "User K-Block Codes." |
| Data, Separator, Separator-7 | These blocks are combined to create frames carrying user data.<br>• Data blocks carry eight octets of data.<br>• Separator blocks indicate the end of the current frame; the next frame begins on the next block. Separator blocks carry 0 to 6 octets of data: one byte in the block is used to indicate how many octets in the block are valid.<br>• Separator-7 blocks are the same as Separator blocks, but always carry exactly seven valid octets of data.<br>See Section 2.5 "Data and Separator Block Format." |
| Idle | Idle blocks are transmitted whenever a higher priority block is not available for the channel. Idles are normally the lowest priority block, but when an Aurora interface is responding to a native flow control request, their priority moves up to 7, above Data blocks, Separator blocks and Separator-7 blocks. |

Table 2-2 shows the relative priority of blocks for transmission, valid at all times, except when the transmitter is processing a flow control request.

Table 2-2: **Normal Aurora 64B/66B Block Transmission Priority**

| Block Name | Priority |
|---|---|
| Clock Compensation | 1 (Highest) |
| Not Ready | 2 |
| Channel Bonding | 3 |
| Native Flow Control | 4 |
| User Flow Control, Data blocks carrying UFC message | 5 |
| User K-Blocks | 6 |
| Data, Separator, Separator-7 | 7 |
| Idle | 8 |

The priority of the blocks shifts when the transmitter is processing a flow control request. During native flow control countdown or user flow control countdown, idles have a higher priority than data. Since idles are always transmitted when no higher priority block is available, this shift in priority effectively prevents frame transmission until the flow control countdown is complete (see Section 3, "Flow Control" for details). Table 2-3 shows the relative priority of the blocks while a flow control countdown is in progress.

*Table 2-3:* **Aurora 64B/66B Block Transmission Priority during Flow Control Countdown**

| Block Name | Priority |
|---|---|
| Clock Compensation | 1 (Highest) |
| Not Ready | 2 |
| Channel Bonding | 3 |
| Native Flow Control | 4 |
| User Flow Control, Data blocks carrying UFC message | 5 |
| User K-Blocks | 6 |
| Idle | 7 |
| Data, Separator, Separator-7 | 8 |

## 2.3 Frame Transmission Procedure

The Aurora interface performs the following procedure to transmit a frame from the user application through an initialized Aurora channel:

- Frames are delineated using Separator and Separator-7 blocks
- Blocks are 64B/66B encoded
- Data stream is serialization and the clock embedded

Figure 2-1 illustrates how frames are mapped to block codes by this procedure.

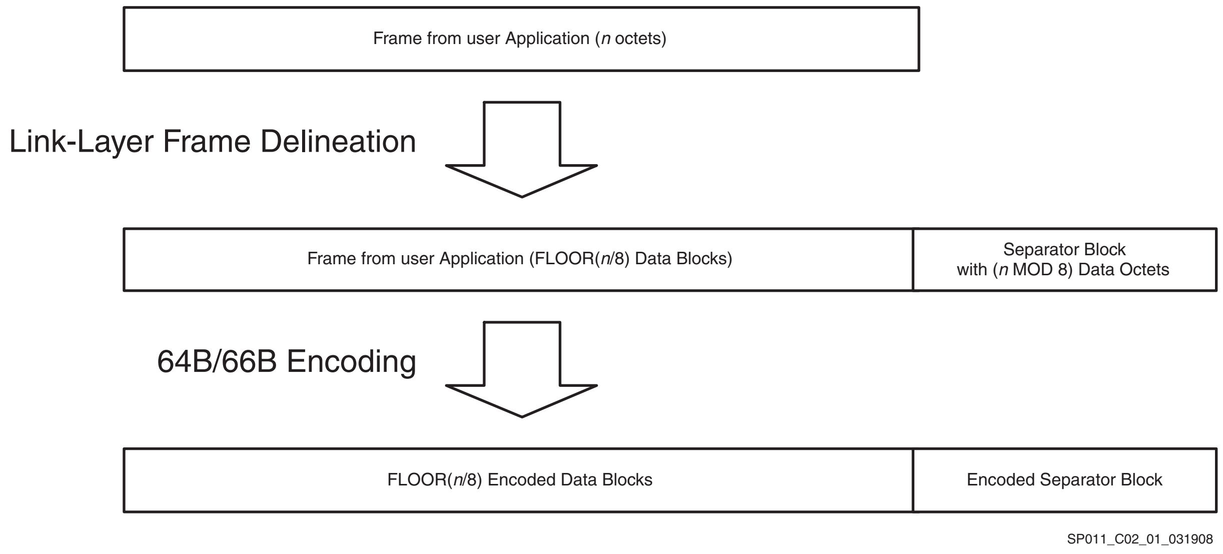

Frame from user Application (n octets)
Link-Layer Frame Delineation
Frame from user Application (FLOOR(n/8) Data Blocks)
Separator Block with (n MOD 8) Data Octets
64B/66B Encoding
FLOOR(n/8) Encoded Data Blocks
Encoded Separator Block
SP011_C02_01_031908

*Figure 2-1:* **Mapping Frames to Encoded Block Codes for Transmission**

## 2.3.1 Link-Layer Frame Delineation

The end of every frame from the user application is delineated in the Aurora channel using a Separator or Separator-7 block. This delineation allows the channel partner to distinguish data from different frames. The frame itself is cut into eight-octet chunks to fit into Data blocks. Any remaining octets at the end of the frame are transmitted in the data section of the Separator block. If the number of remaining octets is six or less, a Separator block is used. If seven octets remain, a Separator-7 block is used.

***Note:*** The Aurora interface can choose to send a higher priority block at any time during the frame. Idle blocks can also be intermingled with Data and Separator blocks at any time.

## 2.3.2 64B/66B Encoding

After the Frame has undergone link-layer frame delineation, the Data and Separator blocks comprising the frame are 64B/66B coded by the physical coding sub-layer (PCS) prior to transmission, transforming 64-bit blocks in to 66-bit block codes. The details of the coding process are described in Section 5.2 "Aurora Encoding."

## 2.3.3 Serialization and Clock Encoding

After the blocks comprising the frame have undergone 64B/66B encoding, they are serialized for transmission. The details of this process are described in Section 7.2 "Bit and Byte Ordering Convention." The serialized data stream is transmitted in differential non-return-to-zero (NRZ) format in accordance Section 8, "Electrical Specifications."

## 2.3.4 Multi-Lane Transmission

Multi-lane frame transmission is the same as single-lane transmission, except that the block codes created from frame data during the link-layer frame delineation step are scheduled for transmission using the striping rules defined in Section 6.4 "Frame Data Striping." These rules distribute block codes carrying frame data evenly across all the lanes comprising the channel.

## 2.4 Frame Reception Procedure

The Aurora interface performs the following procedure to receive a frame from an initialized Aurora channel and pass it to a user application:

- Data is deserialization
- 64B/66B decoding of all blocks is performed
- Control blocks are stripped from the data stream

Figure 2-2 illustrates how frames are received using this procedure.

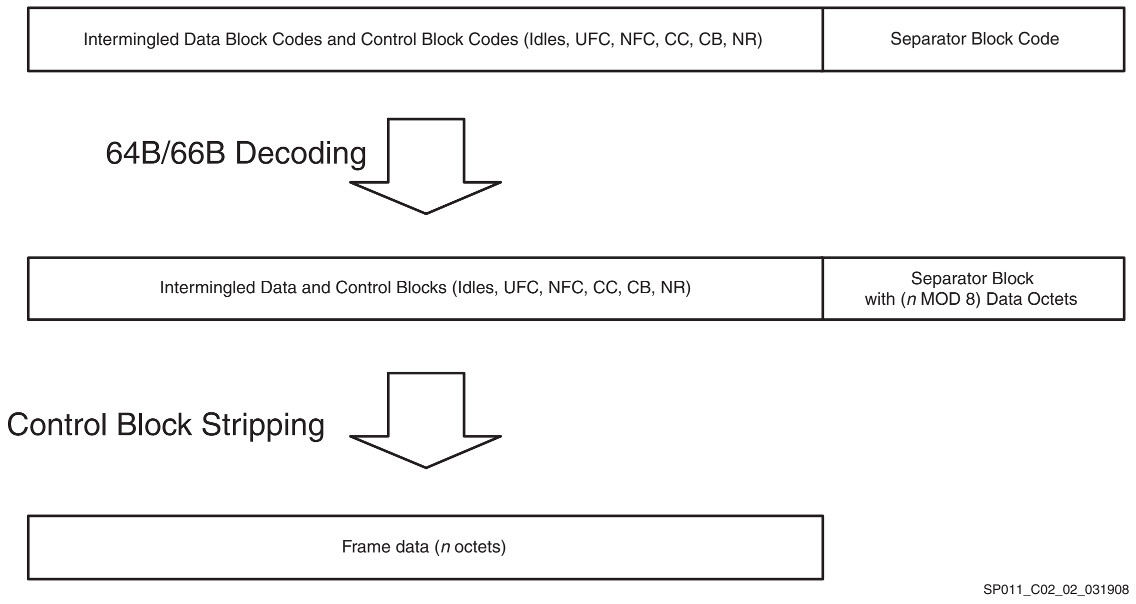

Intermingled Data Block Codes and Control Block Codes (Idles, UFC, NFC, CC, CB, NR)
Separator Block Code
64B/66B Decoding
Intermingled Data and Control Blocks (Idles, UFC, NFC, CC, CB, NR)
Separator Block with (n MOD 8) Data Octets
Control Block Stripping
Frame data (n octets)
SP011_C02_02_031908

*Figure 2-2:* **Receiving Data from an Aurora Channel**

### 2.4.1 Deserialization

The serial data stream is received in differential non-return-to-zero (NRZ) format in accordance with the electrical specifications described in Section 8, "Electrical Specifications." The receive logic deserializes the data to recover the 66-bit encoded block codes. The details of this process are described in Section 7.4 "Deserialization."

Block alignment within the stream is established during the lane initialization procedure described in Section 4.2.1 "Lane Initialization."

***Note:*** The lane initialization procedure must be carried out before the Aurora channel is used to transfer frame data.

### 2.4.2 64B/66B Decoding

After deserialization, the incoming encoded 66-bit block codes are decoded into 64-bit blocks. The details of the 64B/66B decoding process are described in Section 5.2 "Aurora Encoding."

### 2.4.3 Control Block Stripping

Data and Separator blocks from the Aurora channel can be intermingled with control blocks, such as Clock Compensation, Channel Bonding, flow control, User K-Blocks and idles. All of these control blocks are separated from the frame data and processed separately. Only the data octets from the Data blocks and Separator block are delivered to the user interface as a frame.

User flow control messages also use Data blocks to transfer message data. When part of a Data block is used for user flow control message octets, none of the octets in that block are used to carry data octets. User flow control is discussed in more detail in Section 3, "Flow Control."

Not Ready blocks reset frames. Frames in progress are considered lost if a Not Ready block is received. Blocks received after a Not Ready block are considered to be part of a new frame.

### 2.4.4 Multi-Lane Reception

Multi-lane frame reception is the same as single-lane reception, except that the block codes are received from multiple lanes. The block codes are ordered according to the striping rules defined in Section 6.4 "Frame Data Striping" before the control block stripping step.

## 2.5 Data and Separator Block Format

Figure 2-3 shows the format of a Data block used to carry frame data. Data blocks must be filled starting from the least-significant octet, and data in the block must be contiguous. Data blocks used for frame data must be filled completely.

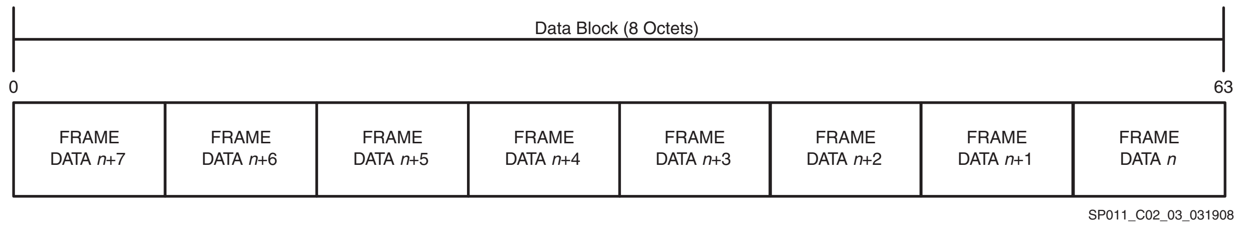


*Figure 2-3:* **Data Block Used for Frame Data**

Figure 2-4 shows the format of a Separator block. The two most significant octets of the block are used for control information; the remaining blocks can be used for frame data. Frame data is added to the block starting from the least-significant octet, and the data in the block must be contiguous. Separator blocks can be partially filled. Unused octets are considered don't-care octets in the current frame. The 8-bit valid octet count field indicates how many of the remaining octets in the Separator block carry frame data. The block can carry 0 to 6 frame data octets.

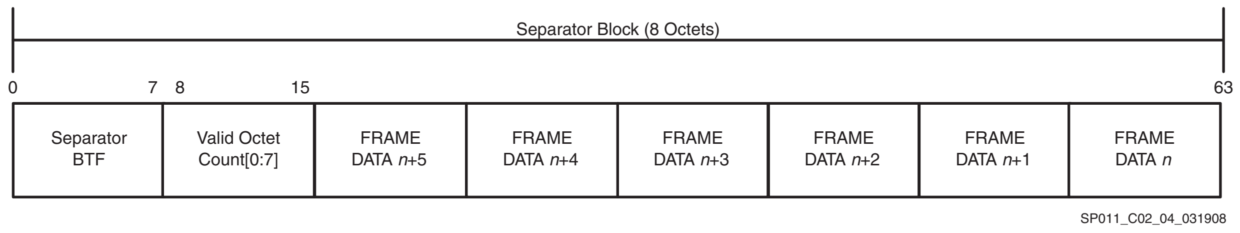


*Figure 2-4:* **Separator Block**

Figure 2-5 shows the format of a Separator-7 block. The most significant octet of the block is used for control information; the remaining blocks are used for frame data. Frame data is added to the block starting from the least-significant octet, and the data in the block must be contiguous. All seven data octets of the Separator-7 block must be filled with frame data.

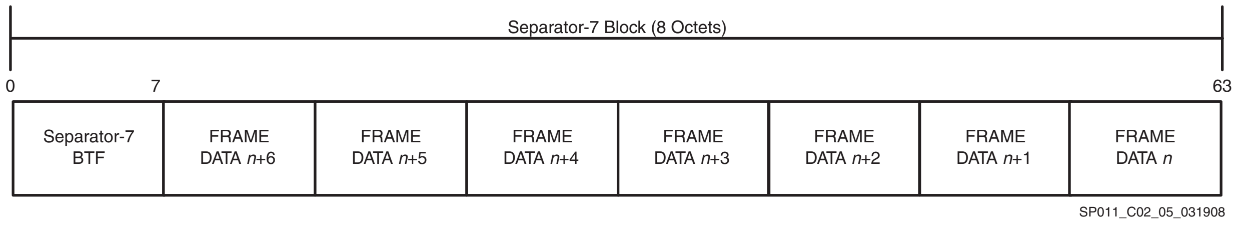


*Figure 2-5:* **Separator-7 Block**

Figure 2-6 and Figure 2-7 show examples of frame data transfer through single-lane and multi-lane Aurora channels respectively. Each frame transferred carries a series of incrementing octets.

***Note:*** Frames can be interrupted by idles, and can be back-to-back. Separator blocks by themselves can be used to end frames divisible by eight, or carry all the data for frames of seven octets or smaller.

| Lane 0 |
|---|
| IDLE |
| 07,06,05,04,03,02,01,00 |
| IDLE |
| SEP,2,--,--,--,--,09,08 |
| IDLE |
| 17,16,15,14,13,12,11,10 |
| 1F,1E,1D,1C,1B,1A,19,18 |
| SEP,0,--,--,--,--,--,-- |
| SEP-7,25,24,23,22,21,20 |
| SEP,1,--,--,--,--,--,30 |

SP011_C02_06_021408

*Figure 2-6:* **Example of Frame Data Transfer through a Single-Lane Channel**

| Lane 0 | Lane 1 |
|---|---|
| IDLE | IDLE |
| 07,06,05,04,03,02,01,00 | SEP,2,--,--,--,--,09,08 |
| IDLE | IDLE |
| 17,16,15,14,13,12,11,10 | 1F,1E,1D,1C,1B,1A,19,18 |
| SEP,0,--,--,--,--,--,-- | SEP-7,25,24,23,22,21,20 |
| SEP,1,--,--,--,--,--,30 | IDLE |

SP011_C02_07_021408

*Figure 2-7:* **Example of Frame Data Transfer through a Multi-Lane Channel**

Section 3

# Flow Control

## 3.1 Overview

This chapter describes the optional flow control features in the Aurora 64B/66B protocol.

Aurora supports the following flow control mechanisms:

- **Native flow control:** a link-layer flow control mechanism, allowing receivers to request that their channel partner transmit idles instead of data. These requests are executed by the remote Aurora interface.
- **User flow control**: a mechanism allowing short, high-priority control messages to be sent through the Aurora channel. These messages can be used to implement higher level flow control mechanisms and application-specific control schemes.

Aurora 64B/66B flow control is very similar to Aurora 8B/10B flow control with four key differences:

- Flow control uses block codes and 64B/66B encoding instead of 8B/10B symbols and 8B/10B encoding
- User flow control messages can be 1 to 256 octets long
- User flow control messages can be interrupted by control characters at block boundaries
- Native flow control PAUSE intervals are not encoded

## 3.2 Native Flow Control Operation

To start the native flow control (NFC) mechanism, the Aurora interface must send a native flow control request message to its channel partner. This message carries a PAUSE value telling the channel partner to make idles a higher priority than data until the number of non-data blocks (excluding Clock Compensation and Not Ready) requested by PAUSE are transmitted on each of its lanes. The time when the transmitter is processing the native flow control request is called the native flow control countdown.

NFC messages are not cumulative: if a new NFC message arrives while an Aurora interface is still processing a previous NFC request, the new PAUSE value immediately replaces the old.

There are two modes of operation for native flow control: immediate, and completion mode. In immediate mode, Aurora interfaces must process any NFC request they receive immediately. In completion mode, Aurora interfaces must wait until they are at the beginning of a new frame (in a frame where no frame data has yet been transmitted) before processing the NFC request.

PAUSE values are 8-bit values and used to carry a binary value indicating the length of the NFC countdown in cycles. There is also an XOFF bit used to request XOFF mode. In XOFF mode, idles have a higher priority than data until the Aurora interface receives an NFC request for a non-XOFF PAUSE, or transmits or receives a Not Ready block.

When a request is transmitted with PAUSE and XOFF both set to 0, NFC is set to XON mode. When XON is received, any NFC countdowns in progress are halted, and idles immediately become the lowest priority block.

## 3.3 Native Flow Control Latency

In immediate mode, the time between NFC request message transmission and the first Idle block sent due to that request must be no more than the time it takes the originator of the request to send 256 blocks on a single lane.

## 3.4 Native Flow Control Block Format

NFC requests transmitted using Native Flow Control blocks (Figure 3-1 shows the format of an NFC block). To send a request, the Aurora interface transmits the NFC block with the XOFF bit and PAUSE fields set as desired.

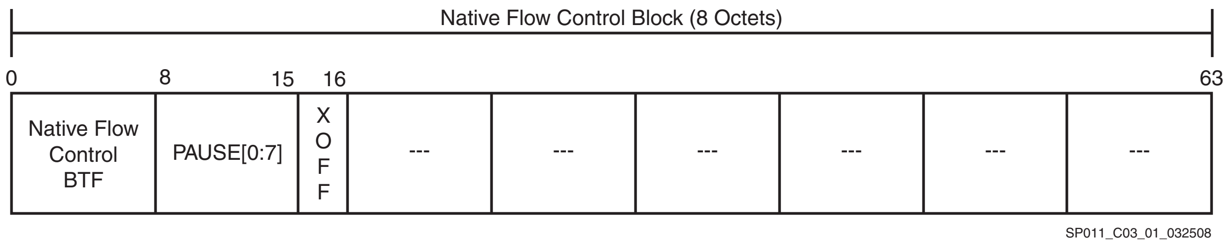


*Figure 3-1:* **NFC Block**

## 3.5 User Flow Control Operation

User flow control (UFC) messages are made up of a User Flow Control header block and a set of Data blocks, and can be sent at any time, as long as the channel is initialized, and no higher priority block is being transmitted.

UFC messages are made up of a header and message data. The UFC header indicates how many octets of data are in the message. The message can be any length from 1 to 256 octets. Message octets are transmitted using Data blocks. If the last Data block is not completely filled, the unused octets are treated as don't-care — they cannot be used to carry regular frame data.

While a UFC message is being transmitted, Data, Separator, and Separator-7 blocks for frame data drop to the lowest priority (below idles). This drop in priority prevents regular frame data from being mixed with UFC message data. The time when a UFC message is being transmitted is called the user flow control countdown. The countdown is considered complete for a given lane if there are no more UFC message octets to send.

If the channel is reset or the Aurora interface receives a Not Ready block in the middle of a UFC message, the message is considered dropped — the Aurora interface need not wait for the remaining octets.

## 3.6 User Flow Control Message Format

Figure 3-2 shows the format of an UFC Block. This block is used to send the header of a UFC message. Figure 3-3 shows the format of a Data block carrying UFC message data. Octets in the Data block must be contiguous, but there can be any number of block codes between the blocks that make up the UFC message, provided they are not blocks carrying frame data, or Not Ready blocks.

The UFC_COUNT is the 8-bit field indicating the total length of the UFC message in octets. The length of the message is UFC_COUNT + 1 octets.

UFC message data is sent using Data blocks. The blocks must be filled starting with the least-significant octet, with all data octets contiguous within the block. Any unused message data octets are don't-care, and cannot be used for regular frame data.

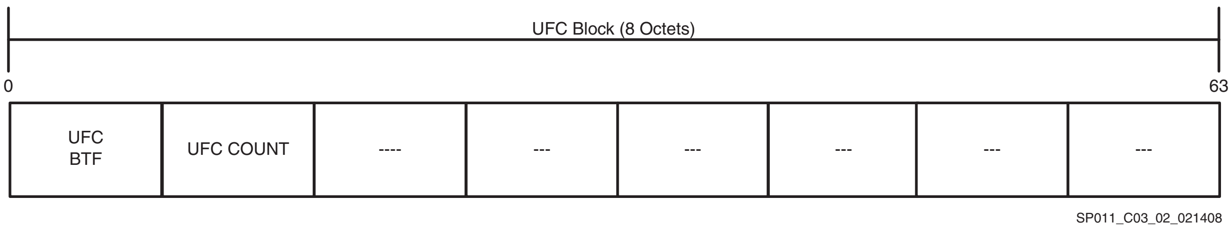


*Figure 3-2:* **UFC Block with UFC Header**

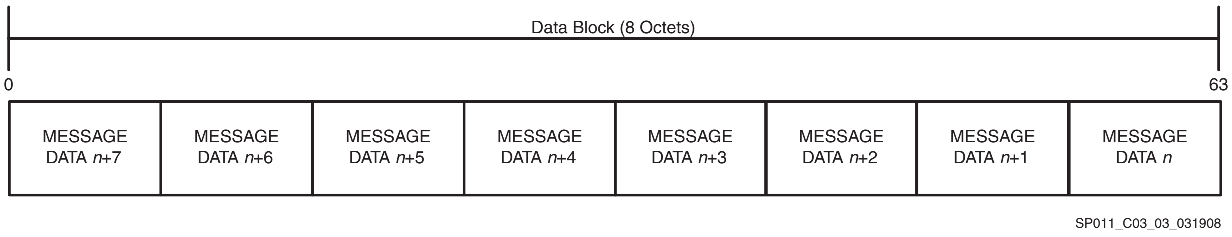


*Figure 3-3:* **Data Block Used to Carry UFC Message Data**

Figure 3-4 and Figure 3-5, page 32 show examples of legal UFC messages being transferred through single-lane and multi-lane Aurora channels respectively. UFC header blocks are labeled with their block type and the value of their message length field in decimal. The messages carry incrementing hex octets.

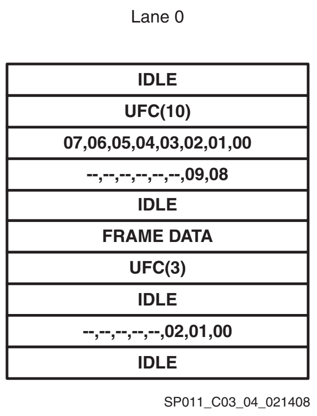


*Figure 3-4:* **Example UFC Messages for Single-Lane Channel**

| Lane 0 | Lane 1 |
|---|---|
| IDLE | UFC(10) |
| 07,06,05,04,03,02,01,00 | --,--,--,--,--,--,09,08 |
| IDLE | FRAME DATA |
| UFC(3) | IDLE |
| --,--,--,--,--,02,01,00 | IDLE |

SP011_C03_05_021408

*Figure 3-5:* **Example UFC Messages for a Multi-Lane Channel**

# Initialization and Error Handling

## 4.1 Overview

This chapter describes the procedure used to prepare an Aurora channel for operation. Initialization of an Aurora 64B/66B channel is a two-stage process consisting of the following stages:

- **Lane initialization:** During this stage, each serial lane in the channel is individually reset and aligned to the block boundaries of incoming data.
- **Channel bonding:** During this stage, the Aurora interface uses channel bonding to compensate for the skew between each of the individual lanes. After successful channel bonding, the lanes are treated as a single communication channel.

Full-duplex channels include an additional state to ensure that both Aurora interfaces connected to the channel are ready to receive data before frame transmission commences.

Aurora 64B/66B initialization differs from Aurora 8B/10B initialization in several key ways:

- The number of initialization messages and handshakes is reduced
- There is no channel verification stage
- The 64B/66B block lock state machine defined in IEEE 803.3ae [Ref 1] for 10-Gigabit Ethernet is now used for block alignment and error handling
- The Aurora protocol recommends a specific CRC polynomial for compatibility with the 64B/66B scrambling polynomial

Figure 4-1, page 34 illustrates the initialization procedure for Aurora 64B/66B, including the additional state used for full-duplex initialization. Simplex channels do not include the Wait For Remote state (in Figure 4-1, this state is treated as an edge allowing simplex Aurora interfaces to immediately leave the state).

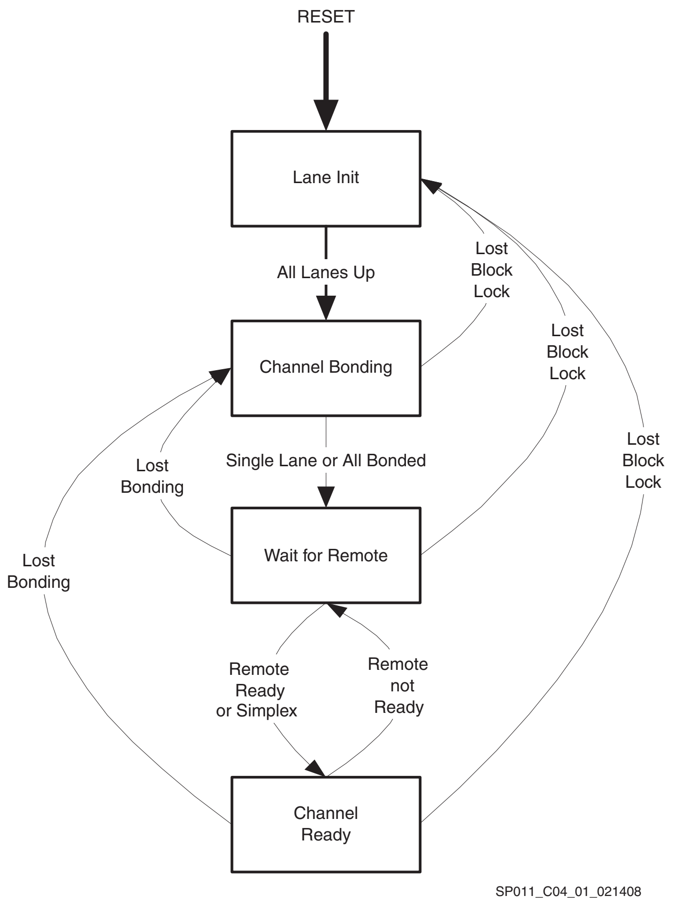


*Figure 4-1:* **Initialization Overview**

## 4.2 Aurora Channel Initialization

### 4.2.1 Lane Initialization

Aurora channels are made up of one or more individual serial links (lanes). During lane initialization, each lane prepares itself to transmit and receive data, then transmits blocks while attempting to find the block boundaries of the incoming data and align to them.

To align to incoming blocks, the Aurora interface uses the block sync state machine defined in IEEE 802.3ae [Ref 2] (see IEEE 802.3ae (2002), figure 49-12). The Aurora interface stays in the lane initialization state until the block sync state machine reaches the 64_GOOD state and asserts block_lock (see Figure 4-2, page 35).

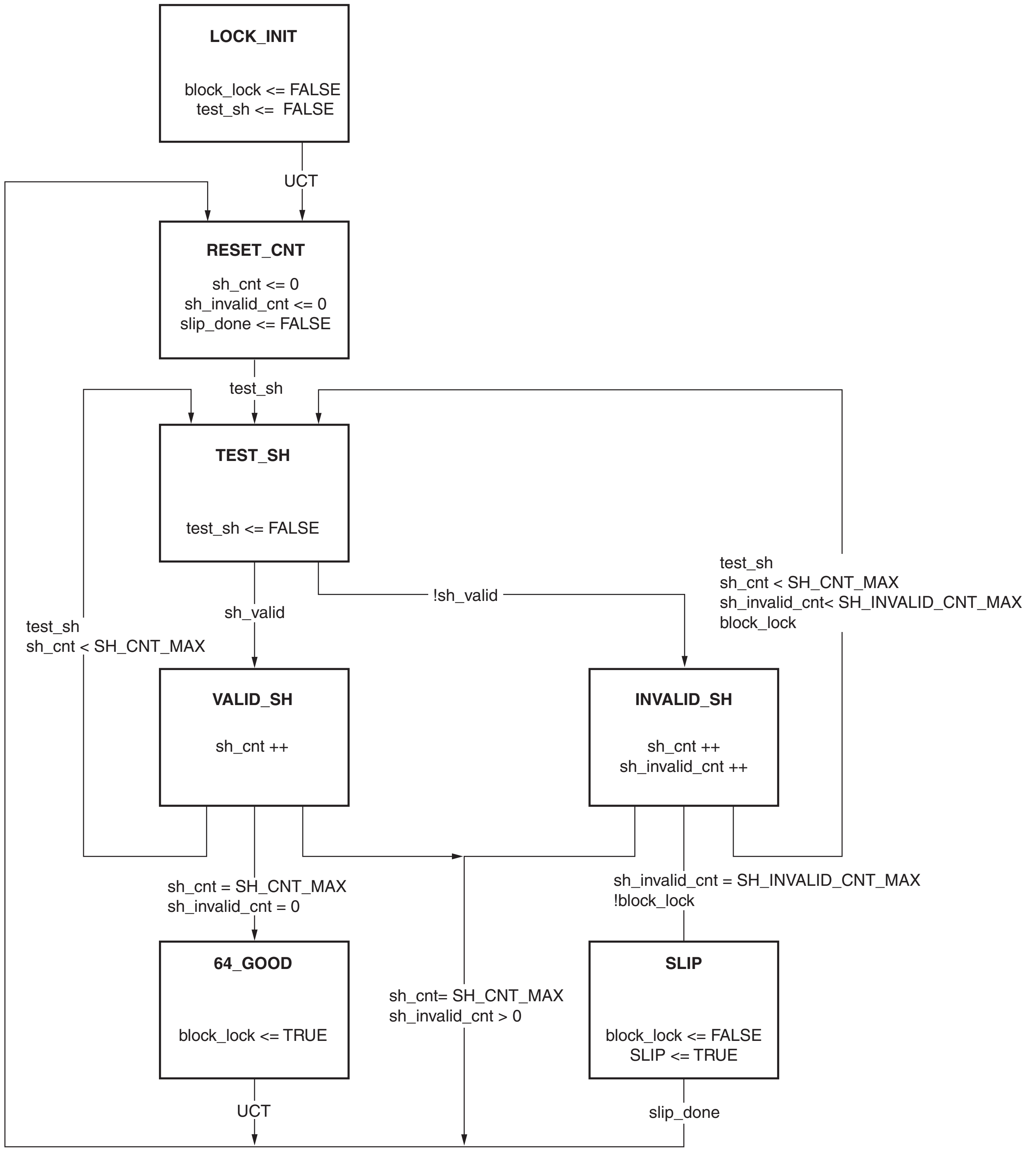


*Figure 4-2:* **Block Sync State Machine**

If at any time block_lock is deasserted, the lock is considered lost, and all lanes in the channel must return to the lane initialization state.

Aurora channels can be full-duplex or simplex. Similarly, they can also be single- or multi-lane. Table 4-1 shows what each lane should be transmitting during lane initialization for each possible configuration.

Table 4-1: **Required Block Transmission during Lane Initialization**

| | Full-Duplex | Simplex |
|---|---|---|
| Single-Lane | Not Ready block. | Idle block. |
| Multi-Lane | Not Ready block and Channel Bonding block. There must be at least four Not Ready blocks between each Channel Bonding block. | Idle block and Channel Bonding block. There must be at least four Idle blocks between each Channel Bonding block. |

Aurora interfaces should leave the lane initialization state only after block_lock is asserted for all lanes in the channel. Table 4-2 shows the next state for each type of Aurora interface after lane initialization is complete.

Table 4-2: **Required State Transition after Lane Initialization**

| | Full-Duplex | Simplex |
|---|---|---|
| Single-Lane | Wait For Remote | Channel Ready |
| Multi-Lane | Channel Bonding | Channel Bonding |

Full-duplex differential receivers can adjust their polarity during lane initialization. If after block lock is achieved, the incoming block codes are Data instead of control blocks, the lane may reverse its RX polarity and reset block lock. Resetting block lock gives the remote Aurora interface time to send control blocks, ensuring that the polarity in not defined incorrectly due to actual Data blocks in the channel.

### 4.2.2 Channel Bonding

Multi-lane channels cannot start sending data until the skew between the lanes is eliminated. In the Aurora 64B/66B protocol, deskew is accomplished during the channel bonding stage.

While channel bonding, all lanes in a full-duplex channel must transmit a mix of Not Ready and Channel Bonding blocks (all lanes must transmit the Channel Bonding blocks simultaneously). The minimum number of Not Ready blocks between each Channel Bonding block is four. Simplex channels must transmit regular Idle blocks in the place of the Not Ready blocks.

When the lanes in the Aurora interface receive Channel Bonding blocks, they must each individually adjust their PCS latency so that the Channel Bonding blocks are all available at the RX data interface of the PCS layer simultaneously. When the Aurora channel control logic sees that all lanes in the channel are simultaneously delivering Channel Bonding blocks, channel bonding is complete.

For a simplex Aurora interface, Channel Bonding blocks should be transmitted periodically on all lanes simultaneously during regular operation, provided there is not a higher priority block awaiting transmission.

If at any time, non-simultaneous Channel Bonding blocks are received, the receiving Aurora interface must return to the channel bonding state.

Aurora interfaces should leave the channel bonding state after channel bonding succeeds. Table 4-3, page 37 shows the next state for each type of Aurora interface.

*Table 4-3:* **Required State Transition after Successful Channel Bonding**

| | Full-Duplex | Simplex |
|---|---|---|
| Single-Lane | Wait For Remote | Channel Ready |
| Multi-Lane | Wait For Remote | Channel Ready |

### 4.2.3 Wait For Remote

Full-duplex Aurora interfaces go to the Wait For Remote state when their channel partner appears unready to receive data. Channel partners are considered to be unready to receive data if they are transmitting Not Ready blocks.

While in the Wait For Remote state, Aurora interfaces must transmit a mix of Idle and Channel Bonding blocks. The Channel Bonding blocks must be separated by at least four Idle blocks.

Aurora interfaces may leave the Wait For Remote state when they receive any block not a Channel Bonding, Clock Compensation, or Not Ready. Aurora interfaces must be ready to receive data while in the Wait For Remote state, but may not transmit data.

### 4.2.4 Channel Ready

In the channel ready state, full-duplex Aurora interfaces may send and receive data and flow control freely. Simplex interfaces may send or received data freely, depending on their type.

Full-duplex modules must return to the lane initialization state if block_lock (from the IEEE 802.3ae [Ref 2] block sync state machine) is deasserted for any lane, or if the Aurora interface is reset for any reason. If they have multiple lanes, the modules must return to the channel bonding state if they ever receive a Channel Bonding block on one lane, but not all the others. If the modules receive a Not Ready block, they must return to the Wait For Remote state. Full-duplex modules should transmit a minimum of 64 idle characters and receive a minimum of 16 idle characters to transition to Channel Ready state.

Simplex TX Aurora interfaces return to the lane initialization state if they are reset for any reason. Simplex RX Aurora interfaces return to the lane initialization state if they are reset for any reason, or if block_lock is deasserted for any lane. If composed of multiple lanes, the interfaces must return to the channel bonding state if they ever receive a Channel Bonding block on one lane, but not all the others. Simplex TX Aurora interfaces should transmit a minimum of 64 idle characters and Simplex RX Aurora interfaces should receive a minimum of 16 idle characters to transition to Channel Ready state.

### 4.2.5 Simplex Transmitters

Because simplex transmitters do not know when the channel is initialized, they do not use the initialization states shown in Figure 4-1, page 34. Instead, simplex transmitters transmit continuously, and it is left to the user application to use outside information to determine the best time to start sending real data. Multi-lane simplex transceivers should always transmit Channel Bonding blocks periodically, so that channel bonding can be performed any time the channel is reset.

## 4.3 Error Handling

The Aurora 64B/66B protocol defines two types of errors: hard and soft. Hard errors are catastrophic or irrecoverable errors, such as channel disconnection, buffer overflow, or hardware failure. Soft errors are transient, statistical errors, expected in the normal course of operation.

Aurora interfaces respond to hard errors by resetting. Any hardware failure condition detected by the Aurora interface should be treated as a hard error.

Soft errors do not cause a reset, but can be reported to the user application. Illegal sync header values (`11` or `00`), or illegal block type field (BTF) values in a control block should be reported as soft errors.

***Note:*** The block sync state machine used for lane initialization is operational at all times. Since a large number of soft errors can indicate loss of lock, the state machine deasserts block_lock and forces the Aurora interface back to lane initialization if too many soft errors occur in a short period of time.

## 4.4 CRC

Because data is scrambled before transmission, if CRC is used for error detection in the channel, it is recommended that the following standard polynomial (Equation 4-1) be used to calculate a 32-bit CRC for outgoing data:

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$$

*Equation 4-1*

# PCS Layer

## 5.1 Overview

This section specifies the functions performed in the PCS layer of the Aurora 64B/66B protocol:

- Section 5.2 "Aurora Encoding" specifies how data and control information must be encoded before transmission through an Aurora channel, and decoded upon reception.
- Section 5.3 "64B/66B Scrambling" specifies how data and control information must be scrambled before transmission and de-scrambled after reception.
- Section 5.4 "64B/66B Gearbox" specifies how 64-bit blocks must be transformed to 66-bit blocks for transmission, and how 66-bit blocks must be transformed to 64-bit blocks upon reception.
- Section 5.5 "Channel Bonding" (discussed in detail in Section 4.2.2 "Channel Bonding")
- Section 5.6 "Clock Compensation" specifies the mechanism used in Aurora 64B/66B to compensate for small differences between the reference clocks on each side of the Aurora channel.

## 5.2 Aurora Encoding

All data and control information in Aurora 64B/66B is encoded in 64-bit blocks. Each 64-bit block is marked with a sync header value, indicating whether or not it is a Data, or control block. The sync header is combined with the block to form a block code. There are 15 types of control blocks available in Aurora 64B/66B. Of these 15, 5 are permanently assigned specific functions in the Aurora 64B/66B protocol, and 9 can be assigned any meaning by a higher level application (User K-Blocks); the one remaining control block is reserved.

### 5.2.1 Block Codes in 64B/66B

All block codes in 64B/66B have eight octets of data or control information preceded by a 2-bit sync header. Data block codes have a `01` sync header. Control block codes have a `10` sync header. Sync header values `11` and `00` are illegal and should be flagged as soft errors whenever they appear (see Section 4.3 "Error Handling").

Sync header bits are never scrambled. The remaining eight octets, however, are always scrambled before transmission.

Aurora 64B/66B Data blocks can carry frame data or user flow control message data. Data blocks are always filled starting at the least-significant octet (the rightmost octet, furthest from the sync header). When sending regular data, all eight octets must be used. When carrying UFC message data, Data blocks may be left partially filled if the UFC message finishes before the end of a block (the remaining octets are don't-care).

The most-significant byte of every control block is always used to carry the block type field (BTF) identifying the type of control block being transmitted. There are 14 legal BTF values in Aurora 64B/66B and one reserved value (Table 5-1). The control blocks corresponding to these BTFs are described in more detail in the sections that follow.

*Table 5-1:* **Valid Block Type Field Values in Aurora 64B/66B**

| Control Block Name | Block Type Field (BTF) Value: Block Code[2:9] |
|---|---|
| Idle/NotReady/Clock Compensation/ Channel Bonding | `0x78` |
| Native Flow Control | `0xaa` |
| User Flow Control | `0x2d` |
| Separator | `0x1e` |
| Separator-7 | `0xe1` |
| User K-Block 0 | `0xd2` |
| User K-Block 1 | `0x99` |
| User K-Block 2 | `0x55` |
| User K-Block 3 | `0xb4` |
| User K-Block 4 | `0xcc` |
| User K-Block 5 | `0x66` |
| User K-Block 6 | `0x33` |
| User K-Block 7 | `0x4b` |
| User K-Block 8 | `0x87` |
| Reserved | `0xff` |

## 5.2.2 Idle/Not Ready/Clock Compensation/Channel Bonding Block Code

Figure 5-1, page 41 shows the format of the Idle/Not Ready/Clock Compensation/Channel Bonding block codes.

### 5.2.2.1 Clock Compensation Idles

Bit 10 of the Idle block code is the clock compensation bit (CC). When this bit is High, the block code is a Clock Compensation idle. When transmitting an Idle block with bit 10 set High, bits 11, 12, and 13 must be Low. In multi-lane channels, Clock Compensation idles must be transmitted simultaneously on all lanes.

The function of Clock Compensation idles is described in more detail in Section 5.6 "Clock Compensation."

#### 5.2.2.2 Channel Bonding Idles

Bit 11 of the Idle block code is the channel bonding bit (CB). When this bit is High, the block code is a Channel Bonding idle. When transmitting an Idle block code with bit 11 set High, transmitters must set bits 10, 12, and 13 must be Low. In multi-lane channels, Channel Bonding idles must be transmitted simultaneously on all lanes

The function of Channel Bonding idles is described in more detail in Section 4.2.2 "Channel Bonding" and Section 5.5 "Channel Bonding."

#### 5.2.2.3 Not Ready Idles

Bit 12 of the Idle block code is the not ready bit (NR). When this bit is High, the block code is a Not Ready idle. Not Ready idles are used for full-duplex initialization, as described in Section 4.2.3 "Wait For Remote." Simplex Aurora interfaces do not use Not Ready idles — simplex Aurora transmitters must always set bit 12 low, and simplex receivers should act as if bit 12 is low on all incoming Idle block codes.

When transmitting an Idle block code with bit 12 set High, transmitters must set bits 10 and 11 Low. In multi-lane channels, if bit 12 is High for any idles transmitted on a given cycle, all Idle block codes transmitted on that cycle must have bit 12 set High.

#### 5.2.2.4 Regular Idles

When bits 10, 11 and 12 of the Idle block code are Low, the block code is considered to be a regular idle, with no special properties.

#### 5.2.2.5 The Strict Alignment Bit

Bit 13 of the Idle block code is the strict alignment (SA) bit. Full-duplex modules with receivers that obey the rules for strict alignment described in Section 6.5 "Strict-Alignment Frame Data Striping" and Section 6.7 "Strict-Alignment User Flow Control Striping" must set bit 13 High whenever transmitting a regular idle or a Not Ready idle (Figure 5-1).

If an Idle block code is received with bit 13 set High, this indicates the receiver from the remote Aurora interface only accepts strictly aligned data.

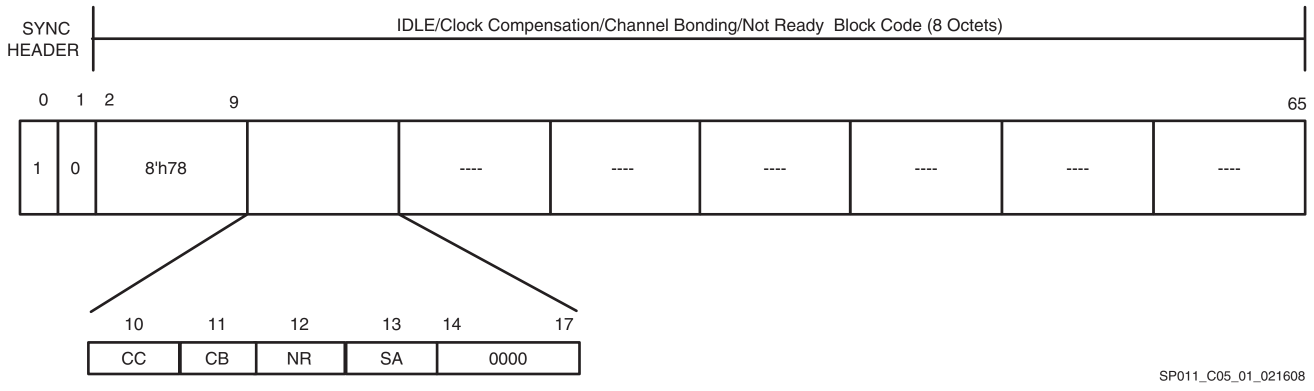


*Figure 5-1:* **Idle/Not Ready/NFC Block Code**

### 5.2.3 Native Flow Control Block Code

Figure 5-2 shows the format of the Native Flow Control block code. This block code is used to send NFC messages to perform the functions described in Section 3.2 "Native Flow Control Operation."

The second octet of the block code (bits 10 to 17) is used to carry the PAUSE field for the NFC message. PAUSE is used to request a specific number of idle cycles. Bit 18 is the XOFF bit. This bit is set High to make an NFC XOFF request. When this bit is High, the PAUSE field can be treated as don't-care.

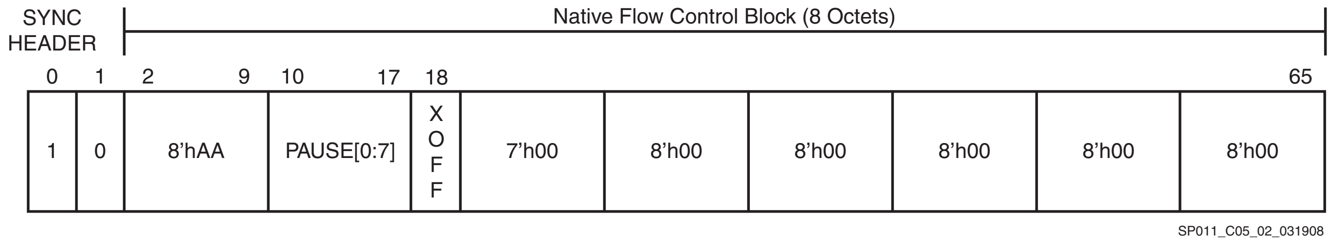


*Figure 5-2:* **Native Flow Control Block Code**

### 5.2.4 Data Block Code for Frame Data

Figure 5-3 shows the format of the Data block code when it is used to carry frame data. See Section 2, "Data Transmission and Reception" for details about how the Data block code is used to carry frame data.

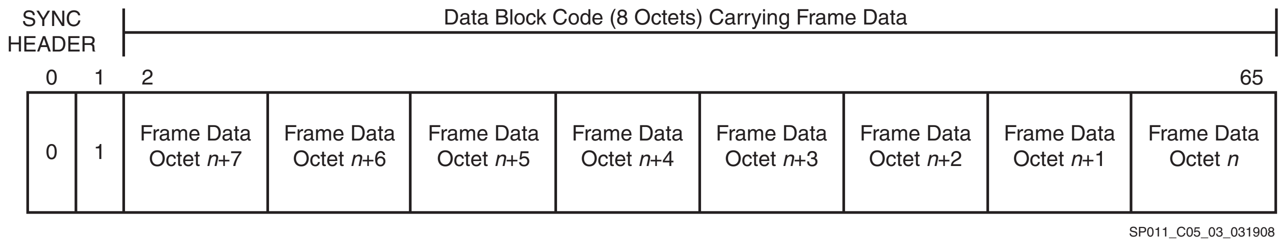


*Figure 5-3:* **Data Block Code Carrying Frame Data**

### 5.2.5 Separator Block Code

Figure 5-4 shows the format of the Separator block code. See Section 2, "Data Transmission and Reception" for details about how the Separator block code is used to delineate frames and carry frame data.

Valid octet count is an 8-bit field indicating the number of frame data octets in the Separator block. Table 5-2, page 43 lists valid octet count values and their description.

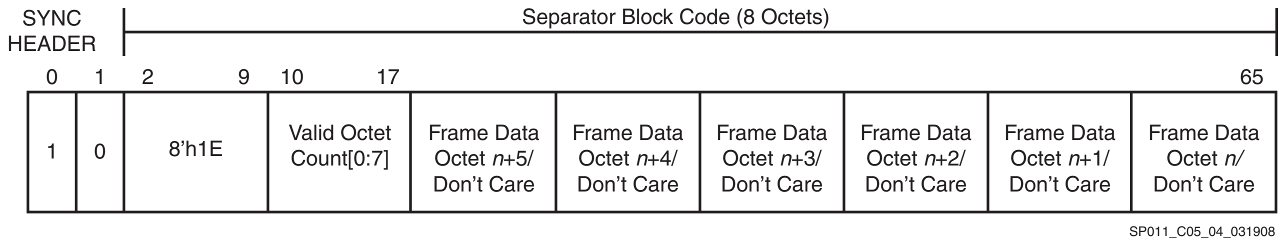

SYNC HEADER
Separator Block Code (8 Octets)
0
1
2
9
10
17
65
1
0
8'h1E
Valid Octet Count[0:7]
Frame Data Octet n+5/ Don't Care
Frame Data Octet n+4/ Don't Care
Frame Data Octet n+3/ Don't Care
Frame Data Octet n+2/ Don't Care
Frame Data Octet n+1/ Don't Care
Frame Data Octet n/ Don't Care
SP011_C05_04_031908

*Figure 5-4:* **Separator Block Code**

*Table 5-2:* **Valid Octet Count Field Values for Separator Block Code**

| Valid_Octet_Count[0:7] (SEP Block Code [10:17]) | Description |
|---|---|
| xxxxx000 | No Frame Data Octets are valid |
| xxxxx001 | Frame Data Octet *n* is valid |
| xxxxx010 | Frame Data Octets *n* and *n*+1 are valid |
| xxxxx011 | Frame Data Octets *n* to *n*+2 are valid |
| xxxxx100 | Frame Data Octets *n* to *n*+3 are valid |
| xxxxx101 | Frame Data Octets *n* to *n*+4 are valid |
| xxxxx110 | Frame Data Octets *n* to *n*+5 are valid |
| xxxxx111 | Invalid Code |

## 5.2.6 Separator-7 Block Code

Figure 5-5 shows the format of the Separator-7 block code. See Section 2, "Data Transmission and Reception" for details about how the Separator-7 block code is used to delineate frames and carry frame data.

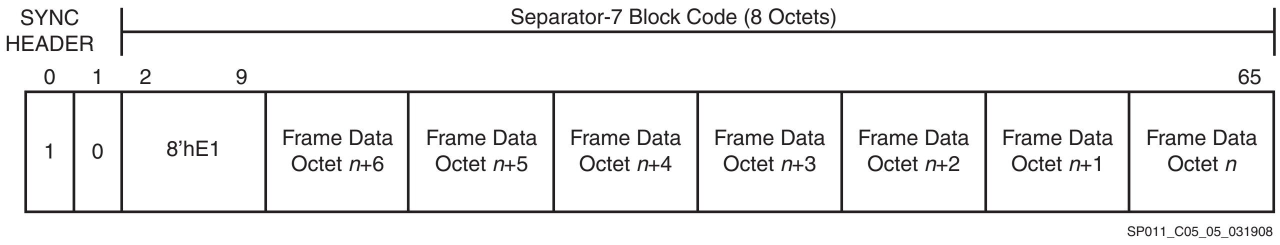


*Figure 5-5:* **Separator-7 Block Code**

## 5.2.7 User Flow Control Block Code

Figure 5-6 shows the format of the User Flow Control block code. See Section 3.6 "User Flow Control Message Format" for details about how the User Flow Control block code is used to start user flow control messages and carry user flow control message data.

UFC Count is an 8-bit field which can be set to any 8-bit value. Total number of octets in UFC message = UFC_Count +1.

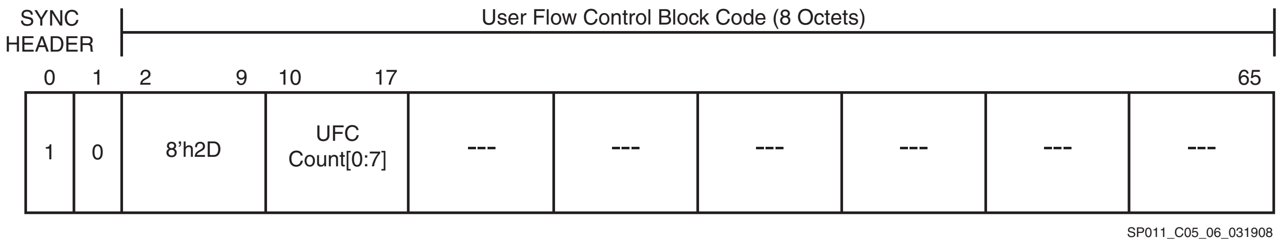


*Figure 5-6:* **User Flow Control Block Code**

## 5.2.8 Data Block Code for User Flow Control Message

Figure 5-7 shows the format of the Data block code when used to carry user flow control message octets. The Data block can be partially filled. All valid message octets must be placed contiguously in the block starting at the least-significant octet position.

If at the end of the user flow control message there are not enough octets to fill the entire block, the remaining spaces in the block are treated as don't-care. There remaining spaces may not be used for Frame data, or UFC message data from a different message

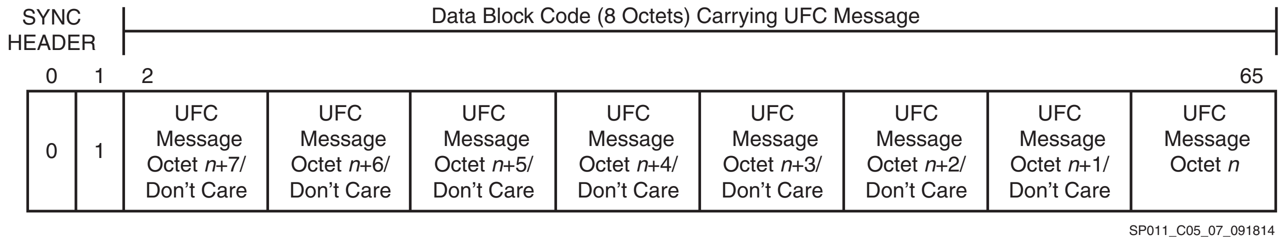

SYNC HEADER
Data Block Code (8 Octets) Carrying UFC Message
0
1
2
65
0
1
UFC Message Octet n+7/ Don't Care
UFC Message Octet n+6/ Don't Care
UFC Message Octet n+5/ Don't Care
UFC Message Octet n+4/ Don't Care
UFC Message Octet n+3/ Don't Care
UFC Message Octet n+2/ Don't Care
UFC Message Octet n+1/ Don't Care
UFC Message Octet n
SP011_C05_07_091814

*Figure 5-7:* **Data Block Code Carrying User Flow Control Message Data**

## 5.2.9 User K-Block Codes

There are nine unassigned BTF values in Aurora 64B/66B for user applications. When an Aurora interface receives one of these blocks, it passes the BTF and the seven remaining octets in the block directly to the user application as a User K-Block. Figure 5-8 shows the format of User K-Blocks. Table 5-3 shows the BTFs that may be used for User K-Blocks, and the name associated with each BTF in the Aurora 64B/66B protocol.

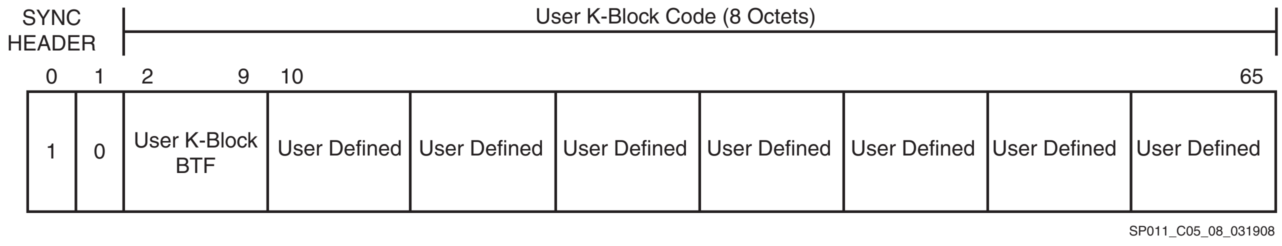


*Figure 5-8:* **User K-Block Code Format**

*Table 5-3:* **Valid Block Type Field Values for User K-Blocks**

| User K-Block Name | User K-Block BTF (User K-block[2:9]) |
|---|---|
| User K-Block 0 | `0xD2` |
| User K-Block 1 | `0x99` |
| User K-Block 2 | `0x55` |
| User K-Block 3 | `0xB4` |
| User K-Block 4 | `0xCC` |
| User K-Block 5 | `0x66` |
| User K-Block 6 | `0x33` |
| User K-Block 7 | `0x4B` |
| User K-Block 8 | `0x87` |

## 5.3 64B/66B Scrambling

Whenever a lane transmits a block code, the 8 octets following 2-bit sync header must be scrambled using a self-synchronizing scrambler with the polynomial:

$$G(x) = 1 + x^{39} + x^{58} \qquad \text{Equation 5-1}$$

Equation 5-1 is the scrambling algorithm used in IEEE 802.3ae [Ref 2]. The 2-bit sync header must never be scrambled.

When a block code is received, the 8 octets following the 2-bit sync header must be de-scrambled using a self-synchronizing de-scrambler with the same polynomial in Equation 5-1. This is the same de-scrambling algorithm as is used in IEEE 802.3ae [Ref 2]. The 2-bit sync header is never descrambled.

## 5.4 64B/66B Gearbox

When transmitting block codes, the 64B/66B block sync state machine combines the sync header with a 64-bit block from the scrambler to present a 66-bit block code to the PMA layer. This is the same algorithm as is used in IEEE 802.3ae [Ref 2].

When receiving block codes, the 64B/66B Gearbox separates the 66-bit block from the PMA into a 2-bit sync header and a 64-bit block for de-scrambling. This is the same algorithm as is used in IEEE 802.3ae [Ref 2].

***Note:*** The nomenclature for MSB and LSB in Aurora is the opposite of 802.3: See Section 7.2 "Bit and Byte Ordering Convention" for details.

## 5.5 Channel Bonding

Channel bonding is described in detail in Section 4, "Initialization and Error Handling." The PCS layer on the receive side of an Aurora interface must be capable of performing channel bonding by buffering incoming data, recognizing Channel Bonding blocks, and communicating with other lanes to adjust the incoming data streams in each lane so that the Channel Bonding blocks are presented at the output of the PCS from all the lanes simultaneously.

The depth of the buffer used in the PCS layer determines the maximum skew the Aurora interface can support.

## 5.6 Clock Compensation

Clock Compensation idle blocks can be used to prevent data corruption due to small differences between the recovered clock and local reference clock. These differences occur when independent clock sources are used to drive the clocks on the TX and RX side of a connection. If a shared clock is used, Clock Compensation blocks are not required.

If the channel does not use a shared clock, Clock Compensation blocks must be transmitted for three consecutive cycles on every lane in the channel at least once every 10,000 cycles.

Lanes performing clock compensation must use elastic buffers to move data from the recovered clock domain to their local clock domain. If the elastic buffer is almost full, Clock Compensation idle blocks should not be written to the buffer to prevent it from filling. If the elastic buffer is almost empty, Clock Compensation idle blocks should be read from buffer without being removed, to prevent the buffer from underflowing. This algorithm prevents buffer overflow or underflow and allows asynchronous operation. Clock Compensation idle blocks are treated as regular idles in all other parts of the receiver.

# Channel Control

## 6.1 Overview

Aurora interfaces can communicate using channels composed of one or many Aurora lanes. This section defines the striping rules for using multi-lane channels. These rules dictate how block codes are distributed among the channel-bonded lanes of an Aurora channel on each cycle.

## 6.2 Idle Block Striping

### 6.2.1 Not Ready Blocks

Not Ready idle blocks must be transmitted on all lanes simultaneously.

### 6.2.2 Idle Blocks

Regular idle blocks can be transmitted on any lane, restricted only by the Aurora 64B/66B rules of block code priority defined in Section 2.2 "Block Codes."

### 6.2.3 Clock Compensation Blocks

Clock Compensation idle blocks must be transmitted on all lanes simultaneously.

### 6.2.4 Channel Bonding Blocks

Channel Bonding idle blocks must be transmitted on all lanes simultaneously.

## 6.3 Native Flow Control Striping

Any number of Native Flow Control blocks can be transmitted per cycle. These blocks must obey the Aurora 64B/66B rules of block code priority defined in Section 2.2 "Block Codes." All Native Flow Control blocks transmitted on the same cycle must carry the same PAUSE code and XOFF bit setting.

When multiple Native Flow Control blocks arrive on the same clock cycle, the receiver must process only one block, choosing any of the blocks for processing.

## 6.4 Frame Data Striping

Frame data is striped as follows:

- Each lane in the channel is assigned an index number.
    - ♦ The least-significant lane is assigned index number 0.
    - ♦ All lanes are numbered in order from most significant to least significant.
- On every clock cycle, frame data is broken into Data blocks and Separator blocks.

Each Aurora interface must receive data using the same indexing as used by its channel partner to transmit the data.

## 6.5 Strict-Alignment Frame Data Striping

Strict-alignment frame data striping is the same as frame data striping with the following additional restrictions:

- If any lanes transmit frame data on a cycle, all lanes must transmit data
- If a frame ends on a given cycle, all lanes must be filled with data except for the lane with the separator, and any of the lanes with higher indexes than the separator.

  ***Note:*** Lanes with higher indexes than the separator cannot contain either flow control or user data blocks.

## 6.6 User Flow Control Striping

User flow control messages use frame data striping rules — the UFC header block must precede the UFC data blocks. There can be no frame data interspersed with user flow control data.

## 6.7 Strict-Alignment User Flow Control Striping

Strict-alignment user flow control striping is the same as frame data striping with the following additional restrictions:

- User flow control headers may be transmitted only in the least-significant lane (min index)
- User flow control data must be transmitted the cycle after the user flow control header has been transmitted
- User flow control data may be transmitted only from the least-significant lane (min index)
- If any lanes transmit user flow control data on a cycle, all lanes must transmit user flow control data, except:
- If a user flow control message ends on a given cycle, all lanes must be filled with user flow control data until the message ends.

  ***Note:*** Lanes with higher indexes than the separator cannot contain either frame or user data blocks.

## 6.8 User K-Block Striping

All User K-Blocks must be transmitted in the same cycle. No frame data or flow controls can be interspersed with User K-Blocks in a given cycle.

## 6.9 Strict-Alignment User K-Block Striping

Strict-alignment User K-Blocks are striped as follows:

- If any lanes transmit User K-Blocks on a cycle, all lanes must transmit User K-Blocks
- Transmitted User K-Blocks must contain any of the nine valid User K-Block numbers

*Section 7*

# PMA Layer

## 7.1 Overview

This section specifies the following functions that are performed in the PMA layer of the Aurora 64B/66B protocol:

- Section 7.3 "Serialization" specifies how data and control information must be encoded before transmission through an Aurora channel, and decoded upon reception.
- Section 7.4 "Deserialization" specifies how data and control information must be scrambled before transmission and de-scrambled after reception.
- Section 7.5 "Clock Data Recovery" specifies how 64-bit blocks must be transformed to 66-bit blocks for transmission, and how 66-bit blocks must be transformed to 64-bit blocks upon reception.

## 7.2 Bit and Byte Ordering Convention

Bit and byte order notation in the Aurora protocol are opposite to the bit and byte order notation used in IEEE 802.3ae [Ref 2]. The leftmost bits of the encoded block code are the sync header bits. These are the most-significant bits of the block code. The leftmost byte of the block is the most significant byte. The rightmost byte is the least-significant byte.

## 7.3 Serialization

Each lane must transmit the most-significant bit of each block code first, followed in order by the remaining sync bit and the bits of the block code (Figure 7-1).

***Note:*** The order above is the same order specified for block codes in IEEE 802.3ae [Ref 2], but the names of the bit positions have been inverted.

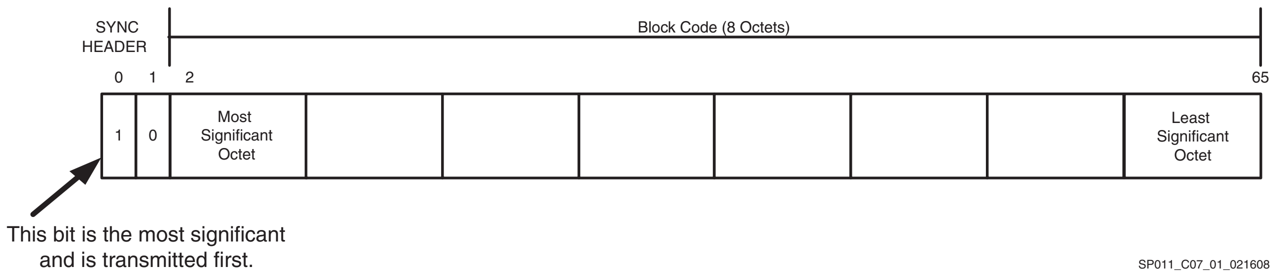


*Figure 7-1:* **Serialization Order for Aurora 64B/66B Block Codes**

## 7.4 Deserialization

Each lane should expect the data it deserializes to be in the serialization order presented in Section 7.2 “Bit and Byte Ordering Convention.”

## 7.5 Clock Data Recovery

Aurora interfaces are expected to be able to recover a high-speed serial clock from the incoming serial stream within the parameters given by the specification Section 8, “Electrical Specifications.”

# Electrical Specifications

## 8.1 Overview

This section specifies differential signaling in quantities representing the voltage difference between the true and complement signals, referred to peak-to-peak voltage. This voltage is twice that of the peak voltage of either the true or the complement signal. To ensure interoperability between drivers and receivers of different vendors and technologies, AC coupling at the receiver input is required. The Aurora 64B66B protocol does not define the electrical characteristics and is expected to work at the maximum line rate supported by the transceiver.

Refer to the respective GT family data sheet for the electrical specifications for the transmitter and receiver.

## 8.2 Signal Definition

The Aurora protocol uses differential signaling between ports. Figure 8-1 shows how the signals are defined, displaying waveforms for either a transmitter (TD and $\overline{\text{TD}}$) or a receiver (RD and $\overline{\text{RD}}$). Each signal swings between A and B volts. Using these waveforms, the definitions are as follows:

- The transmitter or receiver has a peak-to-peak range of A – B
- The differential signal ranges from + | A-B | to – | A–B |
- The differential peak-to-peak signal is 2 × (A–B)

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

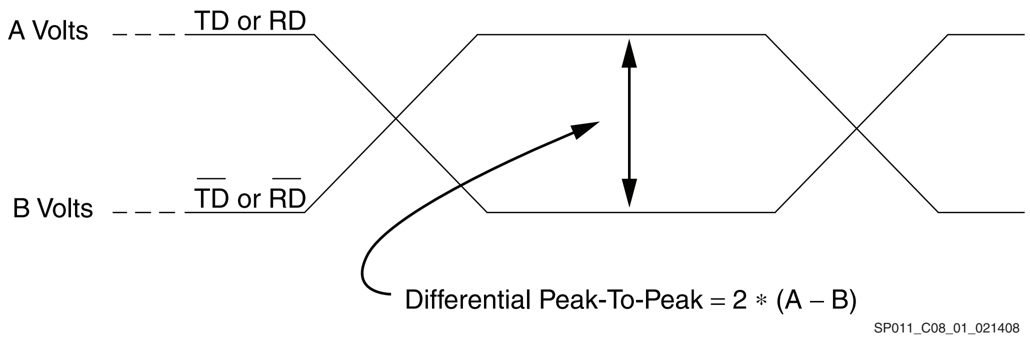

A Volts
TD or RD
B Volts
$\overline{\text{TD}}$ or $\overline{\text{RD}}$
Differential Peak-To-Peak = 2 ∗ (A – B)
SP011_C08_01_021408

*Figure 8-1:* **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 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.

## 8.3 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

*Appendix 1*

# *References*

1. IEEE 803.3ae
2. IEEE 802.3ae