Universal Chiplet Interconnect Express (UCIe)

Specification Revision 1.0

Februrary 24, 2022

Contents

Appendices

Figures

Tables

Terminology

Table 1. Terms and Definitions

Term	Definition
UCIe	Universal Chiplet Interconnect express
PCIe	Peripheral Component Interconnect Express
CXL	Compute eXpress Link
SoC	System on a Chip
USB	Universal Serial Bus
NVMe	Non-Volatile Memory express
SERDES	Serializer/Deserializer
PHY	Physical Layer
CPU	Central Processing Unit
I/O	Input/Output
DDR	Double Data Rate Memory
BER	Bit Error Ratio
EMIB	Embedded Multi-die Interconnect Bridge
CoWoS	Chip on Wafer on Substrate
CRC	Cyclic Redundancy Check
D2D	Die-to-Die
MHz	Mega Hertz
Ack	Acknowledge
AFE	Analog Front End
Endpoint	As defined in the *PCI Express Base Specification*.
LSB	Least Significant Bit
MSB	Most Significant Bit
NAck	Negatively acknowledge
PCIe	Peripheral Component Interconnect Express
Root Complex	As defined in the *PCI Express Base Specification*.
Root Port	As defined in the *PCI Express Base Specification*.
UI	Unit Interval
Vref	Reference voltage for receivers
UCIe Link	A UCIe connection between two chiplets. These chiplets are Link partners in the context of UCIe since they communicate with each other using a common UCIe Link.
UCIe die	This term is used throughout this specification to denote the logic associated with the UCIe Link on any given chiplet with a UCIe Link connection. It is used as a common noun to denote actions or messages sent or received by an implementation of UCIe.

Table 1. Terms and Definitions

Term	Definition
remote Link partner	This term is used throughout this specification to denote the logic associated with the far side of the UCIe Link; to denote actions or messages sent or received by the Link partner of a UCIe die.
Lane	A pair of signals mapped to physical bumps, one for Transmission, and one for Reception. A xN UCIe Link is composed of N Lanes.
Module	UCIe's main data path on the physical bumps is organized as a group of Lanes called a Module. For Standard Package, 16 Lanes constitute a single Module. For Advanced Package, 64 Lanes constitute a single Module.
UCIe Flit Mode	Operating Mode where CRC bytes are inserted and checked by the D2D Adapter. If applicable, Retry is also performed by the D2D Adapter.
UCIe Raw Mode	Operating Mode where all the bytes of a Flit are populated by the Protocol Layer.
{<SBMSG>}	Sideband message requests or responses are referred to by their names enclosed in curly brackets. Refer to Chapter 6.0 to see the mapping of sideband message names to relevant encodings. An asterisk in the <SBMSG> name is used to denote a group of messages with the same prefix or suffix in their name.

Reference Documents

Table 2. Reference Documents

Document	Document Location
PCI Express® Base Specification	https://pcisig.com
Compute Express Link Specification	https://computeexpresslink.org

Revision History

Table 3 lists the significant changes in different revisions.

Table 3. Revision History

Revision	Date	Description
0.7	*January 4, 2022*	First release
0.9	*February 6, 2022*	The following changes were included: 1) Details and support for Retimers 2) Protocol Layer interoperability rules 3) Replaced Optimized mode utilizing Spare Lanes with new Latency Optimized formats common for Advanced and Standard packages 4) Support for runtime Link Testing through Parity 5) Details on Training, Multi-module operation and Repair 6) Updates on channel metrics and bump map 7) Sideband encoding and format updates 8) Software view of UCIe and Register details 9) Interface and signal transition rule updates 10) CRC RTL code added in the appendix
1.0	*February 17, 2022*	Fixing formatting and grammatical errors, and added clarifications and corrections in a few places.

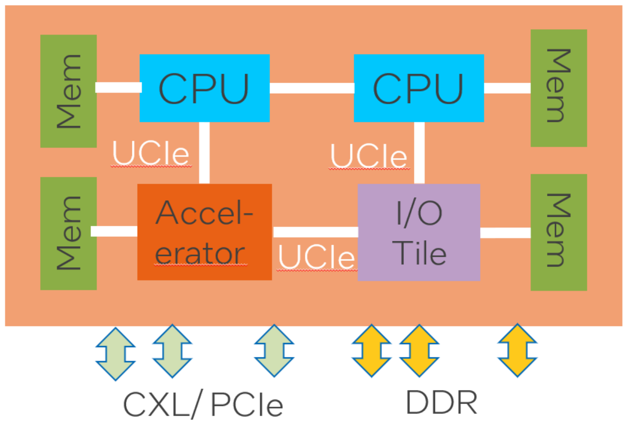
1.0 Introduction

This chapter provides an overview of the Universal Chiplet Interconnect express (UCIe) architecture. UCIe is an open, multi-protocol capable, on-package interconnect standard for connecting multiple dies on the same package. The primary motivation is to enable a vibrant ecosystem supporting disaggregated die architectures which can be interconnected using UCIe. UCIe supports multiple protocols (PCIe, CXL, and a raw mode that can be used to map any protocol of choice as long as both ends support it) on top of a common physical and Link layer. It encompasses the elements needed for SoC construction such as the application layer, as well as the form-factors relevant to the package (e.g., bump location, power delivery, thermal solution, etc.). The specification is defined to ensure interoperability across a wide range of devices having different performance characteristics. A well-defined debug and compliance mechanism is provided to ensure interoperability. It is expected that the specification will evolve in a backward compatible manner.

While UCIe supports a wide range of usage models, some are provided here as an illustration of the type of capability and innovation it can unleash in the compute industry. The initial protocols being mapped to UCIe are PCIe and CXL. The mappings for all protocols are done using a Flit format, including the raw mode. Both PCIe and CXL are widely used and these protocol mappings will enable more on-package integration by replacing the PCIe SERDES PHY and the PCIe/CXL LogPHY along with the Link level Retry with a UCIe Adapter and PHY in order to improve the power and performance characteristics. UCIe also supports a raw mode which is protocol-agnostic to enable other protocols to be mapped; while allowing usages such as integrating a stand-alone SERDES/transceiver tile (e.g., ethernet) on-package.

Figure 1 demonstrates an SoC package comprising of CPU dies, accelerator die(s) and I/O Tile die(s) connected through UCIe. The accelerator or I/O Tile can be connected to the CPU(s) using CXL transactions running on UCIe - leveraging the I/O, coherency, and memory protocols of CXL. The I/O tile can provide the external CXL, PCIe and DDR pins of the package. The accelerator can also be connected to the CPU using PCIe transactions running on UCIe. The CPU to CPU connectivity on-package can also use the UCIe interconnect, running coherency protocols.

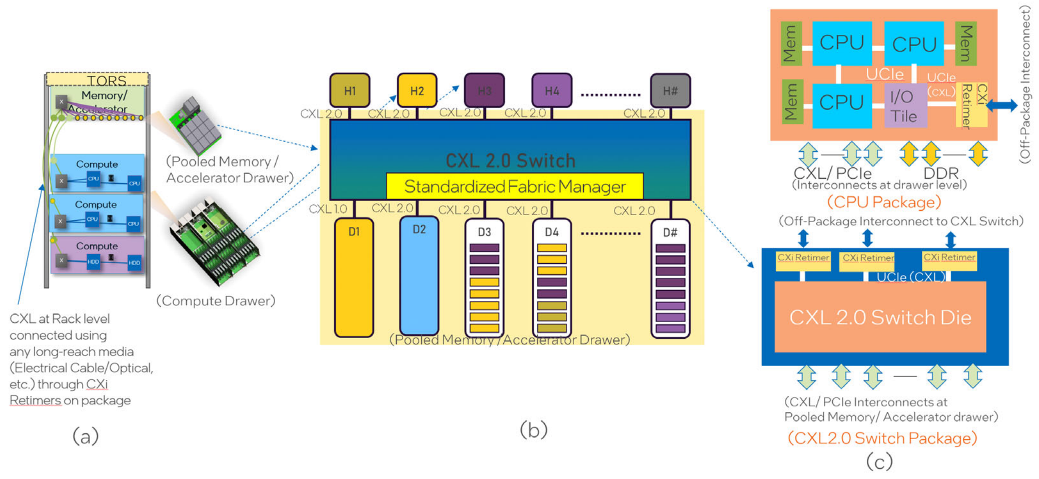
Figure 1. A Package comprising of CPU dies, Accelerator die(s), and I/O Tile die connected through UCIe



A UCIe Retimer may be used to extend the UCIe connectivity beyond the package using an Off-Package Interconnect. Examples of Off-Package Interconnect include electrical cable or optical cable or any other technology to connect packages at a Rack/Pod level as shown in Figure 2. The UCIe specification requires the UCIe Retimer to implement the UCIe interface to the die that it connects on its local package and ensure that the Flits are delivered across the two dies across the packages following UCIe protocol using the channel extension technology of its choice.

Figure 2 demonstrates a rack/pod-level disaggregation using CXL 2.0 (or later) protocol. Figure 2a shows the rack level view where multiple compute nodes (virtual hierarchy) from different compute chassis connect to a CXL switch which connects to multiple CXL accelerators/Type-3 memory devices which can be placed in one or more separate drawer. The logical view of this connectivity is shown in Figure 2b, where each "host" (H1, H2,...) is a compute drawer. Each compute drawer connects to the switch using an Off-Package Interconnect running CXL protocol through a UCIe Retimer, as shown in Figure 2c. The switch also has co-package Retimers where the Retimer tiles connect to the main switch die using UCIe and on the other side are the PCIe/CXL physical interconnects to connect to the accelerators/memory devices, as shown in Figure 2c.

Figure 2. UCIe enabling long-reach connectivity at Rack/Pod Level


TORS
Memory/ Accelerator
Compute
CPU
Compute
CPU
Compute
HDD
(Pooled Memory / Accelerator Drawer)
(Compute Drawer)
CXL at Rack level connected using any long-reach media (Electrical Cable/Optical, etc.) through CXi Retimers on package
(a)
H1
H2
H3
H4
H#
CXL 2.0
CXL 2.0 Switch
Standardized Fabric Manager
CXL 1.0
D1
D2
D3
D4
D#
(Pooled Memory / Accelerator Drawer)
(b)
Mem
CPU
CPU
Mem
UCIe
UCIe (CXL)
Mem
CPU
I/O Tile
CXi Retimer
(Off-Package Interconnect)
CXL/ PCIe
DDR
(Interconnects at drawer level)
(CPU Package)
(Off-Package Interconnect to CXL Switch)
CXi Retimer
CXi Retimer
CXi Retimer
UCIe (CXL)
CXL 2.0 Switch Die
(CXL/ PCIe Interconnects at Pooled Memory/ Accelerator drawer)
(CXL2.0 Switch Package)
(c)

This version of UCIe Specification does not deal with 3D packaging. It allows two different packaging options: Standard Package (2D) and Advanced Package (2.5D). This covers the spectrum from lowest cost to best performance interconnects.

1. **Standard Package** - This packaging technology is used for low cost and long reach (10mm to 25mm) interconnects using traces on organic package/substrate, while still providing significantly better BER characteristics compared to off-package SERDES. Figure 3 shows an example application using the Standard Package option. Table 4 shows a summary of the characteristics of the Standard Package option with UCIe.

Figure 3. Standard Package interface

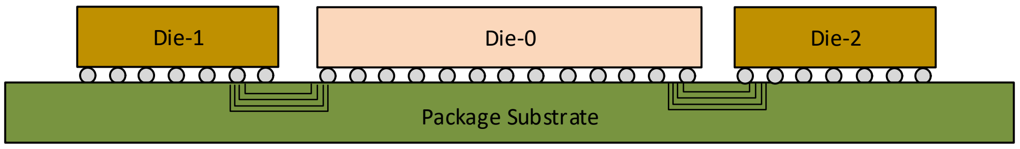


Table 4. Characteristics of UCIe on Standard Package

Index	Value
Supported speeds (per Lane)	4 GT/s, 8 GT/s, 12 GT/s, 16 GT/s, 24GT/s, 32 GT/s
Bump Pitch	100 um to 130 um
Channel reach (short reach)	10 mm
Channel reach (long reach)	25 mm
Raw Bit Error Rate (BER)[1]	1e-27 (<= 8 GT/s)
	1e-27 (short reach, 16 GT/s)
	1e-15 (long reach, 16 GT/s)
	1e-15 (>= 32 GT/s)

1. Refer to Chapter 5.0 for details about BER characteristics.

2. **Advanced Package** - This packaging technology is used for performance optimized applications. Consequently, the channel reach is short (less than 2mm) and the interconnect is expected to be optimized for high bandwidth and low latency with best performance and power efficiency characteristics. Figure 4, Figure 5 and Figure 6 show example applications using the Advanced Package option.

   Table 5 shows a summary of the main characteristics of the Advanced Package option.

Table 5. Characteristics of UCIe on Advanced Package

Index	Value
Supported speeds (per Lane)	4 GT/s, 8 GT/s, 12 GT/s, 16 GT/s, 24 GT/s, 32 GT/s
Bump pitch	25 um to 55 um
Channel reach	2 mm
Raw Bit Error Rate (BER)[1]	1e-27 (<=12GT/s)
	1e-15 (>=16GT/s)

1. Refer to Chapter 5.0 for details about BER characteristics.

Figure 4. Advanced Package interface: Example 1

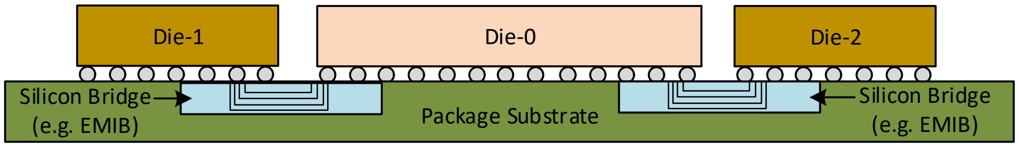


Figure 5. Advanced Package interface: Example 2

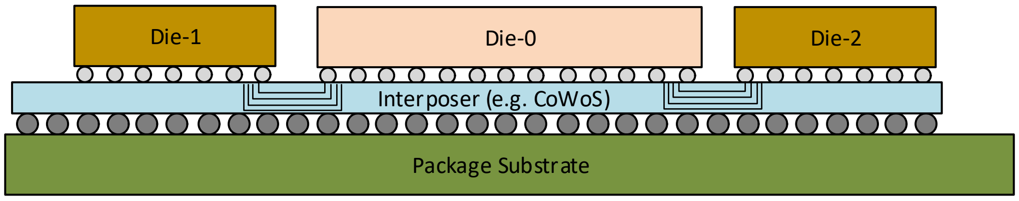
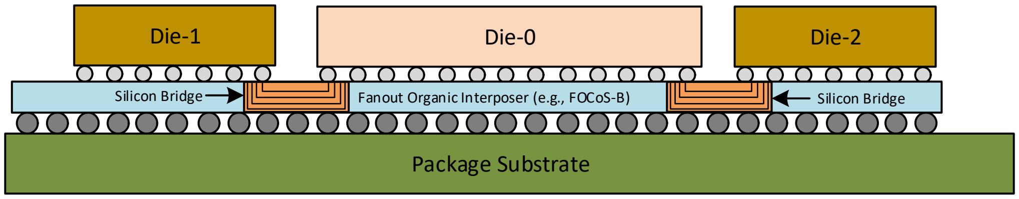


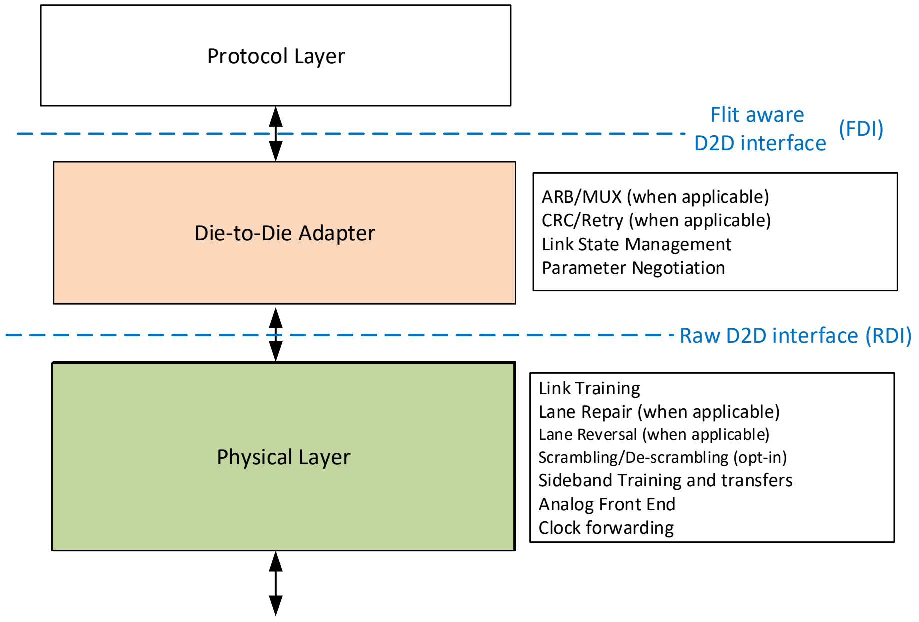
Figure 6. Advanced Package interface: Example 3



1.1 UCIe Components

UCIe is a layered protocol, with each layer performing a distinct set of functions. Figure 7 shows the three main components of the UCIe stack and the functionality partitioning between the layers. It is required for every component in the UCIe stack to be capable of supporting the advertised functionality and bandwidth.

Figure 7. UCIe Layers and functionalities


Protocol Layer
Flit aware D2D interface (FDI)
Die-to-Die Adapter
ARB/MUX (when applicable)
CRC/Retry (when applicable)
Link State Management
Parameter Negotiation
Raw D2D interface (RDI)
Physical Layer
Link Training
Lane Repair (when applicable)
Lane Reversal (when applicable)
Scrambling/De-scrambling (opt-in)
Sideband Training and transfers
Analog Front End
Clock forwarding

1.1.1 Protocol Layer

While the Protocol Layer may be application specific, UCIe specification provides examples of transferring CXL or PCIe protocols over UCIe Links. The following protocols are supported in UCIe for enabling different applications:

- PCIe 6.0 Flit Mode from *PCI Express Base Specification.*
- CXL 2.0 and later protocols from *Compute Express Link Specification.*
- Streaming Protocol: This offers generic modes for a user defined protocol to be transmitted using UCIe.

For each protocol, different optimizations and associated Flit transfers are available for transfer over UCIe. Chapter 2.0 and Chapter 3.0 cover the relevant details of different modes and Flit formats.

1.1.2 Die-to-Die (D2D) Adapter

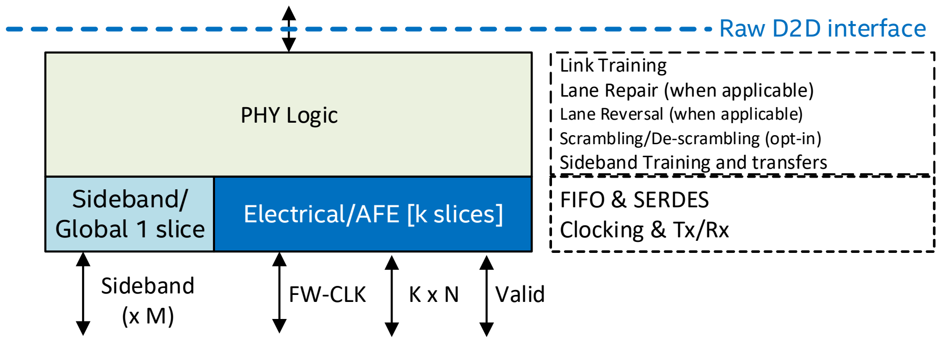
The D2D Adapter coordinates with the Protocol Layer and the Physical Layer to ensure successful data transfer across the UCIe Link. It minimizes logic on the main data path as much as possible giving a low latency, optimized data path for protocol Flits. When transporting CXL protocol, the ARB/MUX functionality required for multiple simultaneous protocols is performed by the D2D Adapter. For options where the Raw BER is less than 1e-27, a CRC and Retry scheme are provided in the UCIe specification, which is implemented in the D2D Adapter. Refer to section 3.7 for Retry rules.

D2D Adapter is responsible for coordinating higher level Link state machine and bring up, protocol options related parameter exchanges with remote Link partner, and when supported, power management coordination with remote Link partner. Chapter 3.0 covers the relevant details for the D2D Adapter.

1.1.3 Physical Layer

The Physical Layer has three sub-components as shown in Figure 8:

Figure 8. Physical Layer components


Raw D2D interface
PHY Logic
Link Training
Lane Repair (when applicable)
Lane Reversal (when applicable)
Scrambling/De-scrambling (opt-in)
Sideband Training and transfers
Sideband/ Global 1 slice
Electrical/AFE [k slices]
FIFO & SERDES
Clocking & Tx/Rx
Sideband (x M)
FW-CLK
K x N
Valid

UCIe’s main data path on the physical bumps is organized as a group of Lanes called a Module. A Module forms the atomic granularity for the structural design implementation of UCIe’s AFE. The number of Lanes per Module for Standard and Advanced package is specified in Chapter 4.0. A given instance of Protocol Layer or D2D adapter can send data over multiple modules where bandwidth scaling is required.

The physical Link of UCIe comprises of two connections:

1. **Sideband**:

   This connection is used for parameter exchanges, register accesses for debug/ compliance and coordination with remote partner for Link training and management. It consists of a forwarded clock pin and a data pin in each direction. The clock is fixed at 800MHz regardless of the main data path speed. The sideband logic for UCIe’s Physical Layer must be on auxiliary power and an "always on" domain. Each module has its own set of sideband pins.

   For the Advanced Package option, a redundant pair of clock and data pins in each direction is provided for repair.

2. **Main-band**:

   This connection constitutes the main data path of UCIe. It consists of a forwarded clock, a data valid pin, and N Lanes of data per module.

   For the Advanced Package option, N=64 (also referred to as x64) and overall four extra pins for Lane repair are provided in the bump map.

   For the Standard Package option, N=16 (also referred to as x16) and no extra pins for repair are provided.

The Logical Physical Layer co-ordinates the different functions and their relative sequencing for proper Link bring up and management (for example, sideband transfers, main-band training and repair etc.). Chapter 4.0 and Chapter 5.0 cover the details on Physical Layer operation.

1.1.4 Interfaces

UCIe defines the interfaces between the Physical Layer and the D2D Adapter (Raw D2D Interface), and the D2D Adapter and the Protocol Layer (Flit-aware D2D Interface) in Chapter 8.0.

The motivation for this is two-fold:

- Allow vendors and SoC builders to easily mix and match different layers from different IP providers at low integration cost and faster time to market. (for example, getting a Protocol Layer to work with the D2D Adapter and Physical Layer from any different vendor that conforms to the interface handshakes provided in this specification)
- Given that inter-op testing during post-silicon has greater overhead and cost associated with it, a consistent understanding and development of Bus Functional Models (BFMs) can allow easier IP development for this stack.

1.2 UCIe Configurations

This section describes the different configurations and permutations allowed for UCIe operation.

1.2.1 Single module configuration

A single module configuration is a x16 data interface on Standard Package and a x64 data interface in Advanced Package as shown in Figure 9 and Figure 10. In multiple instantiations of a single module configuration, each module must operate independently (for example transferring independent data at different frequency).

1.2.2 Multi Module Configurations:

This spec allows two and four module configurations. The modules in two and four module configuration must not operate independently. Examples of two and four module configurations are shown in Figure 11 and Figure 12 respectively.

Figure 9. Single module configuration: Advanced Package

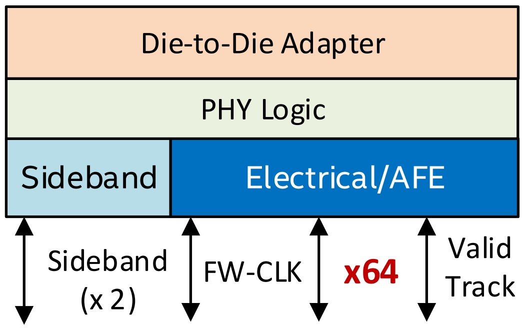


Figure 10. Single module configuration: Standard Package

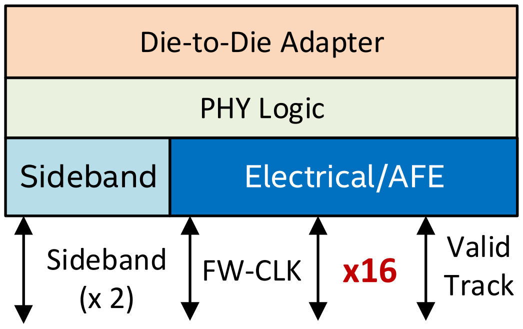


Figure 11. Two module configuration for Standard Package

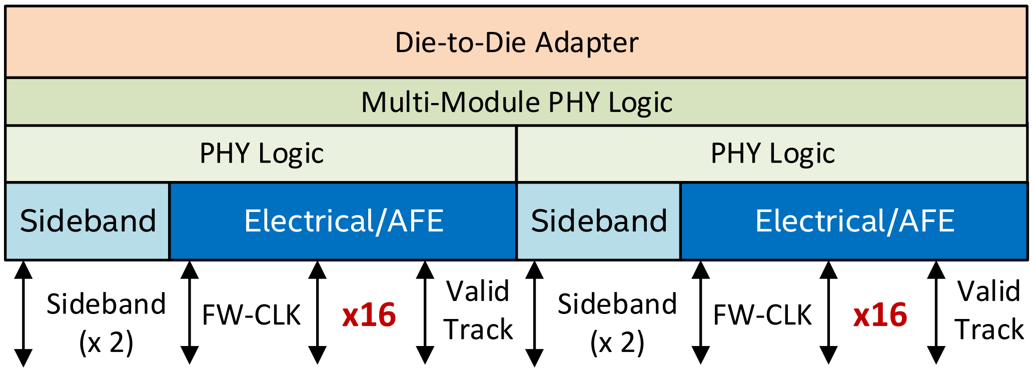
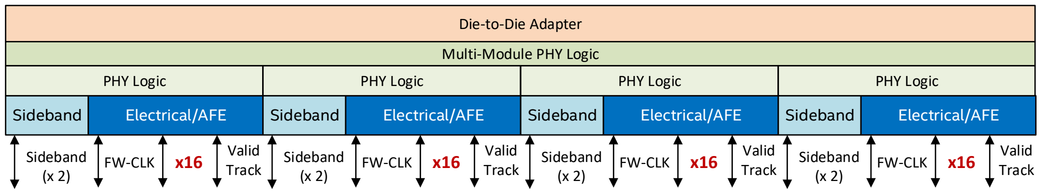


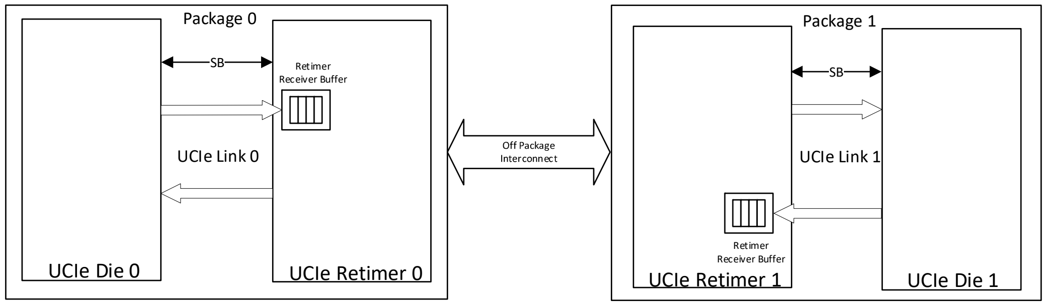
Figure 12. Four module configuration for Standard Package



1.3 UCIe Retimers

As described previously, UCIe Retimers are used to enable different types of Off Package Interconnects to extend the channel reach between two UCIe Dies on different packages. Each UCIe Retimer has a local UCIe Link connection to a UCIe die on-package as well as an external connection for longer reach. Figure 13 shows a high level block diagram demonstrating a system utilizing UCIe Retimers to enable an Off Package Interconnect between UCIe Die 0 and UCIe Die 1. UCIe Retimer 0 and UCIe Die 0 are connected through UCIe Link 0 within Package 0. UCIe Retimer 1 and UCIe Die 1 are connected through UCIe Link 1 within Package 1. The terminology of "remote Retimer partner" is used to reference the UCIe Retimer die connected to the far side of the Off Package Interconnect.

Figure 13. Block Diagram for UCIe Retimer Connection



The responsibility of a UCIe Retimer include:

- Reliable Flit transfer across the Off Package Interconnect. Three options are available for achieving this as described below:
  a. The Retimer is permitted to use the FEC and CRC natively defined by the underlying specification of the protocol it carries (e.g., PCIe or CXL) as long as the external interconnect conforms to the underlying error model (e.g., BER and error correlation) of the specification corresponding to the protocol it transports. The UCIe Links would be setup to utilize the Raw Mode to tunnel native bits of the protocol it transports (e.g., PCIe or CXL Flits). In this scenario, the queue sizes (Protocol Layer buffers) must be adjusted on the UCIe Dies to meet the underlying round trip latency.
  b. The Retimer is permitted to provide the necessary FEC, CRC and Retry capabilities to handle the BER of the Off Package Interconnect. In this case, the Flits undergo three independent Links; each UCIe Retimer performs an independent Ack/Nak for Retry with the UCIe die within its package and a separate independent Ack/Nak for Retry with the remote Retimer partner.
  c. The Retimer provides its own FEC by replacing the native PCIe or CXL defined FEC with its own, or adding its FEC in addition to the native PCIe or CXL defined FEC, but takes advantage of the built in CRC and Replay mechanisms of the underlying protocol. In this scenario, the queue sizes (Protocol Layer buffers, Retry buffers) must be adjusted on the UCIe Dies to meet the underlying round trip latency.
- Resolution of Link and Protocol Parameters with remote Retimer partner to ensure interoperability between UCIe Dies end-to-end (E2E). For example, Retimers are permitted to force the same Link width, speed, protocol (including any relevant protocol specific parameters) and Flit formats on both Package 0 and Package 1 in Figure 13. The specific mechanism of resolution including message transfer for parameter exchanges across the Off Package Interconnect is implementation specific for the Retimers and they must ensure a consistent operational mode

taking into account their own capabilities along with the UCIe Die capabilities on both Package 0 and Package 1. However, for robustness of the UCIe Links to avoid unnecessary timeouts in case the external interconnect requires a longer time to Link up or resolution of parameters with remote Retimer partner, UCIe Specification defines a "Stall" response to the relevant sideband messages that can potentially get delayed. The Retimers must respond with the "Stall" response within the rules of UCIe Specification to avoid such unnecessary timeouts while waiting for, or negotiating with remote Retimer partner. It is the responsibility of the Retimer to ensure the UCIe Link is not stalled indefinitely.

- Resolution of Link States for Adapter Link State Machine (LSM) or the RDI states with remote Retimer partner to ensure correct E2E operation. Refer to Chapter 3.0 for more details.
- Flow control and back-pressure:
  - Data transmitted from a UCIe die to a UCIe Retimer is flow-controlled using credits. These credits are on top of any underlying protocol credit mechanism (such as PRH, PRD credits in PCIe). These UCIe D2D credits must be for flow control across the two UCIe Retimers and any data transmitted to the UCIe Retimer must eventually be consumed by the remote UCIe die without any other dependency. Every UCIe Retimer must implement a Receiver Buffer for Flits it receives from the UCIe die within its package. The Receiver buffer credits are advertised to the UCIe die during initial parameter exchanges for the D2D Adapter, and the UCIe die must not send any data to the UCIe Retimer, if it does not have a credit for it. One credit corresponds to 256B of data (including any FEC, CRC etc.). Credit returns are overloaded on the Valid framing (refer to section 4.1.2). Credit counters at the UCIe Die are re-assigned to initial advertised value whenever RDI states transition away from Active. UCIe Retimer must drain or dump (as applicable) the data in its receiver buffer before re-entering Active state.
  - Data transmitted from a UCIe Retimer to a UCIe die is not flow controlled at the D2D adapter level. The UCIe Retimer may have its independent flow-control with the other UCIe Retimer if needed, which is beyond the scope of this specification.

1.4 UCIe Key Performance Targets

Table 6 gives a summary of the performance targets for UCIe

Table 6. UCIe Key Performance Targets

Metric		Advanced Package	Standard Package
Die Edge Bandwidth Density[1] (GB/s per mm)	4 GT/s	165	28
	8 GT/s	329	56
	12 GT/s	494	84
	16 GT/s	658	112
	24 GT/s	988	168
	32 GT/s	1317	224

Table 6. UCIe Key Performance Targets

Metric		Advanced Package	Standard Package
Energy Efficiency[2] (pJ/bit)	0.7V	0.5 (<=12 GT/s)	0.5 (4 GT/s, short reach)
		0.6 (>=16 GT/s)	1.0 (<=16 GT/s, long reach)
		-	1.25 (32 GT/s, long reach)
	0.5V	0.25 (<=12 GT/s)	0.5 (<=16 GT/s, long reach)
		0.3 (>=16 GT/s)	0.75 (32 GT/s, long reach)
Latency Target[3]		<=2ns	

1. Die Edge Bandwidth Density (as defined in Chapter 5.0) is with 45um (Advanced Package) and 110um (Standard Package) bump pitch
2. Energy Efficiency includes all the Physical Layer related circuitry: TX, RX, PLL, Clock Distribution etc. The reach of the channel (short vs long) is discussed in Chapter 5.0.
3. Latency includes the latency of the Adapter and the Physical Layer (FDI to pin interface delay) on TX and RX. Refer to Chapter 5.0 for details of Physical Layer latency.

1.5 Interoperability

Package designers need to make sure that dies that are connected on a package can inter-operate. This includes compatible package interconnect (e.g. advanced vs standard), protocols, voltage levels, etc. It is strongly recommended that dies adopt Transmitter voltage less than 0.85V; so that they can inter-operate with a wide range of process nodes in the foreseeable future.

This specification comprehends interoperability across a wide range of bump pitch for advanced packaging options. It is expected that over time the smaller bump pitches will be predominantly used. With smaller bump pitch, we expect designs will reduce the maximum advertised frequency (even though they can go to 32G) to optimize for area and to address the power delivery and thermal constraints of high bandwidth with reduced area. Table 7 summarizes these bump pitches across four groups. Interoperability is guaranteed within each group as well as across groups, based on the PHY dimension specified in Chapter 5.0. The performance targets provided in Table 6 are with the 45um bump pitch, based on the technology widely deployed at the time of publication of this specification.

Table 7. Groups for different bump pitches

Bump Pitch (um)	Minimum Frequency (GT/s)	Expected Maximum Frequency (GT/s)
Group 1: 25 - 30	4	12
Group 2: 31 - 37	4	16
Group 3: 38 - 44	4	24
Group 4: 45 - 55	4	32

2.0 Protocol Layer

Universal Chiplet Interconnect express (UCIe) maps PCIe, CXL as well as any Streaming Protocol. The following protocol mappings are used:

- PCIe 6.0 Flit Mode
- CXL 2.0, "CXL 68B-Enhanced Flit Mode", "CXL 256B Flit Mode”: If CXL is negotiated, each of CXL.io, CXL.cache and CXL.mem protocols are negotiated independently.
- Streaming Protocol: This offers generic modes for a user defined protocol to be transmitted using UCIe.

Note: CXL 1.1 is not supported. PCIe non-Flit Mode is supported in the form of CXL.io 68B Flit Mode.

The Protocol Layer requirements for interoperability:

- A Protocol Layer must support the 68B Flit Mode as defined in CXL 2.0 Specification for CXL.io protocol if it is advertising PCIe support.
- If a Protocol Layer supports "CXL 256B Flit Mode", it must support PCIe 6.0 Flit Mode and 68B Flit Mode as defined in CXL 2.0 Specification for CXL.io protocol.
- A Protocol Layer advertising CXL is permitted to only support CXL 2.0 or CXL 68B-Enhanced Flit Mode without supporting "CXL 256B Flit Mode" or PCIe 6.0 Flit Mode

IMPLEMENTATION NOTE:

The following tables summarize the mapping of the above rules from a specification version to a protocol mode:

Table 8. Specification to protocol mode requirements

Native Specification Supported	CXL 2.0 68B Flit Mode	CXL 256B Flit Mode	PCIe 6.0 Flit Mode
PCIe	Mandatory (CXL.io)	N/A	Optional
CXL 2.0	Mandatory	N/A	N/A
CXL 3.0	Mandatory	Mandatory	Mandatory (CXL.io)

Table 9. End Point vs Root Port Interoperability Matrix

EndPoint	**Root Port**			
	PCIe Gen5	**PCIe Gen6**	**CXL 2.0**	**CXL 3.0**
PCIe Gen5	CXL 2.0 68B Flit Mode (CXL.io)	CXL 2.0 68B Flit Mode (CXL.io)	CXL 2.0 68B Flit Mode (CXL.io)	CXL 2.0 68B Flit Mode (CXL.io)
PCIe Gen6	CXL 2.0 68B Flit Mode (CXL.io)	PCIe 6.0 Flit Mode	CXL 2.0 68B Flit Mode (CXL.io)	PCIe 6.0 Flit Mode
CXL 2.0	CXL 2.0 68B Flit Mode (CXL.io)	CXL 2.0 68B Flit Mode (CXL.io)	CXL 2.0 68B Flit Mode (CXL.io)	CXL 2.0 68B Flit Mode (CXL.io)
CXL 3.0	CXL 2.0 68B Flit Mode (CXL.io)	PCIe 6.0 Flit Mode	CXL 2.0 68B Flit Mode (CXL.io)	CXL 256B Flit Mode

The Die-to-Die (D2D) Adapter negotiates the protocol with remote Link partner and communicates it to the Protocol Layer(s). For each protocol, UCIe supports multiple modes of operation (that must be negotiated with the remote Link partner depending on the advertised capabilities, Physical Layer Status as well as usage models). These modes have different Flit formats and are defined to enable different trade-offs around efficiency, bandwidth and interoperability. The spectrum of supported protocols, advertised modes and Flit formats must be determined at SoC integration time or during the die-specific reset bring up flow. The Die-to-Die Adapter uses this information to negotiate the operational mode as a part of Link Training and informs the Protocol Layer over the Flit-aware Die-to-Die Interface (FDI). Refer to section 3.1 for parameter exchange rules in the Adapter.

The subsequent sections give an overview of the different modes from the Protocol Layer's perspective, hence they cover the supported modes of operation as sub-sections per protocol. The Protocol Layer is responsible for transmitting data over FDI in accordance with the negotiated mode and Flit format. The illustrations of the formats in this chapter are showing an example configuration of a 64B data path in the Protocol Layer mapped to a 64 Lane module of Advanced Package configuration on the Physical Link of UCIe. Certain Flit formats have dedicated byte positions filled in by the Adapter, and details associated with these are illustrated separately in Chapter 3.0. For other Link widths, refer to the Byte to Lane mappings defined in section 4.1.1.

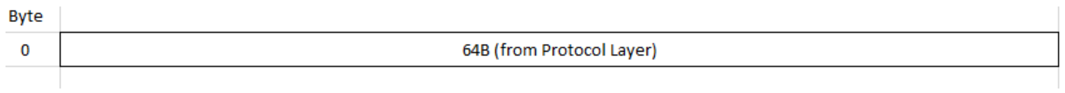
2.1 PCIe 6.0

UCIe only supports the Flit Mode defined in *PCI Express Base Specification Revision 6.0*. Please refer to *PCI Express Base Specification Revision 6.0* for the protocol definition. There are two UCIe operating modes supported for PCIe, and these are defined in the sub-sections below. In the figures illustrating the different formats for these modes, the gray colored bytes represent information inserted by the Adapter. In cases where these are shown as part of the Flit (for example in the Standard 256B Flit), the Protocol Layer must drive 0 on them on the Transmitter, and ignore them on the Receiver.

2.1.1 Raw Mode for PCIe 6.0

This mode is optional. The intended usage is for UCIe Retimers transporting PCIe protocol. An example usage of this mode is where a CPU and an I/O Device are in different Rack/chassis and connected through a UCIe Retimer using Off-Package Interconnect as shown in Figure 2. For Raw Mode, Retry, CRC and FEC are taken care of by the Protocol Layer. All 64 bytes are populated by the Protocol Layer. It is strongly recommended for the UCIe Retimers to check and count errors using either the parity bits of the 6B FEC or the Flit Mode 8B CRC defined in *PCIe Express Base Specification* for this mode to help characterize the D2D Link (in order to characterize or debug the Link which is the dominant source of errors).

Figure 14. Format for Raw Mode without spare Lanes

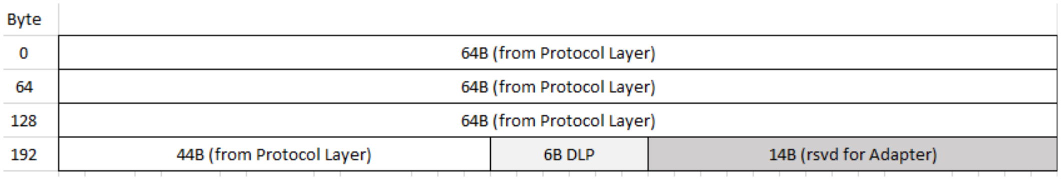


2.1.2 Flit Mode: Standard 256B Flit for PCIe 6.0

This mode is mandatory when PCIe protocol is supported. It is the standard Flit format defined in *PCI Express Base Specification* for Flit Mode and the main motivation of supporting this Flit format is to enable interoperability with vendors that only support the standard PCIe Flit formats. The Protocol Layer must follow the Flit formats for Flit transfer on FDI, driving 0 on the fields reserved for Die-to-Die Adapter. The PM and Link Management DLLPs are not used over UCIe. The other DLLPs (that are applicable for

PCIe Flit Mode) and Flit Status definitions follow the same rules including packing as defined in the *PCI Express Base Specification Revision 6.0*. It is strongly recommended for implementations to optimize out any 8b/10b, 128b/130b and non-Flit Mode related CRC/Retry or framing logic from the Protocol Layer in order to obtain area and power efficient designs for UCIe applications. Portions of the DLP bytes must be driven by the Protocol Layer for Flit_Marker assignment; refer to section 3.2.3 for details on how DLP bytes are driven.

Figure 15. Format for Standard 256B Flit



Byte			
0	64B (from Protocol Layer)		
64	64B (from Protocol Layer)		
128	64B (from Protocol Layer)		
192	44B (from Protocol Layer)	6B DLP	14B (rsvd for Adapter)

2.2 CXL 3.0 256B Flit Mode

Please refer to *Compute Express Link Specification* for details on the protocol layer messages and slot formats for "CXL 256B Flit Mode". There are four possible operational modes for this protocol, and these are defined in the sub-sections below. The gray colored bytes are inserted by the Adapter. In cases where these are shown as part of the main data path (for example in the Standard 256B Flit), the Protocol Layer must drive 0 on them on the Transmitter, and ignore them on the Receiver.

2.2.1 Raw Mode for "CXL 256B Flit Mode"

This mode is optional. The intended usage is for UCIe Retimers transporting "CXL 256B Flit Mode" protocol. An example usage of this mode is where a CPU and an I/O Device are in different Rack/chassis and connected through a UCIe Retimer using Off-Package Interconnect. Retry, CRC and FEC are taken care of by the Protocol Layer. All 64 bytes are populated by the Protocol Layer. Figure 14 shows an example of the transfer for this mode. It is strongly recommended for the UCIe Retimers to check and count errors using either the parity bits of the 6B FEC or the Flit Mode 8B CRC or 6B CRC; depending on which Flit format was enabled. This helps to characterize and debug the D2D Link which is the dominant source of errors. For CXL.cachemem, Viral or poison containment (if applicable) must be handled within the Protocol Layer for this mode.

2.2.2 Flit Mode: Latency-Optimized 256B Flit for "CXL 256B Flit Mode"

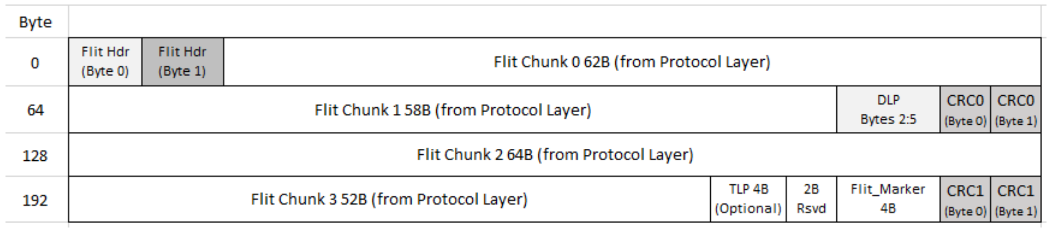
The support for this mode is strongly recommended for "CXL 256B Flit Mode" over UCIe. Two Flit formats are defined for this mode which are derived from the Latency Optimized Flits defined in *Compute Express Link 3.0 Specification*. The only difference for the second Flit format is that it gives higher Flit packing efficiency by providing Protocol Layer with extra bytes - for CXL.io this results in extra 4B of TLP information, and for CXL.cachemem it results in an extra 14B H-slot that can be packed in the Flit. Support for the first or second format are negotiated at the time of Link bring up.

The Latency-Optimized Mode enables the Protocol Layer to consume the Flit at 128B boundary, reducing the accumulation latency significantly. When this mode is negotiated, the Protocol Layer must follow this Flit format for Flit transfer on FDI, driving 0 on the fields reserved for Die-to-Die Adapter.

The Ack, Nak, PM and Link Management DLLPs are not used over UCIe for CXL.io for any of the 256B Flit Modes. The other DLLPs and Flit_Marker definitions follow the same rules as defined in the *Compute Express Link Specification*. Portions of the DLP bytes must be driven by the Protocol Layer for Flit_Marker assignment; refer to section 3.2.3 for details on how DLP bytes are driven.

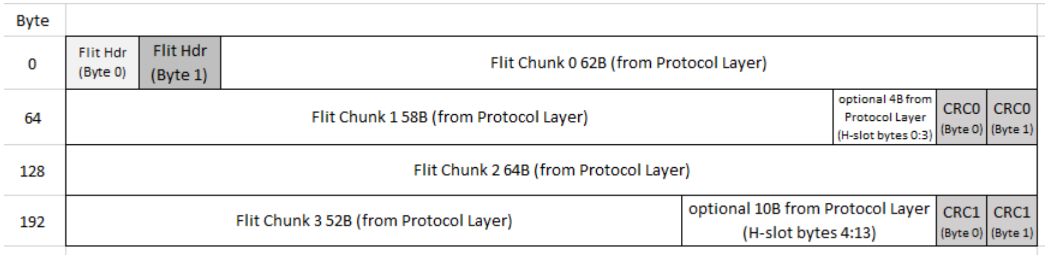
For CXL.cachemem for this mode, FDI provides a **`lp_corrupt_crc`** signal to help optimize for latency while guaranteeing Viral containment. Refer to Chapter 8.0 for details of interface rules for Viral containment.

Figure 16. Format for CXL.io Latency-Optimized 256B Flit



Byte						
0	Flit Hdr (Byte 0)	Flit Hdr (Byte 1)	Flit Chunk 0 62B (from Protocol Layer)			
64	Flit Chunk 1 58B (from Protocol Layer)			DLP Bytes 2:5	CRC0 (Byte 0)	CRC0 (Byte 1)
128	Flit Chunk 2 64B (from Protocol Layer)					
192	Flit Chunk 3 52B (from Protocol Layer)	TLP 4B (Optional)	2B Rsvd	Flit_Marker 4B	CRC1 (Byte 0)	CRC1 (Byte 1)

Figure 17. Second Format for CXL.cachemem Latency-Optimized 256B Flit



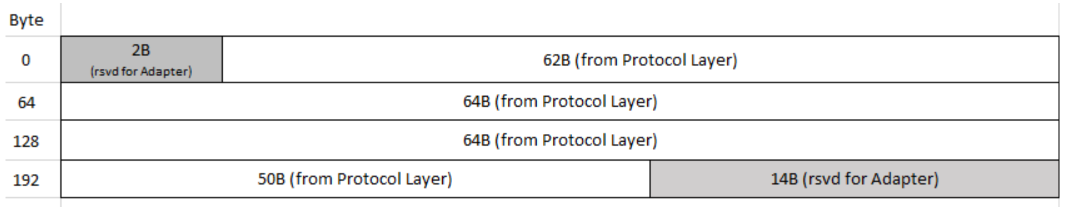
Byte					
0	Flit Hdr (Byte 0)	Flit Hdr (Byte 1)	Flit Chunk 0 62B (from Protocol Layer)		
64	Flit Chunk 1 58B (from Protocol Layer)		optional 4B from Protocol Layer (H-slot bytes 0:3)	CRC0 (Byte 0)	CRC0 (Byte 1)
128	Flit Chunk 2 64B (from Protocol Layer)				
192	Flit Chunk 3 52B (from Protocol Layer)		optional 10B from Protocol Layer (H-slot bytes 4:13)	CRC1 (Byte 0)	CRC1 (Byte 1)

2.2.3 Flit Mode: Standard 256B Flit for "CXL 256B Flit Mode"

This mode is mandatory when "CXL 256B Flit Mode" protocol is supported. It is the Standard Flit format defined in *Compute Express Link Specification* for 256B Flit Mode and the main motivation of supporting this Flit format is to enable interoperability with vendors that only support the Standard 256B Flit formats. The Protocol Layer must follow the Flit formats for Flit transfer on FDI, driving 0 on the fields reserved for Die-to-Die Adapter. The Ack, Nak, PM and Link Management DLLPs are not used over UCIe for CXL.io. The other DLLPs and Flit Status definitions follow the same rules and packing as defined in the *Compute Express Link Specification*. Portions of the DLP bytes must be driven by the Protocol Layer for Flit_Marker assignment; refer to section 3.2.3 for details on how DLP bytes are driven. Figure 18 shows the Flit format for this mode.

For CXL.cachemem for this mode, FDI provides a **`lp_corrupt_crc`** signal to help optimize for latency while guaranteeing Viral containment. Refer to section 8.2 for details of interface rules for Viral containment.

Figure 18. Format for Standard 256B Flit for CXL


Byte
0
2B
(rsvd for Adapter)
62B (from Protocol Layer)
64
64B (from Protocol Layer)
128
64B (from Protocol Layer)
192
50B (from Protocol Layer)
14B (rsvd for Adapter)

2.3 CXL 2.0 or "CXL 68B-Enhanced Flit Mode"

Pleaser refer to *Compute Express Link Specification* for details on the protocol layer messages and slot formats for CXL 2.0 or "CXL 68B-Enhanced Flit Mode". There are two operational modes possible for each of these protocols, and these are defined in the sub-sections below. The gray colored bytes are inserted by the Adapter.

2.3.1 Raw Mode for CXL 2.0 or "CXL 68B-Enhanced Flit Mode"

This mode is optional. The intended usage is for UCIe Retimers transporting "CXL 64B Flit Mode" protocol. An example usage of this mode is where a CPU and an I/O Device are in different Rack/chassis and connected through a UCIe Retimer using an Off-Package Interconnect. Retry and CRC are taken care of by the Protocol Layer. All 64 bytes are populated by the Protocol Layer. Figure 14 shows an example of the transfer for this mode.

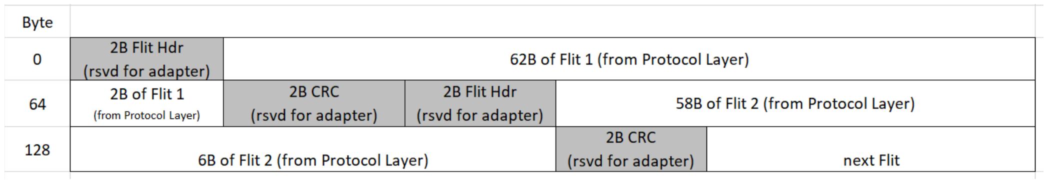
2.3.2 Flit Mode: 68B Flit for CXL 2.0 or "CXL 68B-Enhanced Flit Mode"

This mode is mandatory when CXL 2.0 or "CXL 68B-Enhanced Flit Mode" protocols are negotiated. This follows the corresponding 68B Flit format defined in *Compute Express Link Specification* and the main motivation of supporting this Flit format is to enable interoperability with vendors that only support the baseline CXL formats. The Protocol Layer presents 64B of the Flit (excluding the Protocol ID and CRC) on FDI, and the Die-to-Die Adapter inserts a 2B Flit Header and 2B CRC and performs the byte shifting required to arrange the Flits in the format shown in Figure 19.

PM or Ack or Nak DLLPs are not used for CXL.io in this mode. Credit updates and other remaining DLLPs for CXL.io are transmitted in the Flits as defined in the *Compute Express Link Specification*. For CXL.io, the Transmitter must not implement retry in the Protocol Layer (since Retry is handled in the Adapter). In order to keep the framing rules consistent, Protocol Layer for CXL.io must still drive the LCRC bytes with a fixed value of 0, and the Receiver must ignore these bytes and never send any Ack or Nak DLLPs.

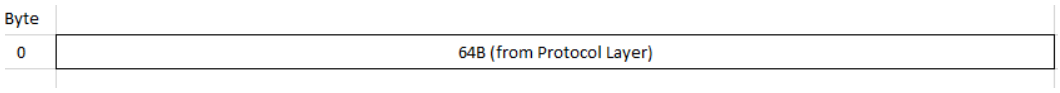
For CXL.cachemem, the "Ak" field in the Flit is reserved, and the Retry Flits are not used (since Retry is handled in the Adapter). Link Initialization begins with sending the INIT.Param Flit without waiting for any received Flits. Viral containment (if applicable) must be handled within the Protocol Layer for the 68B Flit Mode.

Figure 19. Format for 68B Flit on the UCIe Link



Byte				
0	2B Flit Hdr (rsvd for adapter)	62B of Flit 1 (from Protocol Layer)		
64	2B of Flit 1 (from Protocol Layer)	2B CRC (rsvd for adapter)	2B Flit Hdr (rsvd for adapter)	58B of Flit 2 (from Protocol Layer)
128	6B of Flit 2 (from Protocol Layer)	2B CRC (rsvd for adapter)	next Flit	

Figure 20. Format for the 68B Flit on FDI



Byte	
0	64B (from Protocol Layer)

2.4 Streaming protocol

This is the default protocol that must be advertised if none of the PCIe or CXL protocols are going to be advertised and negotiated with the remote Link partner. There is a single operational mode defined in UCIe Specification for this protocol, but vendor defined extensions are permitted.

2.4.1 Raw Mode for Streaming protocol

This is mandatory for Streaming protocol support. All 64 bytes are populated by the Protocol Layer. Figure 14 shows the transfer for this mode across the physical Link for a x64 module.

2.5 Summary of supported modes

The different modes of operation for each protocol are summarized in Table 10. Details of their formats are defined in Chapter 3.0.

Table 10. Modes of operation for supported protocols

Operation Mode	PCIe 6.0 Flit Mode	CXL 2.0 or "CXL 68B-Enhanced Flit Mode"	"CXL 256B Flit Mode"	Streaming Protocol
Raw Mode	Optional	Optional	Optional	Mandatory
Flit Mode: Standard 256B Flit	Mandatory	N/A	Mandatory	N/A
Flit Mode: Latency-Optimized 256B Flit	N/A	N/A	Strongly Recommended	N/A
Flit Mode: 68B Flit	N/A	Mandatory	N/A	N/A

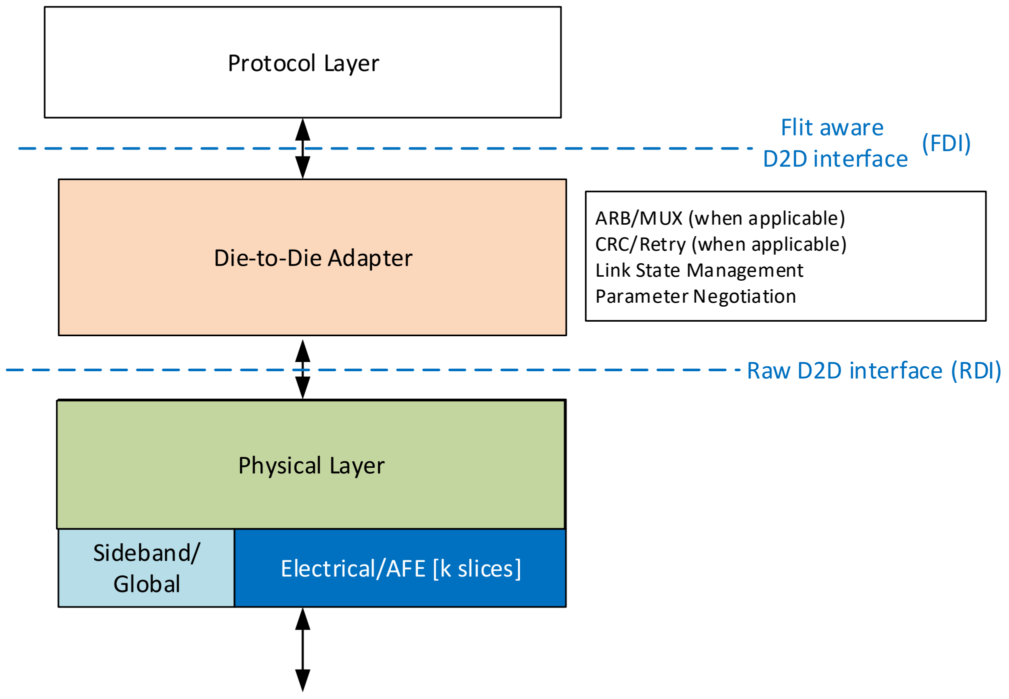
3.0 Die-to-Die Adapter

The Die-to-Die Adapter is responsible for:

- Reliable data transfer (performing CRC computation and Retry, or parity computation when applicable)
- Arbitration and Muxing (in case of multiple Protocol Layers)
- Link State Management
- Protocol and Parameter negotiation with the remote Link partner.

Figure 21 shows a high level description of the functionality of the Adapter.

Figure 21. Functionalities in the Die-to-Die Adapter

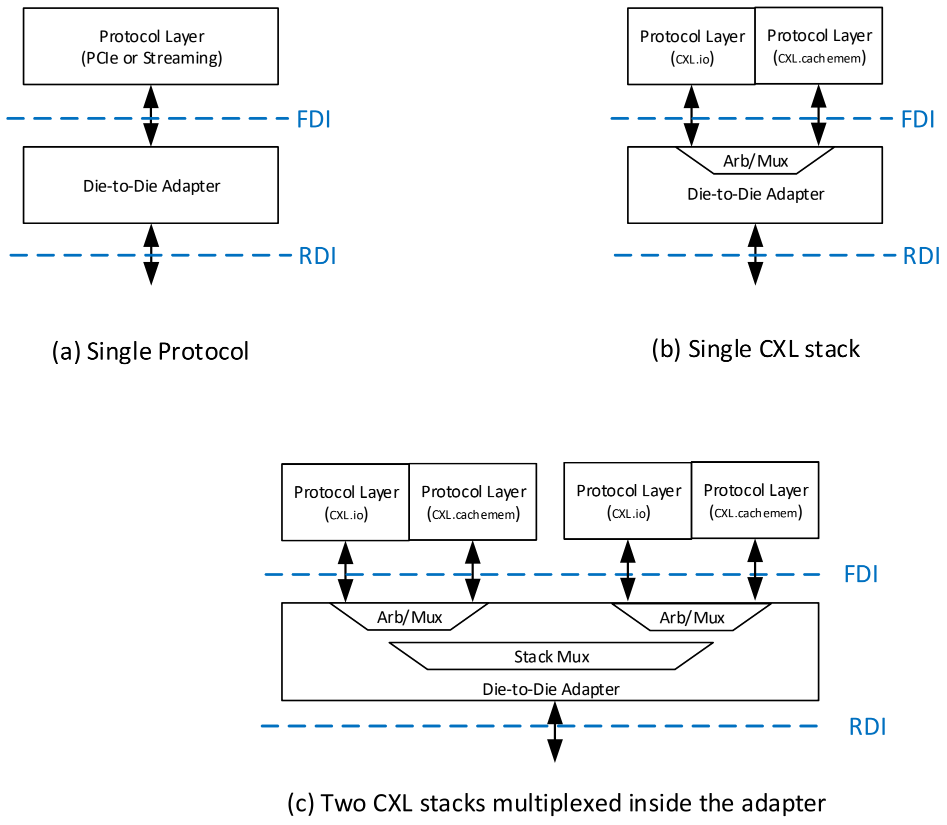


The Adapter interfaces to the Protocol Layer using one or more instances of the Flit-aware Die-to-Die interface (FDI), and it interfaces to the Physical Layer using the raw Die-to-Die interface (RDI). Refer to Chapter 8.0 for interface details and operation.

The D2D Adapter must follow the same rules as the Protocol Layer for protocol interoperability requirements. Figure 22 shows example configurations for the Protocol Layer and the Adapter, where the Protocol identifiers (e.g. PCIe) only signify the protocol, and not the flit formats. In order to provide cost and efficiency trade-offs, UCIe allows configurations in which two protocol stacks are multiplexed onto the same physical Link. Two stacks multiplexed on the same physical Link is supported when each protocol stack needs half the bandwidth that the Physical Layer provides. When this feature is supported and negotiated, the Adapter must guarantee that it will not send consecutive flits from the same protocol stack on the Link. In order to avoid bursts due to rate mismatch between the Physical Link speed and the RDI interface bandwidth, it is required for the Adapter to insert an NOP Flit after a protocol Flit, if there is no Flit insertion from the second Protocol Layer or before the Adapter is pausing the data stream. Note that there is no fixed pattern of Flits alternating from different Protocol Layers; for example, a Flit from Protocol Stack 0 followed by a NOP Flit, followed by a Flit from Protocol Stack 0 is a valid transmit pattern. An NOP Flit is defined as a Flit where the protocol identifier in the Flit Header corresponds to the D2D Adapter, and the body of the Flit is filled with all 0 data (The NOP Flit is defined for all Flit formats supported by the Adapter, including the 68B Flit). It is permitted for NOP flits to bypass the Retry buffer, as long as the Adapter guarantees that it is not sending consecutive Flits for any of the Protocol Layers. On the receiving side, the Adapter must not forward these NOP flits to the Protocol Layer. The receiving Protocol Layer must be capable of sinking at the same rate; i.e., it can receive consecutive chunks of the same Flit at the maximum Link speed, but it will not receive consecutive Flits. In addition to the transfer rate, both protocol stacks must operate with the same protocol and Flit formats, and this configuration is only applicable for formats where the Adapter is inserting the Flit header bytes. Each stack is given a single bit stack identifier that is carried along with the Flit header for de-multiplexing of Flits on the Receiver. The Stack Mux shown maintains independent Link state machines for each protocol stack. Link State transition related messages on sideband are also with unique message codes to identify which stack's Link State Management is affected by that message.

The following sections describe the details of Adapter functionality.

Figure 22. Example configurations

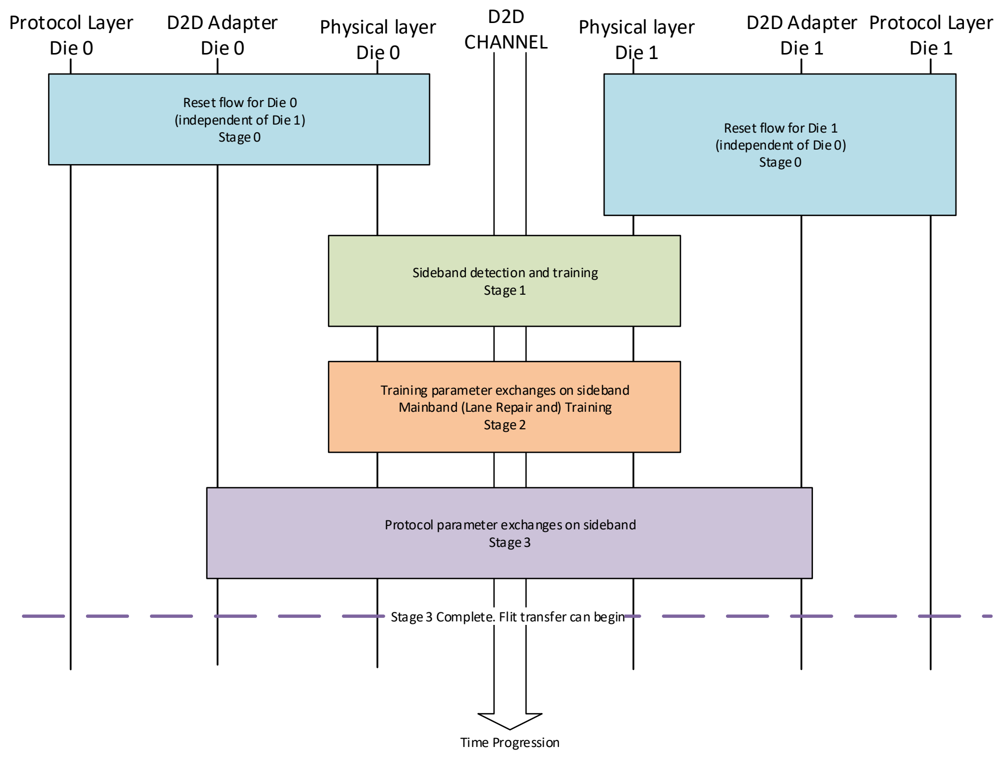


3.1 Link Initialization

Link Initialization consists of four stages before protocol Flit transfer can being on mainband. Figure 23 shows the high level steps involved in the Link initialization flow for

UCIe. Stage 0 is die-specific and happens independently for each die; the corresponding boxes in Figure 23 are of different sizes to denote that different die can take different amount of time to finish Stage 0. Stage 1 involves sideband detection and training. Stage 2 involves main-band training and repair. Details of Stage 1 and 2 are given in Chapter 4.0. Stage 3 involves parameter exchanges between Adapters to negotiate the protocol and Flit formats and is covered in section 3.1.1.

Figure 23. Stages of UCIe Link initialization


Protocol Layer Die 0
D2D Adapter Die 0
Physical layer Die 0
D2D CHANNEL
Physical layer Die 1
D2D Adapter Die 1
Protocol Layer Die 1
Reset flow for Die 0 (independent of Die 1) Stage 0
Reset flow for Die 1 (independent of Die 0) Stage 0
Sideband detection and training Stage 1
Training parameter exchanges on sideband Mainband (Lane Repair and) Training Stage 2
Protocol parameter exchanges on sideband Stage 3
Stage 3 Complete. Flit transfer can begin
Time Progression

3.1.1 Stage 3 of Link Initialization: Adapter Initialization

Stage 2 is complete when the RDI state machine moves to Active State. The initialization flow on RDI to transition the state from Reset to Active is described in section 8.1.6. Once Stage 2 is complete, the Adapter must follow a sequence of steps in order to determine Local Capabilities, complete Parameter Exchanges, and bring FDI state machine to Active.

3.1.1.1 Part 1: Determine Local Capabilities

The Adapter must determine the results of Physical Layer training and if Retry is needed for the given Link speed and configuration. Refer to section 3.7 for the rules on when Retry must be enabled for Link operation. If the Adapter is capable of supporting Retry, it must advertise this capability to the remote Link partner during Parameter Exchanges. For UCIe Retimers, the Adapter must also determine the credits to be advertised for the Retimer Receiver Buffer. Each credit corresponds to 256B of Mainband data storage.

3.1.1.2 Part 2: Parameter Exchange with remote Link partner

The following list of capabilities must be negotiated between Link partners. The capabilities (if enabled) are transmitted to the remote Link partner using a sideband message. In the section below, "advertised" means that the corresponding bit is 1b in the {AdvCap.Adapter} sideband message.

1. "Raw_Mode": This parameter is advertised if the corresponding bit in the UCIe Link Control register is 1b. Software/Firmware enables this based on system usage scenario. If the PCIe or CXL protocols are not supported, and Streaming Protocol is to be negotiated without any vendor-specific extensions, "Raw_Mode" must be 1b and advertised.
2. "68B Flit Mode": This is a protocol parameter. This must be advertised if the Adapter and Protocol Layer support CXL 68B Flit Mode or PCIe Non-Flit Mode. If PCIe Non-Flit Mode is the final negotiated protocol, it will use the CXL.io 68B Flit Mode formats as defined in the CXL 2.0 Specification.
3. "CXL 256B Flit Mode": This is a protocol parameter. This must be advertised if the Adapter and Protocol Layer support CXL 256B Flit Mode.
4. "PCIe Flit Mode": This is a protocol parameter. This must be advertised if the Adapter and Protocol Layer support PCIe Flit Mode.
5. "Streaming": This is a protocol parameter. This must be advertised if the Adapter and Protocol Layer support Streaming protocol in Raw Mode and this capability is enabled.
6. "Retry": This must be advertised if the Adapter supports Retry. The Link cannot be operational if the Adapter has determined Retry is needed, but "Retry" is not advertised or negotiated unless the Link is running in "Raw_Mode".
7. "Multi_Protocol_Enable": This must only be advertised if the Adapter is connected to multiple FDI instances corresponding to two sets of Protocol Layers. It must only be advertised if the Adapter (or SoC firmware in Stage 0 of Link Initialization) has determined that the UCIe Link must be operated in this mode. Both "Stack0_Enable" and "Stack1_Enable" must be 1b if this bit is advertised.
8. "Stack0_Enable": This must be advertised if Protocol Layer corresponding to Stack 0 exists and is enabled for operation with support for the advertised protocols.
9. "Stack1_Enable": This must be advertised if Protocol Layer corresponding to Stack 1 exists and is enabled for operation with support for the advertised protocols.
10. "CXL_LatOpt_Fmt5" : This must be advertised if the Adapter and Protocol Layer support Format 5 defined in section 3.2.4. The Protocol Layer does not take advantage of the spare bytes in this Flit format. This must not be advertised if CXL protocol and 256B Flit Mode of CXL are not supported or enabled.
11. "CXL_LatOpt_Fmt6": This must be advertised if the Adapter and Protocol Layer support Format 6 defined in section 3.2.4. The Protocol Layer is taking advantage of the spare bytes in this Flit format. This must not be advertised if CXL protocol and 256B Flit Mode of CXL are not supported or enabled.
12. "Retimer": This must be advertised if the Adapter of a UCIe Retimer is performing Parameter Exchanges with the UCIe Die within its package.
13. "Retimer_Credits": This is a 9-bit value advertising the total credits available for Retimer’s Receiver Buffer. Each credit corresponds to 256B data.
14. "DP": This is set by Downstream Ports to inform the remote Link partner that it is a Downstream Port. It is useful for Retimers to identify if they are connected to a Downstream Port UCIe die. It is currently only applicable for PCIe and CXL protocols, however Streaming Protocols are not precluded from utilizing this bit. This bit must be set to 0b if "Retimer" is set to 1b.

15. "UP": This is set by Upstream Ports to inform the remote Link partner that it is an Upstream Port. It is useful for Retimers to identify if they are connected to an Upstream Port UCIe die. It is currently only applicable for PCIe and CXL protocols, however Streaming Protocols are not precluded from utilizing this bit. This bit must be set to 0b if "Retimer" is set to 1b.

Once local capabilities are established, the Adapter sends the {AdvCap.Adapter} sideband message advertising its capabilities to the remote Link partner. If PCIe or CXL protocol support is going to be advertised, the Upstream Port (UP) Adapter must wait for the first message from the Downstream Port (DP) Adapter, review the capabilities advertised by DP and then send its own sideband message of advertised capabilities. UP is permitted to change its advertised capabilities based on DP capabilities. Once the downstream port receives the capability advertisement message from UP, it responds with the Finalized Configuration using {FinCap.Adapter} sideband message to UP as shown in Figure 24. Refer to section 6.1.2.3 to see the message format for the relevant sideband messages.

Finalized Configuration determination for Protocol parameters:

- If "68B Flit Mode" is advertised by both Link partners, it is set to 1b in the {FinCap.Adapter} message.If "CXL 256B Flit Mode" is advertised by both Link partners, it is set to 1b in the {FinCap.Adapter} message.
- If "PCIe Flit Mode" is advertised by both Link partners,, "PCIe Flit Mode" bit is set to 1b in the {FinCap.Adapter} message.
- If "Streaming" is advertised by both Link partners,"Streaming" is set to 1b, in the {FinCap.Adapter} message.

If "68B Flit Mode" or "CXL 256B Flit Mode" is set in the {FinCap.Adapter} message, there must be another handshake of Parameter Exchanges using the {AdvCap.CXL} and the {FinCap.CXL} messages to determine the details associated with this mode. This additional handshake is shown in Figure 25. The combination of {FinCap.CXL} and {FinCap.Adapter} determine the Protocol and Flit format. Refer to section 6.1.2.3 to see the message format for the relevant sideband messages. Refer to section 3.3 to see how Protocol and Flit Formats are determined.

Finalized Configuration determination for other parameters :

If "Raw_Mode" is advertised by both Link partners, "Raw_Mode" is set to 1b in the {FinCap.Adapter} message.

- If both Link partners advertised "Retry", Adapter Retry is enabled and "Retry" is set to 1b in the {FinCap.Adapter} message.
- If both Link partners advertised "Multi_Protcol_Enable", both Stack0 and Stack1 are enabled by the adapter, and all three parameters ("Multi_Protcol_Enable", "Stack0_Enable" and "Stack1_Enable") are set to 1b in the {FinCap.Adapter} message.
- If "Multi_Protocol_Enable" is not negotiated, then the lowest common denominator is used to determine if Stack0 or Stack1 is enabled, and the corresponding bit is set to 1b in the {FinCap.Adapter} message. If both Stack enables are advertised, then Stack0 is selected for operational mode and only Stack0_Enable is set to 1b in the {FinCap.Adapter} message.
- If CXL_LatOpt_Fmt5 is advertised by both, then it is set to 1b in the {FinCap.Adapter} message.
- If CXL_LatOpt_Fmt6 is advertised by both, then it is set to 1b in the {FinCap.Adapter} message.

Figure 24. Parameter exchange for Adapter Capabilities

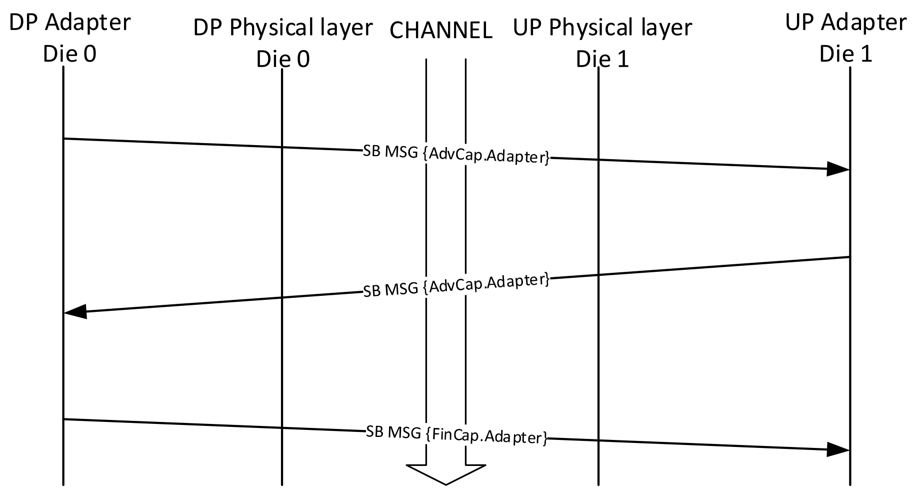
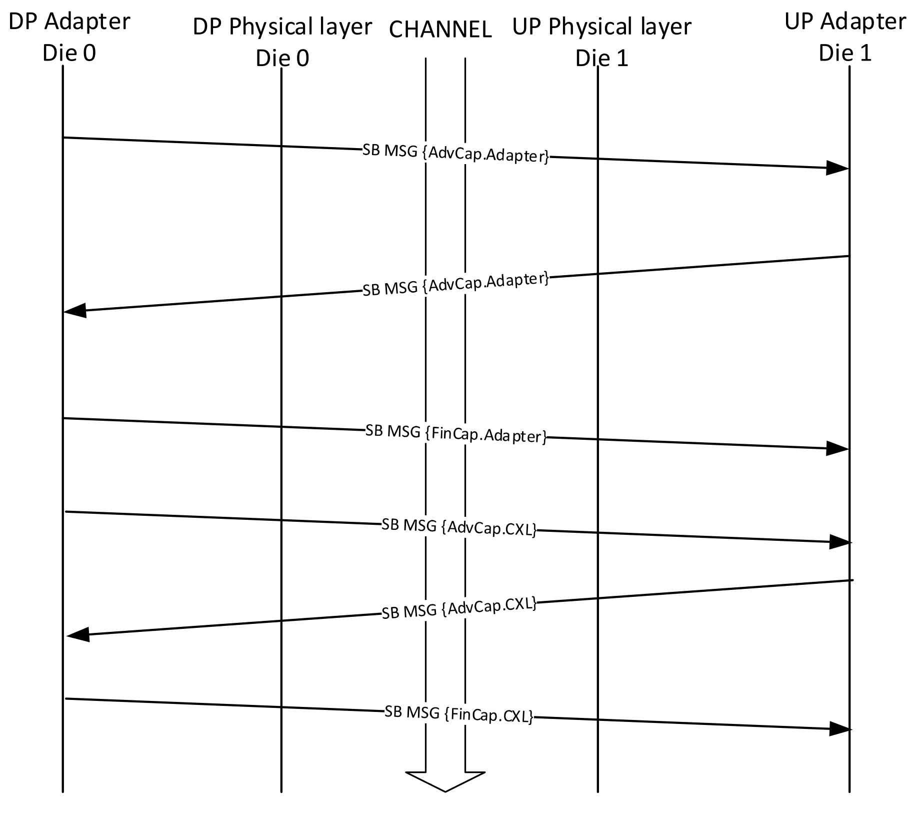


Figure 25. Parameter Exchange if "68B Flit Mode" or "CXL 256B Flit Mode" is 1b in {FinCap.Adapter}



If PCIe or CXL protocols are not advertised, and Streaming protocol is to be negotiated, there is no notion of DP and UP and each side independently advertises its capabilities.Additional Vendor Defined sideband messages are permitted to be exchanged to negotiate vendor-specific extensions, if required for Streaming Protocol. The Finalized Configuration is implicitly determined to be the Raw Mode if no vendor-specific extensions are negotiated. {FinCap.*} messages are not sent for Streaming Protocol. Adapter must determine vendor specific requirements a priori; for example, at design integration or during Stage 0 of Link bring up.

The Adapter must implement a timeout of 8ms for successful Parameter Exchange completion. The timer only increments while RDI is in Active state. The timer must reset if the Adapter receives a {AdvCap.*.Stall} or {FinCap.*.Stall} message from remote Link partner. UCIe Retimers must ensure they resolve the capability advertisement with remote Retimer partner (and merge with their own capabilities) before responding/initiating parameter exchanges with the UCIe die within its package. While resolution is in progress they must send the corresponding stall message once every 4ms to ensure there isn't a timeout on the UCIe die within its package.

3.1.1.3 Part 3: FDI bring up

Once Parameter Exchanges have successfully completed, the Adapter reflects the result to the Protocol Layers on FDI, and moves on to carry out the FDI bring up flow as defined in section 8.2.7. Once FDI is in Active state, it concludes Stage 3 of Link Initialization and protocol Flit transfer can begin. In cases where there are two protocol stacks negotiated for operation over the same UCIe Link, the FDI bring up flow must be performed independently for each protocol stack.

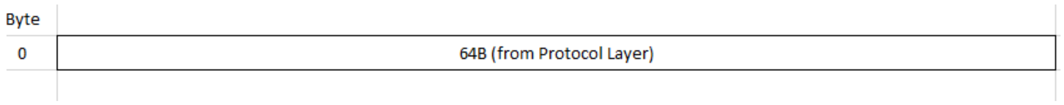
The data width on FDI is a function of the frequency of operation of the UCIe stack as well as the total bandwidth being transferred across the UCIe physical Link (which in turn depends on the number of Lanes and the speed at which the Lanes are operating). The data width on RDI is fixed to at least one byte per physical Lane per module that is controlled by the Adapter. The illustrations of the formats in this chapter are showing an example configuration of RDI mapped to a 64 Lane module of Advanced Package configuration on the Physical Layer of UCIe.

3.2 Modes of operation and Protocols

3.2.1 Raw Mode for all protocols

Raw Mode can only be used for scenarios where Retry support from the Adapter is not required. If Raw Mode is negotiated for CXL or PCIe protocols, then the Adapter transfers data from Protocol Layer to Physical Layer without any modification. Figure 26 shows an example of this for a 64B data path on FDI and RDI. This is identified as *Format 1* during parameter negotiation.

Figure 26. Format 1: Raw Mode



3.2.2 CXL 2.0 or "CXL 68B-Enhanced Flit Mode"

This Flit format is identified as *Format 2* or *Format 7* on UCIe. The support for this is mandatory when CXL 2.0 (*Format 2*) or "CXL 68B-Enhanced Flit Mode" (*Format 7*) protocols are supported.

The Protocol Layer sends 64B of protocol information. The Adapter adds a two byte prefix of Flit Header and a two byte suffix of CRC. Table 12 gives the Flit Header format for *Format 2* or *Format 7* when Retry from the Adapter is required; if Retry from the Adapter is not required, then the Flit Header format is provided in Table 11.

Even if Retry is not required, the Adapter still computes and drives CRC bytes - the Receiver is strongly recommended to treat a CRC error as an Uncorrectable Internal Error in this situation. CRC is computed over 66B (64B of data from the Protocol Layer and 2B of Flit Header from the Adapter). The same 128B CRC computation as previous formats is used, but bytes 127 to 67 are implicitly assumed to be 0 by the Transmitter and Receiver.

Retry is performed over this 68B Flit.

Table 11. Flit Header for Format 2 without Retry

Byte	Bit	Description
Byte 0	[7:6]	Protocol Identifier: 2’b00 : D2D Adapter Flit 2’b01 : CXL.io Flit 2’b10 : CXL.cachemem Flit 2’b11 : ARB/MUX Flit
	[5]	Stack Identifier: 1’b0 : Stack 0 1’b1 : Stack 1
	[4]	1’b0 : Regular Flit Header 1’b1 : Pause of Data Stream (PDS) Flit Header
	[3:0]	Reserved
Byte 1	[7]	1’b0 : Regular Flit Header 1’b1 : Pause of Data Stream (PDS) Flit Header
	[6:0]	Reserved

Table 12. Flit Header for Format 2

Byte	Bit	Description
Byte 0	[7:6]	Protocol Identifier: 2’b00 : D2D Adapter 2’b01 : CXL.io 2’b10 : CXL.cachemem 2’b11 : ARB/MUX
	[5]	Stack Identifier: 1’b0 : Stack 0 1’b1 : Stack 1
	[4]	1’b0 : Regular Flit Header 1’b1 : Pause of Data Stream (PDS) Flit Header
	[3:0]	The upper four bits of Sequence number "S" (i.e. S[7:4])
Byte 1	[7:6]	2’b00 : Regular Flit Header 2’b11 : Pause of Data Stream (PDS) Flit Header Other encodings are reserved
	[5:4]	Ack or Nak information 2’b00 : Explicit Sequence number "S" of Flit if not PDS, otherwise the bitwise inverted value of the Sequence number of the Flit 2’b01 : Ack. The Sequence number "S" carries the Ack’ed sequence number. 2’b10 : Nak. The Sequence number "S" carries 255 if N=1, otherwise it carries N-1, where N is the Nak’ed sequence number. 2’b11 : Reserved
	[3:0]	The lower four bits of Sequence number "S" (i.e. S[3:0]) Sequence number 0 is reserved and if present, it implies no Ack or Nak is sent.

Because of the four bytes added by D2D Adapter, the alignment of the Flit does not exactly match the number of Lanes of the physical Link (which are always in multiples of 16). This requires the Adapter to shift the Flits by four bytes for consecutive Flits. When the transmitting Adapter has no more Flits to send, it terminates the data stream with a Pause of Data Stream (PDS) token and two subsequent transfers of all 0 value data. The two subsequent transfers of all 0 data are present to give the Receiver at least a couple of 64B transfers to reset the receiving byte shifter. The PDS token and the 0 bytes following it must not be forwarded to the Protocol Layer. The PDS token is a variable size Flit that carries a two byte special Flit Header, and 0 bytes padded on the remaining bytes of RDI interface. The Transmitter of PDS drives the following on the Flit header:

- Bit [4] of Byte 0 as 1’b1
- Bit [7] of Byte 1 as 1’b1
- Bit [6] of Byte 1 as 1’b1
- Bit [5:4] of Byte 1 as 2’b00 and the sequence number[7:0] is the bit wise inverted value of the expected sequence number

The Receiver must interpret this Flit header as PDS if any two of the above four conditions are true. This guarantees that a PDS will be detected if there are less than three bit errors in the Flit Header. For three bit errors, it could result in a retry, but that will be handled seamlessly through the retry rules.

If a retry is triggered or if the RDI interface states goes through Retrain, that is an implicit PDS and the Transmitter must start the replayed Flit from fresh alignment. For retry and Retrain scenarios, the Receiver must also look for the expected sequence number in Byte 0 and Byte 1 of the received data bus.

Figure 27 shows this format. Figure 28 gives an example of PDS insertion.

Figure 27. Format 2: CXL 68B Flit Mode

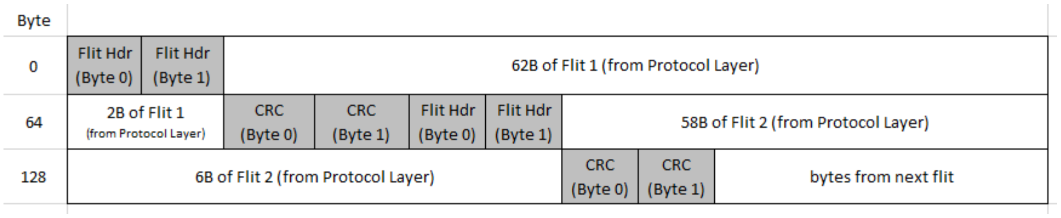
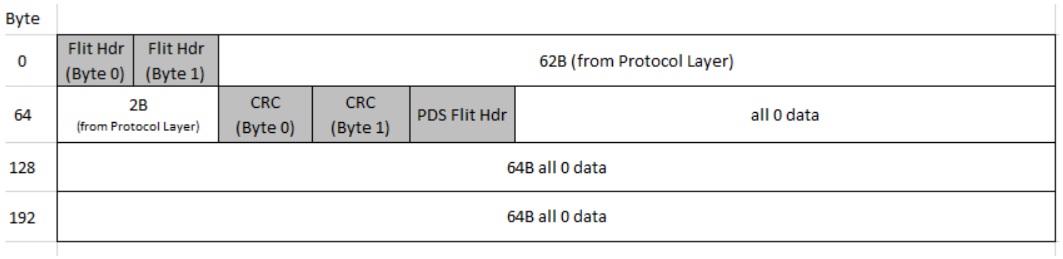


Figure 28. PDS example



3.2.3 PCIe 6.0 or "CXL 256B Flit Mode" with Standard 256B Flit

These are the Standard Flit formats defined in *PCI Express Base Specification Revision 6.0* for PCIe Flit Mode and *Compute Express Link 3.0 Specification* for "CXL 256B Flit Mode", and are identified as *Format 3* and *Format 4* respectively. Support for this is mandatory when PCIe 6.0 or "CXL 256B Flit Mode" protocols are negotiated. The Protocol Layer sends data in 256B Flits, but it drives 0 on the bytes reserved for the Adapter (shown in gray in Figure 29, Figure 30 and Figure 31). The 6B of DLP defined in the *PCI Express Base Specification* exist in *Format 3* and *Format 4* as well. However, since DLLPs are required to bypass the TX Retry buffer in PCIe and CXL.io protocols, the DLP bytes end up being unique since they are partially filled by the Protocol Layer and partially by the Adapter. DLP0 and DLP1 are replaced with the Flit Header for UCIe and are driven by UCIe Adapter. However, if the Flit carries a Flit Marker, the Protocol Layer must populate bit 4 of Flit Header byte 0 to 1b, as well as the relevant information in the Flit_Marker bits (these are driven as defined in the *PCI Express Base Specification*). Protocol Layer must also populate the Protocol Identifier bits in the Flit Header for the Flits it generates.

FDI provides a separate interface for DLLP transfer from the Protocol Layer to the Adapter and vice-versa. The Adapter is responsible for inserting DLLP into DLP bytes 2:5 if a Flit Marker is not present. The credit update information is transferred as regular Update_FC DLLPs over FDI from the Protocol Layer to the Adapter. The Adapter is also responsible for formating these updates as Optimized_Update_FC format when possible and driving them on the relevant DLP bytes. The Adapter is also responsible for adhering to all the DLLP rules defined for Flit Mode in the *PCI Express Base Specification.* On the receive path, the Adapter is responsible for extracting the DLLPs or Optimized_Update_FC from the Flit and driving it on the dedicated DLLP interface provided on FDI.

Two sets of CRC are computed (CRC0 and CRC1). The same 2B over 128B CRC computation as previous formats is used.

For PCIe: CRC0 is computed over the first 128B of the Flit (including the Flit Chunk 0 and Flit Chunk 1). CRC1 is computed over Flit Chunk 2 and Flit Chunk 3 of the Flit as well as the Flit Hdr and DLP bytes - and for CRC computation, the 114B is zero extended in MSB to 128B.

For CXL: CRC0 is computed over the first 128B of the Flit (including the Flit Hdr, Flit Chunk 0 and Flit Chunk 1). CRC1 is computed over Flit Chunk 2, Flit Chunk 3 and for CXL.io, DLP Bytes are including in the CRC computation as well - and for CRC computation, the 114B is zero extended in MSB to 128B.

If Retry is not required, the Adapter still computes and drives CRC bytes - the Receiver is strongly recommended to treat a CRC error as an Uncorrectable Internal Error (UIE) in this situation.

The Flit Header byte formats are shown in Table 14 when Retry is required, else it is as shown in Table 13.

Figure 29. Format 3: Standard 256B Flit Mode for PCIe 6.0

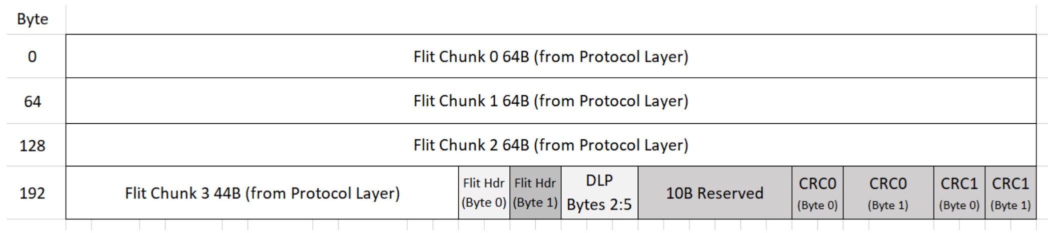

Byte
0
Flit Chunk 0 64B (from Protocol Layer)
64
Flit Chunk 1 64B (from Protocol Layer)
128
Flit Chunk 2 64B (from Protocol Layer)
192
Flit Chunk 3 44B (from Protocol Layer)
Flit Hdr (Byte 0)
Flit Hdr (Byte 1)
DLP Bytes 2:5
10B Reserved
CRC0 (Byte 0)
CRC0 (Byte 1)
CRC1 (Byte 0)
CRC1 (Byte 1)

Figure 30. Format 4: Standard 256B Flit Mode for CXL.cachemem

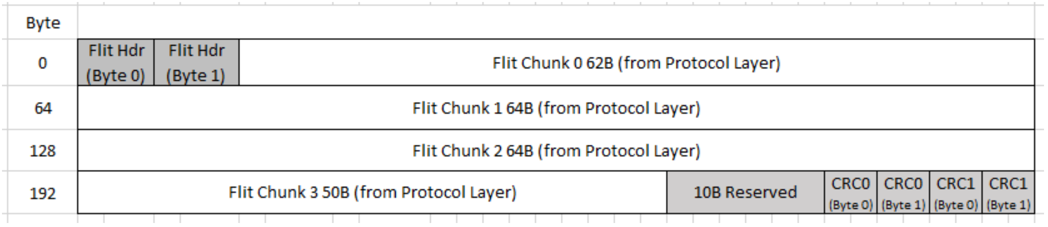

Byte
0
Flit Hdr (Byte 0)
Flit Hdr (Byte 1)
Flit Chunk 0 62B (from Protocol Layer)
64
Flit Chunk 1 64B (from Protocol Layer)
128
Flit Chunk 2 64B (from Protocol Layer)
192
Flit Chunk 3 50B (from Protocol Layer)
10B Reserved
CRC0 (Byte 0)
CRC0 (Byte 1)
CRC1 (Byte 0)
CRC1 (Byte 1)

Figure 31. Format 4: Standard 256B Flit Mode for CXL.io

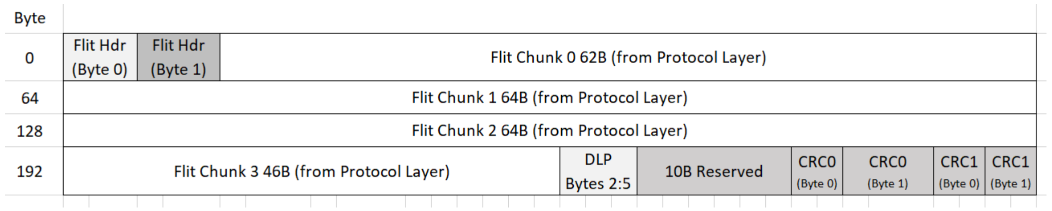

Byte
0
Flit Hdr (Byte 0)
Flit Hdr (Byte 1)
Flit Chunk 0 62B (from Protocol Layer)
64
Flit Chunk 1 64B (from Protocol Layer)
128
Flit Chunk 2 64B (from Protocol Layer)
192
Flit Chunk 3 46B (from Protocol Layer)
DLP Bytes 2:5
10B Reserved
CRC0 (Byte 0)
CRC0 (Byte 1)
CRC1 (Byte 0)
CRC1 (Byte 1)

Table 13. Flit Header for Format 3 or Format 4 without Retry

Byte	Bit	Description "CXL 256B Flit Mode"	Description PCIe 6.0
Byte 0	[7:6]	Protocol Identifier: 2'b00 : D2D Adapter/CXL.io NOP 2'b01 : CXL.io 2'b10 : CXL.cachemem 2'b11 : ARB/MUX	Protocol Identifier: 2'b00 : D2D Adapter/PCIe NOP 2'b01 : PCIe
	[5]	Stack Identifier: 1'b0 : Stack 0 1'b1 : Stack 1	
	[4]	Reserved for CXL.cachemem and Streaming Protocols For CXL.io or PCIe 6.0: 0b DLLP Payload in DLP 2..5 1b Optimized_Update_FC or Flit_Marker in DLP2..5	
	[3:0]	Reserved	
Byte 1	[7:0]	Reserved	

Table 14. Flit Header for Format 3 or Format 4 with Retry

Byte	Bit	Description CXL 3.0	Description PCIe 6.0
Byte 0	[7:6]	Protocol Identifier: 2'b00 : D2D Adapter/CXL.io NOP 2'b01 : CXL.io 2'b10 : CXL.cachemem 2'b11 : ARB/MUX	Protocol Identifier: 2'b00 : D2D Adapter/PCIe NOP 2'b01 : PCIe
	[5]	Stack Identifier: 1'b0 : Stack 0 1'b1 : Stack 1	
	[4]	Reserved for CXL.cachemem and Streaming Protocols For CXL.io or PCIe 6.0: 0b DLLP Payload in DLP 2..5 1b Optimized_Update_FC or Flit_Marker in DLP2..5	
	[3:0]	The upper four bits of Sequence number "S" (i.e. S[7:4])	
Byte 1	[7:6]	Reserved	
	[5:4]	Ack or Nak information 2'b00 : Explicit Sequence number "S" of the current Flit is present 2'b01 : Ack. The sequence number "S" carries the Ack'ed sequence number. 2'b10 : Nak. The sequence number "S" carries 255 if N=1, otherwise it carries N-1; where N is the Nak'ed sequence number. 2'b11 : Reserved	
	[3:0]	The lower four bits of Sequence number "S" (i.e. S[3:0]) Sequence number 0 is reserved and if present, it implies no Ack or Nak is sent.	

3.2.4 "CXL 256B Flit Mode" with Latency-Optimized 256B Flit

Two Flit Formats are defined for this mode: *Format 5* and *Format 6*. It is strongly recommended that UCIe implementations support these to get the best latency benefits.

Both formats look the same from the Adapter perspective, the only difference is whether the Protocol Layer is filling in the optional bytes of protocol information or not. *Format 5* is when the Protocol Layer is not filling in the optional bytes, whereas *Format 6* is when the Protocol Layer is filling in the optional bytes.

Two sets of CRC are computed. The same 2B over 128B CRC computation as previous formats is used. CRC0 is over the first 126B of the Flit (including the Flit Header, Chunk 0,Chunk1 and the optional bytes. Reserved bits are assigned 0 for CRC computation) - for CRC computation, the 126B is zero extended in MSB to 128B. CRC1 is over Chunk 2 and Chunk 3 of the Flit (the optional bytes for Format 6 are included in CRC computation as well) - and for CRC computation, the 122B is 0 extended in MSB to 128B. If Retry is not required, the Adapter still computes and drives CRC bytes - the Receiver is strongly recommended to treat a CRC error as UIE in this situation.

Figure 32. Format 5 and 6: Latency Optimized 256B Flit Mode for CXL.io

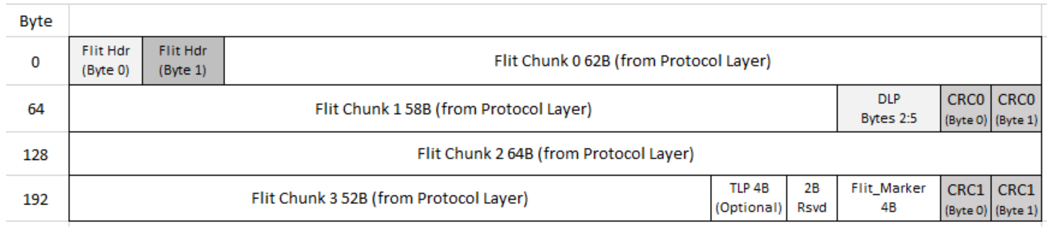
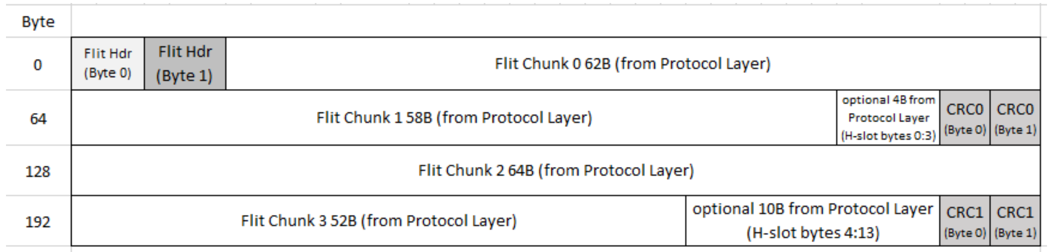

Byte
0
Flit Hdr (Byte 0)
Flit Hdr (Byte 1)
Flit Chunk 0 62B (from Protocol Layer)
64
Flit Chunk 1 58B (from Protocol Layer)
DLP Bytes 2:5
CRC0 (Byte 0)
CRC0 (Byte 1)
128
Flit Chunk 2 64B (from Protocol Layer)
192
Flit Chunk 3 52B (from Protocol Layer)
TLP 4B (Optional)
2B Rsvd
Flit_Marker 4B
CRC1 (Byte 0)
CRC1 (Byte 1)

Figure 33. Format 5 and 6: Latency Optimized 256B Flit Mode for CXL.cachemem


Byte
0
Flit Hdr (Byte 0)
Flit Hdr (Byte 1)
Flit Chunk 0 62B (from Protocol Layer)
64
Flit Chunk 1 58B (from Protocol Layer)
optional 4B from Protocol Layer (H-slot bytes 0:3)
CRC0 (Byte 0)
CRC0 (Byte 1)
128
Flit Chunk 2 64B (from Protocol Layer)
192
Flit Chunk 3 52B (from Protocol Layer)
optional 10B from Protocol Layer (H-slot bytes 4:13)
CRC1 (Byte 0)
CRC1 (Byte 1)

The Flit Header byte formats are the same as Table 14 when Retry is required, else it is the same as Table 13. The DLP rules are also the same as defined in section 3.2.3 for CXL protocol, except that Flit_Marker/Optimized_Update_FC has dedicated space in the Flit, i.e., bit[4] of Byte 0 corresponds to the Flit_Marker bytes, and not the DLP bytes. If Optimized_Uptade_FC is sent, the DLP bytes 2:5 shown in Figure 32 must be reserved.

3.3 Decision table for protocol and flit format

Table 16 shows the truth table for deciding which Flit format to operate in. Table 15 shows the truth table for determining the protocol for operation. Once the protocol and Flit format have been negotiated during initial Link bring up, they cannot be changed until the UCIe Physical Layer transitions to Reset state.

If a valid Protocol and Flit Format are not negotiated, then the Adapter takes the Link down and escalates the error if applicable.

Table 15. Truth Table for determining Protocol

{FinCap.Adapter} bits				{FinCap.CXL} bits		
68B Flit Mode	**CXL 256B Flit Mode**	**PCIe Flit Mode**	**Streaming**	**PCIe**	**CXL.io**	**Protocol**
1	1	1	x	0	1[1]	CXL[2]
1	0	1	x	1[3]	0	PCIe
1	0	0	x	0	1	CXL[2]
1	0	0	x	1	0	PCIe[4]
N/A	N/A	N/A	N/A	N/A	N/A	Streaming[5]

1. CXL.io capable/enable must be 1b if CXL 256B Flit Mode is negotiated.
2. For CXL protocol, the specific combination of Single Protocol vs Type 1 vs Type 2 vs Type 3 is determined using the CXL.cache and CXL.mem capable/enable bits in addition to the CXL.io capable/enable bit in {FinCap.CXL}. The rules for that follow CXL Specification.
3. PCIe capable/enable must be 1b if PCIe Flit Mode is 1b but CXL 256B Flit Mode is 0b
4. If Protocol is PCIe, and 68B Flit Mode is use, it corresponds to PCIe non-Flit mode protocol.
5. No {FinCap.*} message is sent for Streaming protocol negotiation, it is the negotiated protocol if PCIe or CXL are not negotiated, but Streaming protocol is advertised.

Table 16. Truth Table for determining Flit Format

{FinCap.Adapter} bits						{FinCap.CXL} bit	
Raw_Mode	**68B Flit Mode**	**CXL 256B Flit Mode**	**PCIe Flit Mode**	**CXL_LatOpt_Fmt5**	**CXL_LatOpt_Fmt6**	**68B-Enhanced Flit**	**Flit Format**
1	x	x	x	x	x	x	Format 1 : Raw Mode
0	x	1	x	0	0	x	Format 4: Standard 256B Flit Mode for CXL
0	x	1	x	x	1	x	Format 6: Latency Optimized Mode for CXL
0	x	1	x	1	0	x	Format 5: Latency Optimized Mode for CXL
0	x	0	1	x	x	x	Format 3: Standard 256B Flit Mode for PCIe
0	1	0	0	x	x	0	Format 2: CXL 2.0 68B Flit Mode
0	1	0	0	x	x	1	Format 7: CXL 68B Enhanced Flit Mode

3.4 State Machine Hierarchy

UCIe has a hierarchical approach to Link state management in order to have well-defined functionality partitioning between the different layers and also enabling common state transitions or sequencing at FDI and RDI.

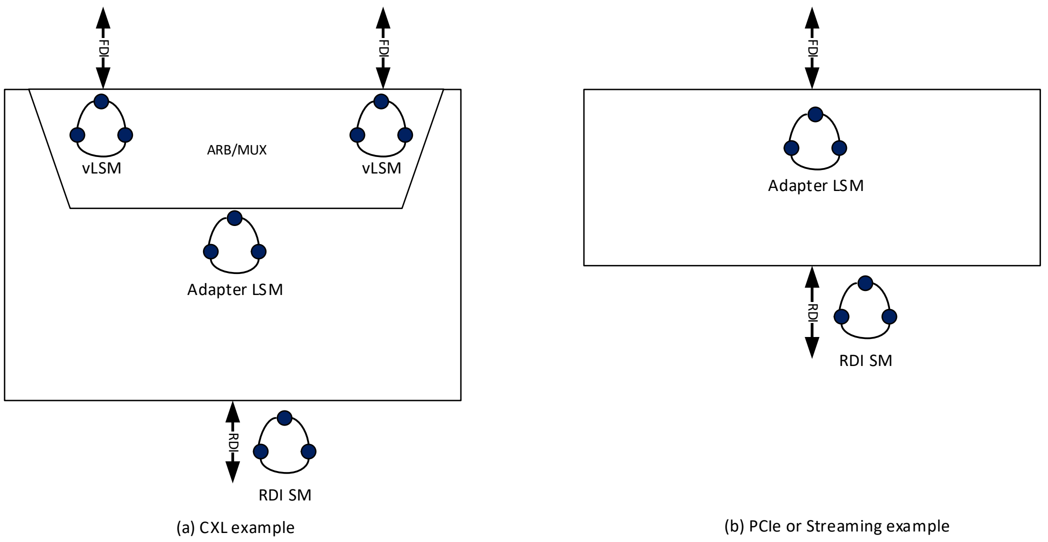
Figure 34 shows examples of state machine hierarchy for different configurations. For CXL, the ARB/MUX vLSMs are exposed on FDI `pl_state_sts`. The Adapter LSM is used to coordinate Link states with remote Link Partner and is required for all configurations.

Each protocol stack has its corresponding Adapter LSM. For PCIe or Streaming protocols, the Adapter LSM is exposed on FDI **`pl_state_sts`**.

Additionally the RDI state machine (SM) is used to abstract the Physical Layer states for the upper layers. The Adapter data path and RDI data width can be extended for multi-module configurations; however, a single RDI state machine is present for all of the modules. The Multi-module PHY Logic creates the abstraction and coordinates between the RDI state and individual modules. The following rules apply:

- vLSM state transitions are co-ordinated with remote Link partner using ALMPs on mainband data path. The rules for state transitions follow the CXL 256B Flit Mode rules.
- Adapter LSM state transitions are co-ordinated with remote Link partner using {LinkMgmt.Adapter*} sideband messages. These messages are originated and received by the D2D Adapter.
- RDI SM state transitions are co-ordinated with the remote Link partner using {LinkMgmt.RDI*} sideband messages. These messages are originated and received by the Physical Layer.

Figure 34. State Machine Hierarchy examples



General rules for State transition hierarchy are captured below. For specific sequencing, please refer to the rules outlined in Chapter 8.0.

- Active State transitions: RDI SM must be in Active before Adapter LSM can begin negotiation to transition to Active. Adapter LSM must be in Active before vLSMs can begin negotiations to transition to Active.
- Retrain State transitions: RDI SM must be in Retrain before propagating Retrain to Adapter LSMs. If RDI SM is in Retrain, Retrain must be propagated to all Adapter LSMs that are in Active state. Adapter must not request Retrain exit on RDI before all the relevant Adapter LSMs have transitioned to Retrain.
- PM State transitions (both L1 and L2): Both CXL.io and CXL.cachemem vLSMs (if CXL), must transition to PM before the corresponding Adapter LSM can transition to PM. All Adapter LSMs (if multi-protocol stacks are enabled) must be in PM before RDI SM is transitioned to PM.

- LinkError State transitions: RDI SM must be in LinkError before Adapter LSM can transition to LinkError. RDI SMs coordinate LinkError transition with remote Link partner using sideband, and each RDI SM propagates LinkError to all enabled Adapter LSMs. Adapter LSM must be in LinkError before propagating LinkError to both vLSMs if CXL. LinkError transition takes priority over LinkReset or Disabled transitions. Adapter must not request LinkError exit on RDI before all the relevant Adapter LSMs and CXL vLSMs have transitioned to LinkError.
- LinkReset or Disabled State transitions: Adapter LSM negotiates LinkReset or Disabled transition with its remote Link partner using sideband messages. LinkReset or Disabled is propagated to RDI SM only if all the Adapter LSMs associated with it transition to LinkReset or Disabled. Disabled transition takes priority over LinkReset transition. If RDI SM moves to LinkReset or Disabled, it must be propagated to all Adapter LSMs. If Adapter LSM moves to LinkReset or Disabled, it must propagate it to both vLSMs for CXL protocol.

For UCIe Retimers, it the responsibility of the Retimer die to negotiate state transitions with remoter Retimer partner and make sure the different UCIe die are in sync and do not timeout waiting for a response. As an example, referring to Figure 13, if UCIe Die 0 sends an Active Request message for the Adapter LSM to UCIe Retimer 0, UCIe Retimer 0 must resolve with UCIe Retimer 1 that an Active Request message has been forwarded to UCIe Die 1 and that UCIe Die 1 has responded with an Active Status message before responding to UCIe Die 0 with an Active Status message. Along the same lines, the Off Package Interconnect cannot be taken to a low power state unless all the relevant states on UCIe Die 0 AND UCIe Die 1 have reached the low power state. UCIe Retimers are must respond with "Stall" encoding every 4ms while completing resolution with the remote Retimer partner.

3.5 Power Management Link States

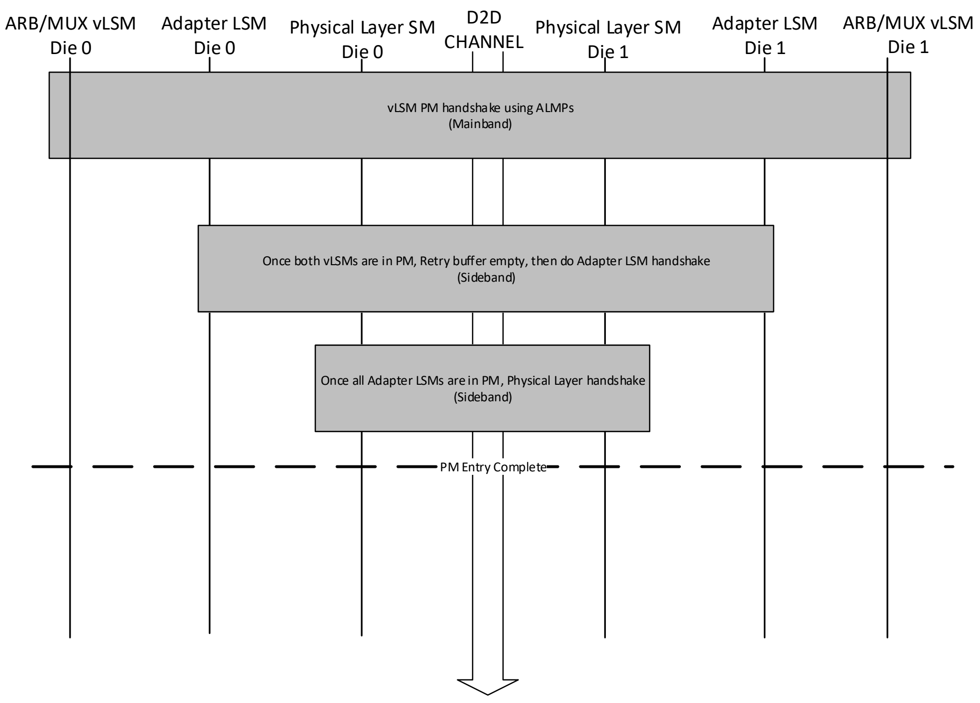
Power management states are mandatory for PCIe and CXL protocols. FDI supports L1 and L2 power states which follow the handshake rules and state transitions of CXL 256B Flit Mode. RDI supports L1 and L2 on the interfaces for Physical Layer to perform power management optimizations, however the Physical Layer is permitted to map both L1 and L2 to a common state internally. These together allow for global clock gating and enable system level flows like Package-Level Idle (C-states). Other Protocols are permitted to disable PM flows through always sending a PMNAK for a PM request from remote Link partner.

Power management state entry follows the following steps in sequence:

1. Protocol Layer PM entry request: FDI defines a common flow for PM entry request at the interface that is based on Link idle time. All protocols using UCIe must follow that flow when PM needs to be supported. For CXL protocol, D2D Adapter implements the ARB/MUX functionality and follows the handshakes defined in Compute Express Link 3.0 Specification (corresponding to the "CXL 256B Flit Mode", since all ALMPs also go through the Retry buffer in UCIe). Even CXL 2.0 and "CXL 68B-Enhanced Flit Mode" over UCIe use the "CXL 256B Flit Mode" ALMP formats and flows (but the Flit is truncated to 64B for them). Both L1 and L2 map to a common PM state on RDI. For PCIe protocol in UCIe Flit Mode, PM DLLP handshakes are NOT used. Protocol Layer requests PM entry on FDI based on Link idle time. The specific algorithm and hysteresis for determining Link idle time is implementation specific.
2. Die-to-Die Adapter Link State Machine PM entry. The PM transition for this is coordinated over sideband with remote Link partner. In scenarios where the Adapter is multiplexing between two protocol stacks, each stack’s Link State Machine must transition to PM independently.
3. PM entry on RDI interface. Once all the Adapter’s LSMs are in a PM state, it initiates PM entry on the RDI interface as defined in section 8.2.8.

4. Physical Layer moves to deeper PM state and takes the necessary actions for power management. Note that the sideband interface must remain active, since PM exit is initiated using that.

Figure 35. Example of hierarchical PM entry for CXL


ARB/MUX vLSM Die 0
Adapter LSM Die 0
Physical Layer SM Die 0
D2D CHANNEL
Physical Layer SM Die 1
Adapter LSM Die 1
ARB/MUX vLSM Die 1
vLSM PM handshake using ALMPs (Mainband)
Once both vLSMs are in PM, Retry buffer empty, then do Adapter LSM handshake (Sideband)
Once all Adapter LSMs are in PM, Physical Layer handshake (Sideband)
PM Entry Complete

PM exit follows the reverse sequence of wake up as mentioned below:

1. Active request from Protocol Layer is transmitted across the FDI and RDI interfaces to the local Physical Layer.
2. The Physical Layer uses sideband to coordinate wake up and retraining of the physical Link.
3. Once the physical Link is retrained, the RDI interface is in Active state on both sides, and the Adapter LSM PM exit is triggered from both sides (coordinated via sideband messages between Adapters as outlined in the FDI PM flow). For PCIe or Streaming protocol scenarios, this also transitions the Protocol Layer to Active state on FDI.
4. For CXL protocol, this step is followed by ALMP exchanges to bring the required protocol to Active state and then protocol Flit transfer can begin.

3.6 CRC Computation

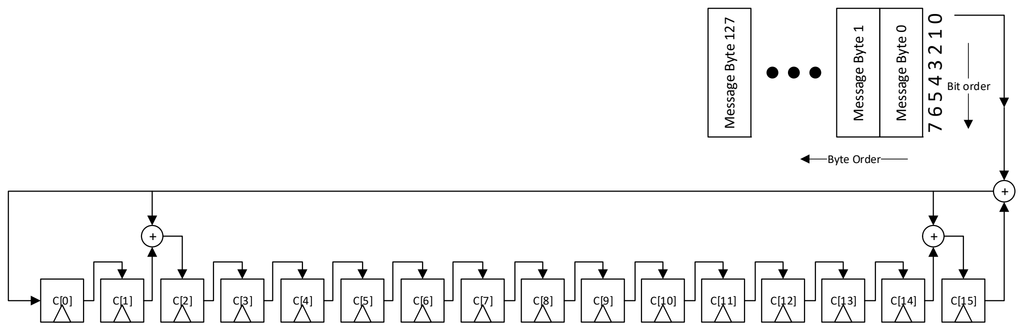
The CRC generator polynomial is $(x+1)*(x^{15} + x + 1) = x^{16} + x^{15} + x^2 + 1$. This gives a 3-bit detection guarantee for random bit errors: 2 bit detection guarantee is because of the primitive polynomial $(x^{15} + x + 1)$, and 1 additional bit error detection guarantee is provided by making it odd parity because of the $(x+1)$ term in the polynomial.

The CRC is always computed over 128 Bytes of the message. For smaller messages, the message is zero extended in the MSB.

The initial value of CRC bits for CRC LFSR computation is 0000h. The CRC calculation starts with bit 0 of byte 0 of the message, and proceeds from bit 0 to bit 7 of each byte as shown in Figure 36. In the figure, C[15] is bit 7 of CRC Byte 1, C[14] is bit 6 of CRC Byte 1 and so on; C[7] is bit 7 of CRC Byte 0, C[6] is bit 6 of CRC Byte 0 and so on.

The Verilog code for CRC code generation is provided in Appendix B and that must be used as the standard for implementing the CRC during encode or decode. The code is provided for the Transmit side. It takes 1024 bits (bit 1023 is bit 7 of message Byte 127, 1022 is bit 6 of message byte 127 and so on; bit 1015 is bit 7 of message Byte 126 and so on until bit 0 is bit 0 of message Byte 0) as an input message and outputs 16 bits of CRC. On the Receiver, the CRC is computed using the received Flit bytes with appropriate zero padding in the MSB to form a 128B message. If the received CRC does not match the computed CRC, the flit is declared Invalid and a replay must be requested.

Figure 36. Diagram of CRC calculation



3.7 Retry Rules

For Link speeds greater than 8GT/s, Retry must be supported in the Adapter, unless the only mode of operation is Raw Mode. If Retry is not supported by the Adapter, Link speeds greater than 8GT/s must NOT be advertised by the Physical Layer during Link Training, unless the mode of operation is Raw Mode. Once Retry has been negotiated during initial Link bring up, it cannot be disabled even if Link speed degrades during runtime. It can only be re-negotiated at the next Link bring up (i.e. RDI moves to Reset).

The Retry scheme on UCIe is a simplified version of the Retry mechanism for Flit Mode defined in *PCI Express Base Specification Revision 6.0*. The rules that do not apply and the corresponding parameter changes are enumerated here:

- Selective Nak and associated rules are not applicable and must not be implemented. RX Retry Buffer related rules are also not applicable and must not be implemented.
- Throughout the duration of Link operation, Explicit Sequence number Flits and Ack/Nak Flits alternate. It is permitted to send consecutive Explicit Sequence number Flits if there are no Ack/Nak Flits to send (or if following the rules of consecutive explicit number Flits).

- All 10-bit counters are replaced with 8-bit counters, and the maximum allowed sequence number is 255 (hence 1023 in all calculations and initial value is replaced by 255).
- REPLAY_TIMEOUT_FLIT_COUNT is a 9-bit counter which saturates to 1FFh.
- NAK_WITHDRAWAL_ALLOWED is always set to 0b.
- IDLE Flit Handshake Phase is not applicable. This is because the transition to Link Active (equivalent to LTSSM being in L0 for PCIe) is managed via handshakes on sideband, and there is no requirement for IDLE Flits to be exchanged.
- Sequence Number Handshake Phase timeout and exit to Link Retrain is 128 Flits transmitted without exiting Sequence Number Handshake Phase
- "Prior Flit was Payload" is always set to 1b. This bit does not exist in the Flit Header, and so from Retry perspective, implementations must assume it is always set to 1.

3.8 Runtime Link Testing using Parity

UCIe defines a mechanism to detect Link health during runtime by periodically injecting parity bytes in the middle of the data stream when this mechanism is enabled. The receiver checks and logs parity errors for the associated parity bytes.

When this mechanism is enabled, the Adapter inserts 64*N Bytes every 256*256*N Bytes of data, where N is obtained from the Error and Link Testing Control register (Field name: Number of 64 Byte Inserts). Software sets this based on the current Link width of operation, including the total number of Active modules interfaced to the Adapter. Only bit 0 of the inserted has the parity information which is computed as follows:

ParityByte X, bit 0 = ^((DataByte [X]) ^ (DataByte [X + 64*N]) ^(DataByte [X + 128*N])^....^(DataByte [X + (256*256*N - 64*N)]))

The remaining 7 bits of the inserted byte are Reserved.

The Transmitter and Receiver in the Adapter must keep track of the number of data bytes elapsed to compute or check the parity information. If the RDI state moves away from Active state, the data count and parity is reset, and both sides must renegotiate the enabling of the Parity insertion before next entry to Active from Retrain (if the mechanism is still enabled in the Error and Link Testing Control Register).

This mechanism can only be enabled by Software monitoring the Link, when it writes to an enable bit in registers in both the Adapters across a UCIe Link. Software must trigger UCIe Link Retrain after writing to the enable bit on both the Adapters. The Adapters exchange sideband messages while the Adapter LSMs are in Retrain to ensure the remote Link partner’s receiver is prepared to receive the extra parity bytes in the data stream once the states transition to Active. The Adapter must not request Retrain exit to local RDI until the Parity Feature exchanges are completed. It is permitted to enable it during Initial Link bring up, by using sideband to access to remote Link partner’s registers or other implementation specific means, however software must trigger Link Retrain for the feature to take effect.

Adapter sends a {ParityFeature.Req} sideband message to remote Link Partner if its Transmitter is enabled to send parity bytes. Remote Adapter responds with a {ParityFeature.Ack} sideband message if its receiver is enabled and ready to accept parity bytes. Figure 37 shows an example of a successful negotiation. If Die 0 Adapter Transmitter is enabled to insert parity bytes, it must send a {ParityFeature.Req} from Die 0 to Die 1.

Adapter responds with a {ParityFeature.Nak} if it is not ready to accept parity bytes, or if the feature has not been enabled for it yet. The requesting Adapter must log the Nak in a status register so that Software can determine that a Nak had occurred. Figure 38 shows an example of an unsuccessful negotiation.

Note: The Adapters are permitted to transition to a higher latency data path if the Parity Feature is enabled. The explicit Ack/Nak handshake is provided to ensure both sides have sufficient time to transition to alternate data path for this mechanism.

The Parity bytes do not consume Retimer receiver buffer credits. The Retimer receiver must not write the Parity bytes into its receiver buffer or forward these to remote Retimer partner. This mechanism is to help characterize local UCIe Links only.

Figure 37. Successful Parity Feature negotiation between Die 1 Tx and Die 0 Rx

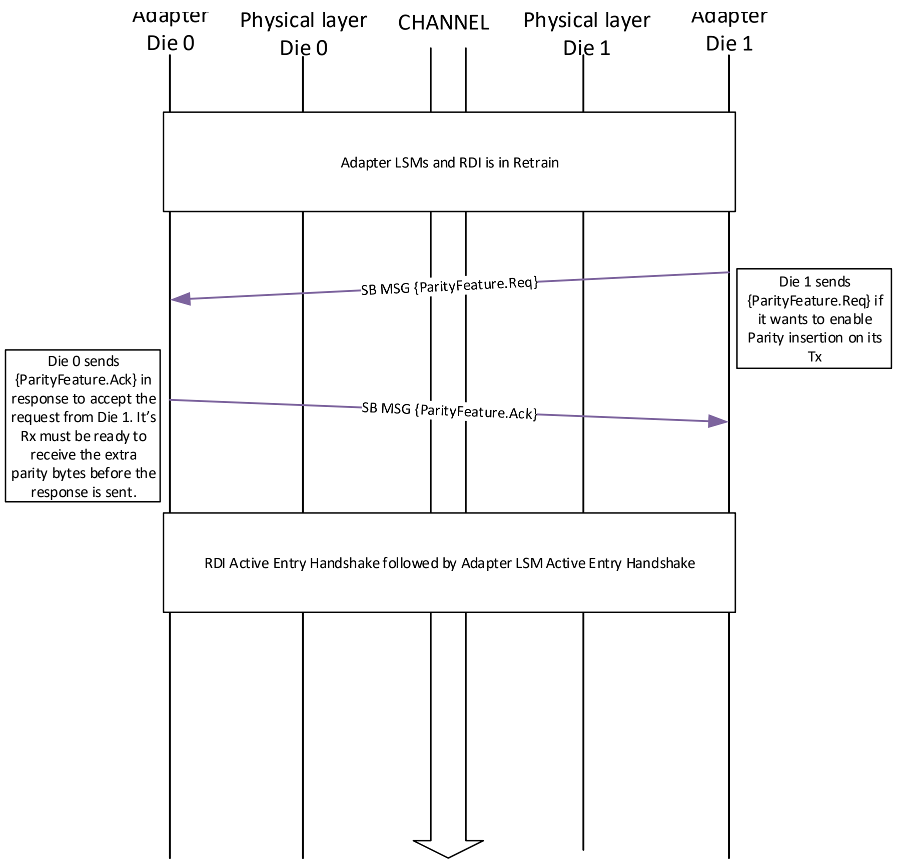
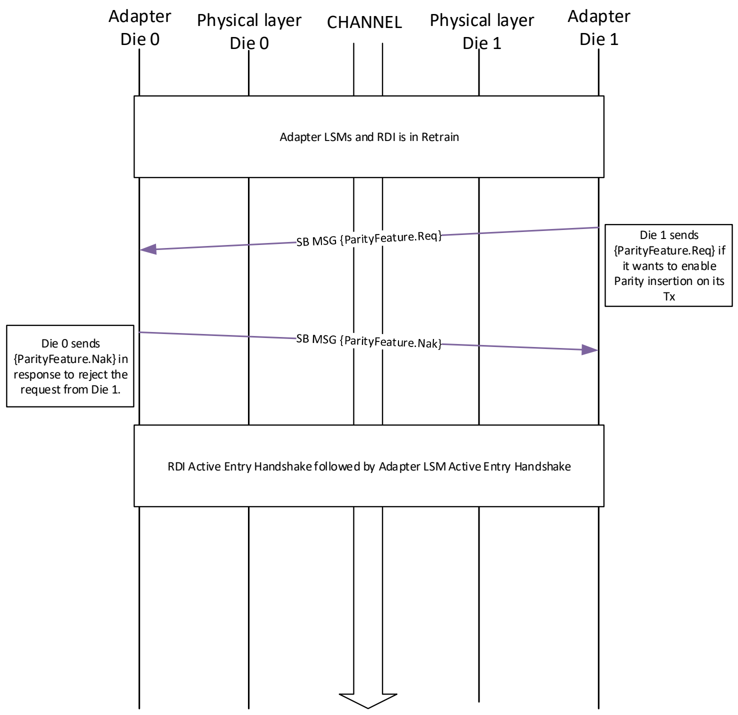

Adapter Die 0
Physical layer Die 0
CHANNEL
Physical layer Die 1
Adapter Die 1
Adapter LSMs and RDI is in Retrain
SB MSG {ParityFeature.Req}
Die 1 sends {ParityFeature.Req} if it wants to enable Parity insertion on its Tx
Die 0 sends {ParityFeature.Ack} in response to accept the request from Die 1. It's Rx must be ready to receive the extra parity bytes before the response is sent.
SB MSG {ParityFeature.Ack}
RDI Active Entry Handshake followed by Adapter LSM Active Entry Handshake

Figure 38. Unsuccessful Parity Feature negotiation between Die 1 Tx and Die 0 Rx


Adapter Die 0
Physical layer Die 0
CHANNEL
Physical layer Die 1
Adapter Die 1
Adapter LSMs and RDI is in Retrain
SB MSG {ParityFeature.Req}
Die 1 sends {ParityFeature.Req} if it wants to enable Parity insertion on its Tx
SB MSG {ParityFeature.Nak}
Die 0 sends {ParityFeature.Nak} in response to reject the request from Die 1.
RDI Active Entry Handshake followed by Adapter LSM Active Entry Handshake

4.0 Logical Physical Layer

The Logical PHY comprehends the following functions:

- Link initialization, training and power management states
- Byte to Lane mapping for data transmission over Lanes
- Interconnect redundancy remapping (when required)
- Transmitting and receiving sideband messages
- Scrambling and training pattern generation
- Lane reversal
- Width degradation (when applicable)

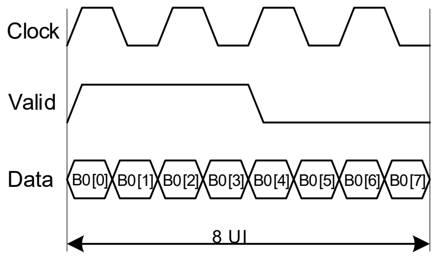
4.1 Data and Sideband Transmission Flow

This specification defines clock, valid and Data to send and receive data over the physical Lanes. The transmitted data is framed by the valid signal.

4.1.1 Byte to Lane Mapping

Data packets are transmitted in Bytes. Within each Byte, bit[0] is transmitted first. Figure 39 shows an example of bit arrangement within one byte transmission over Lane0.

Figure 39. Bit arrangement within a byte transfer



Each Byte is transmitted on a separate Lane. Byte 0 (B0) is transmitted on Lane 0, Byte 1 is transmitted on Lane 1 and so on.

Figure 40 shows an example of a CXL Latency Optimized Flit transmitted over a x64 interface (one Advanced Package module or four Standard Package modules). If the IO width changes to x32 or x16 interface (Standard Package), transmission of one Byte per Lane is preserved as shown in Figure 41 and Figure 42 respectively.

Figure 40. Byte map for x64 interface

UI \ LANE	0	1	2	3	4	5	6	7	...	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0 - 7	B00	B01	B02	B03	B04	B05	B06	B07	...	B48	B49	B50	B51	B52	B53	B54	B55	B56	B57	B58	B59	B60	B61	B62	B63
8 - 15	B64	B65	B66	B67	B68	B69	B70	B71	...	B112	B113	B114	B115	B116	B117	B118	B119	B120	B121	B122	B123	B124	B125	B126	B127
16 - 23	B128	B129	B130	B131	B132	B133	B134	B135	...	B176	B177	B178	B179	B180	B181	B182	B183	B184	B185	B186	B187	B188	B189	B190	B191
24 - 31	B192	B193	B194	B195	B196	B197	B198	B199	...	B240	B241	B242	B243	B244	B245	B246	B247	B248	B249	B250	B251	B252	B253	B254	B255

Figure 41. Byte map for x32 interface

Lane UI	0	1	2	3	4	5	6	7	...	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0 - 7	B0	B1	B2	B3	B4	B5	B6	B7	...	B16	B17	B18	B19	B20	B21	B22	B23	B24	B25	B26	B27	B28	B29	B30	B31
8 - 15	B32	B33	B34	B35	B36	B37	B38	B39	...	B48	B49	B50	B51	B52	B53	B54	B55	B56	B57	B58	B59	B60	B61	B62	B63
16 - 23	B64	B65	B66	B67	B68	B69	B70	B71	...	B80	B81	B82	B83	B84	B85	B86	B87	B88	B89	B90	B91	B92	B93	B94	B95
24 - 31	B96	B97	B98	B99	B100	B101	B102	B103	...	B112	B113	B114	B115	B116	B117	B118	B119	B120	B121	B122	B123	B124	B125	B126	B127
32 - 39	B128	B129	B130	B131	B132	B133	B134	B135	...	B144	B145	B146	B147	B148	B149	B150	B151	B152	B153	B154	B155	B156	B157	B158	B159
40 - 47	B160	B161	B162	B163	B164	B165	B166	B167	...	B176	B177	B178	B179	B180	B181	B182	B183	B184	B185	B186	B187	B188	B189	B190	B191
48 - 55	B192	B193	B194	B195	B196	B197	B198	B199	...	B208	B209	B210	B211	B212	B213	B214	B215	B216	B217	B218	B219	B220	B221	B222	B223
56 - 63	B224	B225	B226	B227	B228	B229	B230	B231	...	B240	B241	B242	B243	B244	B245	B246	B247	B248	B249	B250	B251	B252	B253	B254	B255

Figure 42. Byte map for x16 interface

Lane UI	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 - 7	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15
8 - 15	B16	B17	B18	B19	B20	B21	B22	B23	B24	B25	B26	B27	B28	B29	B30	B31
16 - 23	B32	B33	B34	B35	B36	B37	B38	B39	B40	B41	B42	B43	B44	B45	B46	B47
24 - 31	B48	B49	B50	B51	B52	B53	B54	B55	B56	B57	B58	B59	B60	B61	B62	B63
32 - 39	B64	B65	B66	B67	B68	B69	B70	B71	B72	B73	B74	B75	B76	B77	B78	B79
40 - 47	B80	B81	B82	B83	B84	B85	B86	B87	B88	B89	B90	B91	B92	B93	B94	B95
48 - 55	B96	B97	B98	B99	B100	B101	B102	B103	B104	B105	B106	B107	B108	B109	B110	B111
56 - 63	B112	B113	B114	B115	B116	B117	B118	B119	B120	B121	B122	B123	B124	B125	B126	B127
64 - 71	B128	B129	B130	B131	B132	B133	B134	B135	B136	B137	B138	B139	B140	B141	B142	B143
72 - 79	B144	B145	B146	B147	B148	B149	B150	B151	B152	B153	B154	B155	B156	B157	B158	B159
80 - 87	B160	B161	B162	B163	B164	B165	B166	B167	B168	B169	B170	B171	B172	B173	B174	B175
88 - 95	B176	B177	B178	B179	B180	B181	B182	B183	B184	B185	B186	B187	B188	B189	B190	B191
96 - 103	B192	B193	B194	B195	B196	B197	B198	B199	B200	B201	B202	B203	B204	B205	B206	B207
104 - 111	B208	B209	B210	B211	B212	B213	B214	B215	B216	B217	B218	B219	B220	B221	B222	B223
112 - 119	B224	B225	B226	B227	B228	B229	B230	B231	B232	B233	B234	B235	B236	B237	B238	B239
120 - 127	B240	B241	B242	B243	B244	B245	B246	B247	B248	B249	B250	B251	B252	B253	B254	B255

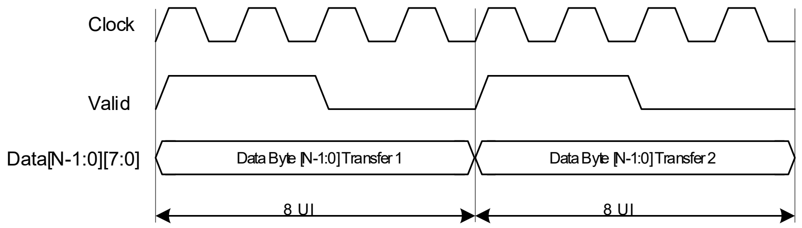
Figure 43. Byte to Lane mapping for Standard package x8 degraded

Lane	8	9	10	11	12	13	14	15
	or							
UI \ Lane	0	1	2	3	4	5	6	7
0 - 7	B0	B1	B2	B3	B4	B5	B6	B7
8 - 15	B8	B9	B10	B11	B12	B13	B14	B15
16 - 23	B16	B17	B18	B19	B20	B21	B22	B23
24 - 31	B24	B25	B26	B27	B28	B29	B30	B31
32 - 39	B32	B33	B34	B35	B36	B37	B38	B39
40 - 47	B40	B41	B42	B43	B44	B45	B46	B47
48 - 55	B48	B49	B50	B51	B52	B53	B54	B55
56 - 63	B56	B57	B58	B59	B60	B61	B62	B63
64 - 71	B64	B65	B66	B67	B68	B69	B70	B71
72 - 79	B72	B73	B74	B75	B76	B77	B78	B79
80 - 87	B80	B81	B82	B83	B84	B85	B86	B87
88 - 95	B88	B89	B90	B91	B92	B93	B94	B95
96 - 103	B96	B97	B98	B99	B100	B101	B102	B103
104 - 111	B104	B105	B106	B107	B108	B109	B110	B111
112 - 119	B112	B113	B114	B115	B116	B117	B118	B119
120 - 127	B120	B121	B122	B123	B124	B125	B126	B127
128-135	B128	B129	B130	B131	B132	B133	B134	B135
136-143	B136	B137	B138	B139	B140	B141	B142	B143
144-151	B144	B145	B146	B147	B148	B149	B150	B151
152-159	B152	B153	B154	B155	B156	B157	B158	B159
160-167	B160	B161	B162	B163	B164	B165	B166	B167
168-175	B168	B169	B170	B171	B172	B173	B174	B175
176-183	B176	B177	B178	B179	B180	B181	B182	B183
184-191	B184	B185	B186	B187	B188	B189	B190	B191
192-199	B192	B193	B194	B195	B196	B197	B198	B199
200-207	B200	B201	B202	B203	B204	B205	B206	B207
208-215	B208	B209	B210	B211	B212	B213	B214	B215
216-223	B216	B217	B218	B219	B220	B221	B222	B223
224-231	B224	B225	B226	B227	B228	B229	B230	B231
232-239	B232	B233	B234	B235	B236	B237	B238	B239
240-247	B240	B241	B242	B243	B244	B245	B246	B247
248-255	B248	B249	B250	B251	B252	B253	B254	B255

4.1.2 Valid Framing

Valid signal is used to frame the transmitted data. For each 8-bit data packet, the valid is asserted for the first 4 UI and de-asserted for 4 UI. This will allow data transfer in raw mode or various Flit modes as described in Chapter 3.0 using one or multiple valid frames. An example is shown in Figure 44 where Transfer 1 and Transfer 2 can be from the same Flit or different Flits.

Figure 44. Valid framing example



4.1.2.1 Valid Framing for Retimers

The UCIe Retimer releases credits to its local UCIe die using the Valid wire, as described in the encoding below. Each credit tracks 256 Bytes of data (including any FEC, CRC, etc). The Valid Framing encodings ensure triple bit flip detection guarantee.

Note: It must be noted that the 8UI blocks is enforced by the Transmitter and tracked by the Receiver in the local UCIe die.

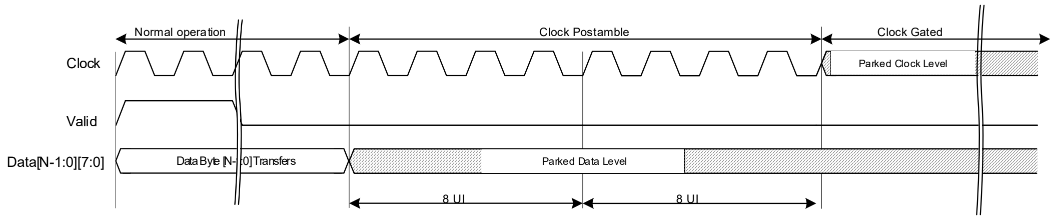
Table 17. Valid framing for Retimers

8-UI Valid (LSB first)	Encoding
1111_1111	Flit data transfer valid + 1 Credit release
1111_0000	Flit data transfer valid + no credit release
0000_1111	No Flit data transfer + 1 credit release
0000_0000	No Flit data transfer + no credit release

4.1.3 Clock Gating

Clocks must be gated when Valid signal is low after providing fixed 16UI (8 cycles) of postamble clock unless free running clock mode is negotiated. Data and Clock signal parking levels are described in section 5.11

Figure 45. Clock gating



4.1.4 Free Running Clock Mode

Free running clock mode is defined as the mode where the forwarded clock remains toggling even when Valid is held low and there is no data transfer on the interface. This mode must be supported to allow disabling dynamic clock gating for normal operation or debug. This must be negotiated prior to main band Link training through parameter exchange.

4.1.5 Sideband transmission

Each module supports a sideband interface with a serial data and clock pin pair. Sideband packet formats and encodings are shown in Chapter 6.0 and the electrical characteristics are shown in section 5.13.

As shown in section 6.1.2, the sideband message formats are defined as 64-bit header with 32 bits or 64 bits of data. A-64 bit serial packet is defined on the IO interface to the remote die as shown in Figure 47. Even 32 bit data is sent using the 64 bit serial packet with MSBs padded with 0b. Two back to back sideband packets on the IO interface are separated by 32-bit low as shown in Figure 46.

Figure 46. Sideband serial packet

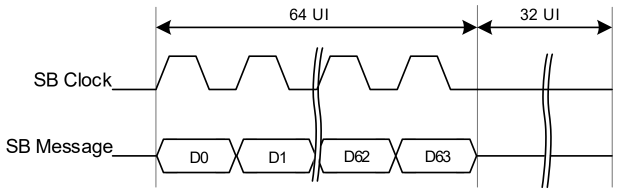
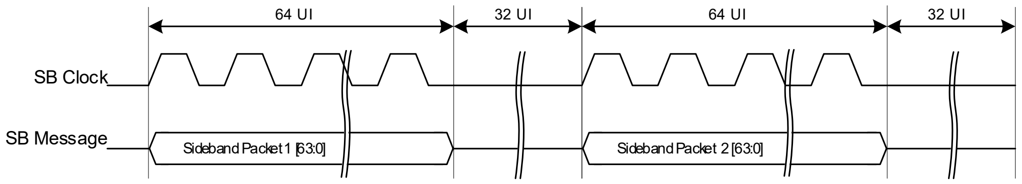


Figure 47. Sideband packet transmission: back to back



4.2 Lane Reversal

In section 4.2, section 4.3 and section 4.5, the following nomenclature is used.

- **TD_P**: Physical Lane for Data Transmitter
- **RD_P:** Physical Lane for Data Receiver
- **TRD_P:** Physical Lane for Redundant Data Transmitter
- **RRD_P:** Physical Lane for Redundant Data Receiver
- **TD_L:** Logical Lane for Data Transmit
- **RD_L:** Logical Lane for Data Receive
- **TCKP_P**, **TCKN_P** and **TTRK_P:** Physical Lane for Clock and Track Transmitter
- **TCKP_L**, **TCKN_L** and **TTRK_L:** Logical Lane for Clock and Track Transmitter
- **RCKP_P**, **RCKN_P** and **RTRK_P:** Physical Lane for Clock and Track Receiver
- **RCKP_L**, **RCKN_L** and **RTRK_L:** Logical Lane for Clock and Track Receiver
- **TRDCK_P:** Physical Lane for Redundant Clock/Track Transmitter
- **RRDCK_P:** Physical Lane for Redundant Clock/Track Receiver
- **TRDCK_L:** Logical Lane for Redundant Clock/Track Transmitter
- **RRDCK_L:** Logical Lane for Redundant Clock/Track Receiver
- **TVLD_P, RVLD_P:** Physical Lane for Valid Transmitter and Receiver
- **TRDVLD_P, RRDVLD_P:** Physical Lane for Redundant Valid Transmitter and Receiver

- **`TVLD_L, RVLD_L:`** Logical Lane for Valid Transmitter and Receiver
- **`TRDVLD_L, RRDVLD_L:`** Logical Lane for Redundant Valid Transmitter and Receiver

Devices must support Lane reversal within a Module. An example of Lane reversal is when physical Data Lane 0 on local die is connected to physical Data Lane (N-1) on the remote die (physical Data Lane 1 is connected to physical Data Lane N-2 and so on) where N = 16 for Standard Package and N=64 for Advanced Package. Redundant Lanes, in case of Advanced Package are also reversed.Lane reversal must be implemented on the Transmitter only. The Transmitter reverses the logical Lane order on Data and Redundant Lanes.

Track, Valid, Clock and sideband signals must not be reversed.

Lane reversal is discovered and applied during initialization and training (section 4.5.3.3.5).

4.2.1 Lane ID

To allow Lane reversal discovery, each logical Data and redundant Lane within a module is assigned a unique Lane ID. The assigned Lane IDs are shown in Table 18 in Table 19 for Advanced and Standard Package modules respectively. Note that logical Lane numbers in Table 18 and Table 19 represent the logical Transmitter and Receiver Lanes. For example, Logical Lane Number = 0 represents **`TD_L[0]/RD_L[0]`** and so on.

In Table 18, for advanced package module, logical Lane numbers 64, 65, 66 and 67 represent Logical redundant Lanes **`TRD_L[0]/RRD_[0], TRD_L[1]/RRD_[1], TRD_L[2]/RRD_[2], TRD_L[3]/RRD_[3]`** respectively

Table 18. Lane ID: Advanced Package module

Logical Lane Number	Lane ID	Logical Lane Number	Lane ID
0	8b'00000000	34	8b'00100010
1	8b'00000001	35	8b'00100011
2	8b'00000010	36	8b'00100100
3	8b'00000011	37	8b'00100101
4	8b'00000100	38	8b'00100110
5	8b'00000101	39	8b'00100111
6	8b'00000110	40	8b'00101000
7	8b'00000111	41	8b'00101001
8	8b'00001000	42	8b'00101010
9	8b'00001001	43	8b'00101011
10	8b'00001010	44	8b'00101100
11	8b'00001011	45	8b'00101101
12	8b'00001100	46	8b'00101110
13	8b'00001101	47	8b'00101111
14	8b'00001110	48	8b'00110000
15	8b'00001111	49	8b'00110001
16	8b'00010000	50	8b'00110010
17	8b'00010001	51	8b'00110011

Table 18. Lane ID: Advanced Package module

Logical Lane Number	Lane ID	Logical Lane Number	Lane ID
18	8b'00010010	52	8b'00110100
19	8b'00010011	53	8b'00110101
20	8b'00010100	54	8b'00110110
21	8b'00010101	55	8b'00110111
22	8b'00010110	56	8b'00111000
23	8b'00010111	57	8b'00111001
24	8b'00011000	58	8b'00111010
25	8b'00011001	59	8b'00111011
26	8b'00011010	60	8b'00111100
27	8b'00011011	61	8b'00111101
28	8b'00011100	62	8b'00111110
29	8b'00011101	63	8b'00111111
30	8b'00011110	64	8b'01000000
31	8b'00011111	65	8b'01000001
32	8b'00100000	66	8b'01000010
33	8b'00100001	67	8b'01000011

Table 19. Lane ID: Standard Package Module

Logical Lane Number	Lane ID	Logical Lane Number	Lane ID
0	8b'00000000	8	8b'00001000
1	8b'00000001	9	8b'00001001
2	8b'00000010	10	8b'00001010
3	8b'00000011	11	8b'00001011
4	8b'00000100	12	8b'00001100
5	8b'00000101	13	8b'00001101
6	8b'00000110	14	8b'00001110
7	8b'00000111	15	8b'00001111

4.3 Interconnect redundancy remapping

As discussed in section 5.9, Advanced Package modules require redundancy remapping to recover from faulty Lanes. This section provides the details of remapping.

4.3.1 Data Lane repair

The specification supports remapping (repair) of up to two data Lanes for each group of 32 data Lanes. `TD_P[31:0] (RD_P[31:0])` and `TD_P[63:32] (RD_P[63:32])` are treated as two separate groups of 32 Lanes that can be independently repaired using redundant Lanes, **`TRD_P[1:0](RRD_P[1:0])`** and **`TRD_P[3:2](RRD_P[3:2])`** respectively.

Lane remapping is accomplished by "shift left" or "shift right" operation. A "shift left" is when data traffic of logical Lane **`TD_L[n]`** on **`TD_P[n]`** is multiplexed onto **`TD_P[n-1]`**. A shift right operation is when data traffic **`TD_L[n]`** is multiplexed onto **`TD_P[n+1]`**.

After a data Lane is remapped, the Transmitter associated with the broken physical Lane is tri-stated and the Receiver is disabled. The Transmitter and the Receiver of the redundant Lane used for the repair are enabled.

Figure 48 shows transmit bump side of data Lane remapping for the first group of 32 Lanes. Both "shift left" and "shift right" remapping is needed to optimally repair up to any two Lanes within the group. Figure 49 shows details of the mux structure used for data Lane repair. .

Note: Example repair implementations are shown for **`TD_P[31:0]`** for clarity. It should be noted that same schemes are also applicable to **`TD_P[63:32]`**.

Figure 48. Data Lane remapping possibilities to fix potential defects

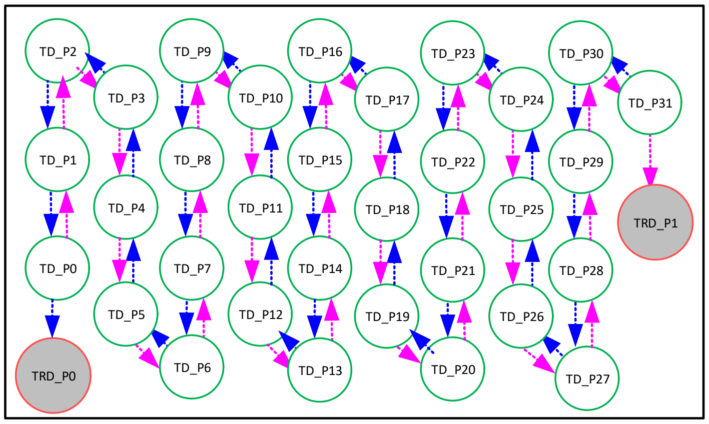
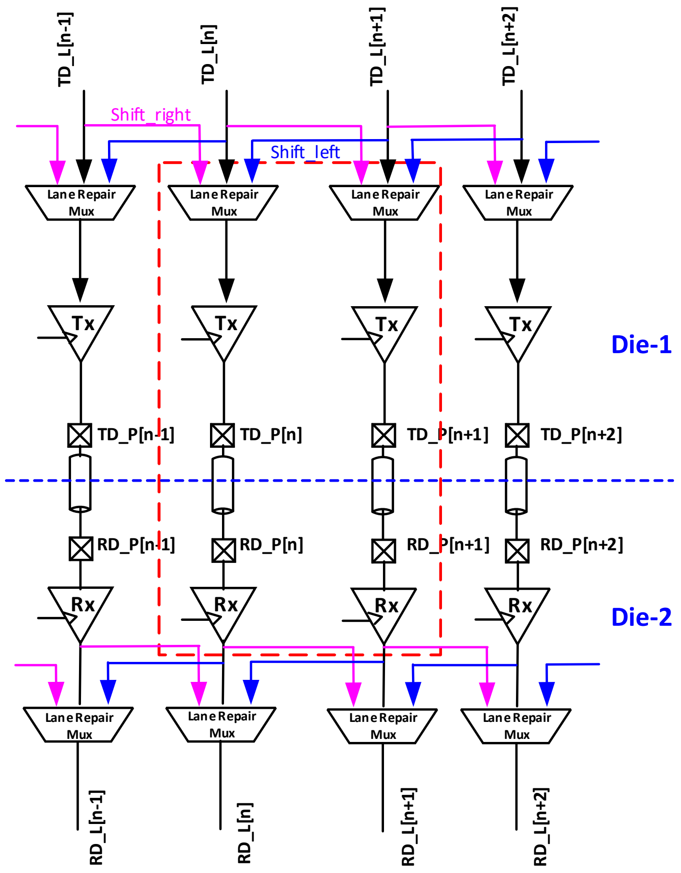


Figure 49. Data Lane remapping: Mux chain



4.3.2 Data Lane repair with Lane reversal

If Lanes are reversed, physical Lane 0 of Transmitter (`TD_P[0]`) is connected to physical Lane N-1 of the Receiver (`RD_P[N-1]`). N is 64 for Advanced package modules and 16 for Standard package module.

Lane repair mapping changes when Lane reversal is required.

4.3.3 Data Lane repair implementation

4.3.3.1 Single Lane repair

`TRD_P[0]` (`RRD_P[0]`) must be used as the redundant Lane to remap any single physical Lane failure for `TD_P[31:0]`(`RD_P[31:0]`). `TRD[2]` (`RRD[2]`) must be used as the redundant Lane to remap any single Lane failure for `TD_P[63:32]` (`RD_P[63:32]`).

Pseudo code for repair in `TD_P[31:0]`(`RD_P[31:0]`) (0<= x <=31):

```
IF failure occurs in TD_P[x]:
        IF x > 0:
                FOR 0 <= i < x:
                        TD_P[x-i-1] = TD_L[x-i]
                        RD_L[x-i] = RD_P[x-i-1]
        TRD_P[0] = TD_L[0]
        RD_L[0] = RRD_P[0]
```

Pseudo code for repair in`TD_P[63:32]`(`RD_P[63:32]`) (32<= x <=63):

```
IF failure occurs in TD_P[x]:
        IF x > 32:
                FOR 0 <= i < x-32:
                        TD_P[x-i-1] = TD_L[x-i]
                        RD_L[x-i] = RD_P[x-i-1]
        TRD_P[2] = TD_L[32]
        RD_L[32] = RRD_P[2]
```

As shown in Figure 50 `TD_P[29]` is remapped in the direction to use `TRD_P[0]` as the repair resource.

Figure 50. Single Lane failure remapping

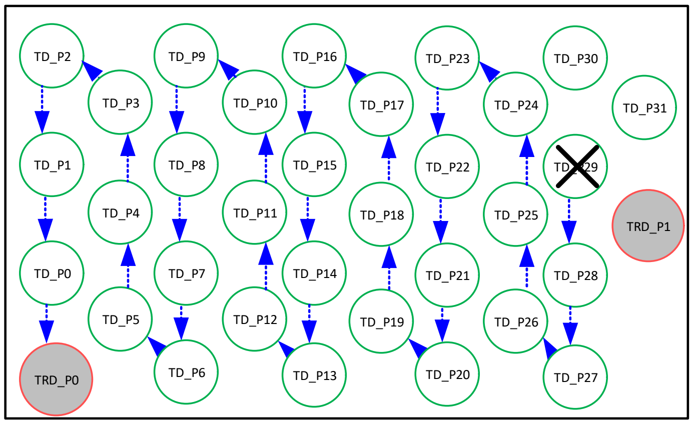
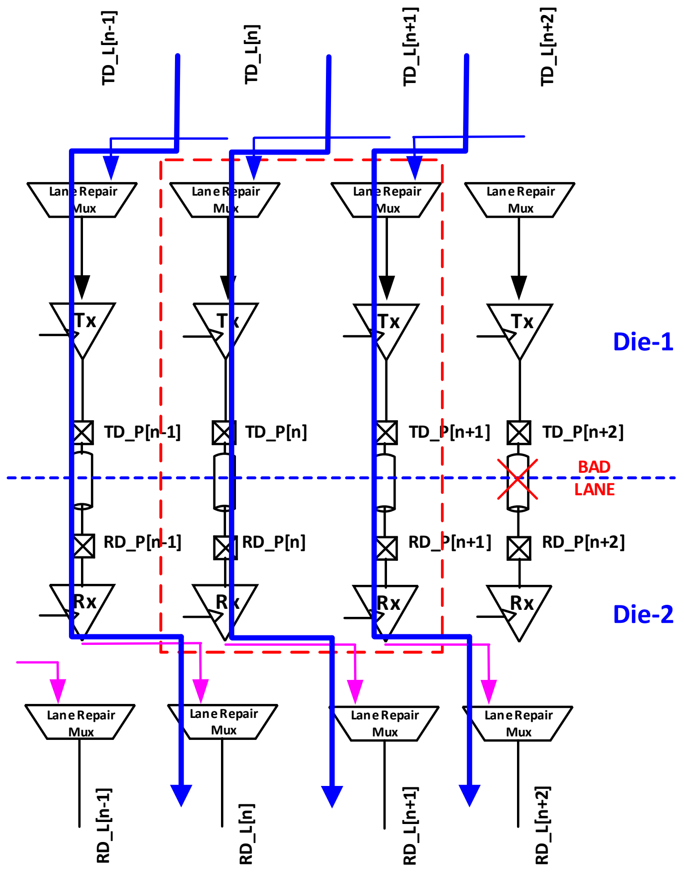


Figure 51. Single Lane remapping implementation



4.3.3.2 Two Lane repair

Any two Lanes within a group of 32 can be repaired using the two redundant bumps. For any two physical Lane failures in `TD_P[31:0]` (`RD_P[31:0]`), the lower Lane must be remapped to `TRD_P[0]` (`RRD_P[0]`) and the upper Lane is remapped to `TRD_P[1]` (`RRD_P[1]`). For any two physical Lane failures in `TD_P[63:32]` (`RD_P[63:31]`), the lower Lane must be remapped to `TRD_P[2]` (`RRD_P[2]`) and the upper Lane is remapped to `TRD_P[3]` (`RRD_P[3]`).

Pseudo code for two Lane repair in `TD_P[31:0]`(`RD_P[31:0]`) (0<= x,y <=31):

```
IF failure occurs in TD_P[x], TD_P[y] AND (x < y):
        IF x > 0:
                FOR 0 <= i < x:
                        TD_P[x-i-1] = TD_L[x-i]
                        RD_L[x-i] = RD_P[x-i-1]
        TRD_P[0] = TD_L[0]
        RD_L[0] = RRD_P[0]
        IF y < 31:
                FOR 0 <= j < (31-y):
                        TD_P[y+j+1] = TD_L[y+j]
                        RD_L[y+j]=RD_P[y+j+1]
        TRD_P[1] = TD_L[31]
        RD_L[31]=RRD_P[1]
```

Pseudo code for two Lanerepair in `TD_P[63:32]` (`RD_P[63:32]`) (32<= x,y <=63):

```
IF failure occurs in TD_P[x], TD_P[y] AND (x < y):
        IF x > 32:
                FOR 0 <= i < x-32:
                        TD_P[x-i-1] = TD_L[x-i]
                        RD_L[x-i] = RD_P[x-i-1]
        TRD_P[2] = TD_L[32]
        RD_L[32] = RRD_P[2]
        IF y < 63:
                FOR 0 <= j < (63-y):
                        TD_P[y+j+1] = TD_L[y+j]
                        RD_L[y+j] = RD_P[y+j+1]
        TRD_P[2] = TD_L[63]
        RD_L[63] = RRD_P[2]
```

Shows in Figure 52 is an example of two (physical Lanes 25 and 26) Lane remapping. Figure 53 shows the circuit implementation. Both Transmitter and Receiver must apply the required remapping.

Figure 52. Two Lane failure remapping

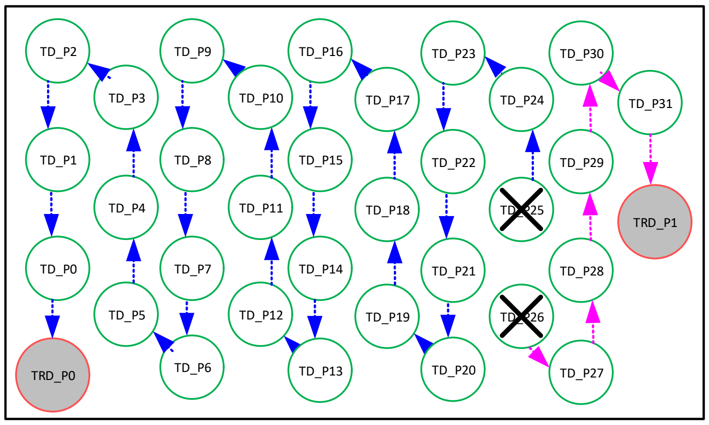
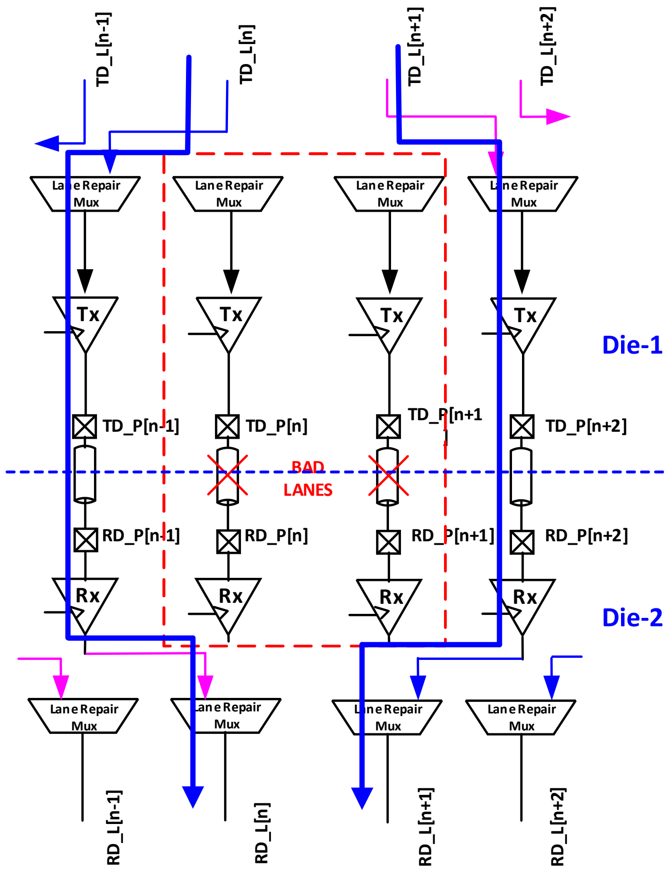


Figure 53. Two Lane remapping implementation



4.3.3.3 Single Lane repair with Lane reversal

For repair with Lane reversal (section 4.3.2) Transmitter side remapping is reversed to preserve shifting order for Receiver side remapping.

Pseudo code for one Lane failure in `TD_P[31:0]`(`RD_P[32:63]`) (0<= x <=31):

```
IF failure occurs in TD_P[x]:
        IF x > 0:
                FOR 0 <= i < x:
                        TD_P[x-i-1] = TD_L[63-x+i]
                        RD_L[63-x+i] = RD_P[63-x+i+1]
        TRD_P[0] = TD_P[63]
        RD_P[63] = RRD_P[3]
```

Pseudo code for one Lane failure in `TD_P[63:32]`(`RD_P[0:31]`) (32<= x <=63):

```
IF failure occurs in TD_P[x]:
      IF x > 32:
            FOR 0 <= i < x-32:
                  TD_P[x-i-1] = TD_L[63-x+i]
                  RD_L[63-x+i] = RD_P[63-x+i+1]
      TRD_P[2] = TD_L[31]
      RD_L[31] = RRD_P[1]
```

4.3.3.4 Two Lane repair with Lane reversal

For repair with Lane reversal (section 4.3.2) Transmitter side remapping is reversed to preserve shifting order for Receiver side remapping.

Pseudo code for two Lane failure in `TD_P[31:0]`(`RD_P[32:63]`) (0<= x <=31):

```
IF failure occurs in TD_P[x], TD_P[y] AND (x < y):
      IF x > 0:
            FOR 0 <= i < x:
                  TD_P[x-i-1] = TD_L[63-x+i]
                  RD_L[63-x+i] = RD_P[63-x+i+1]
      TRD_P[0] = TD_L[63]
      RD_L[63] = RRD_P[3]
      IF y < 31:
            FOR 0 <= j < (31-y):
                  TD_P[y+j+1] = TD_L[63-y-j]
                  RD_L[63-y-j] = RD_P[63-y-(j+1)]
      TRD_P[1] = TD_L[32]
      RD_L[32] = RRD_P[2]
```

Pseudo code for one Lane failure in `TD_P[63:32]`(`RD_P[0:31]`) (32<= x <=63):

```
IF failure occurs in TD_P[x], TD_P[y] AND (x < y):
      IF x > 32:
            FOR 0 <= i < x-32:
                  TD_P[x-i-1] = TD_L[63-x+i]
                  RD_L[63-x+i] = RD_P[63-x+(i+1)]
      TRD_P[2] = TD_L[31]
      RD_L[31] = RRD_P[1]
      IF y < 63:
            FOR 0 <= j < (63-y):
                  TD_P[y+j+1] = TD_L[63-y-j]
                  RD_L[63-y-j] = RD_P[63-y-(j+1)]
      TRD_P[3] = TD_L[0]
      RD_L[0] = RRD_P[0]
```

4.3.4 Clock and Track Lane remapping

The specification supports remapping of one broken Lane for **TCKP_P/RCKP_P**, **TCKN_P/RCKN_P** and **TTRK_P/RTRK_P** physical Lanes. The repair scheme is shown in Figure 54. Clock Lane remapping allows repair of single Lane failure for both differential and pseudo-differential implementation of the clock Receiver. The circuit details are shown in Figure 55 and Figure 56 for differential and pseudo-differential clock Receivers respectively.

After a Lane is remapped, the Transmitter is tri-stated . The Receiver of the physical redundant (**RRDCK_P**) Lane is disabled.

Figure 54. Clock and Track repair

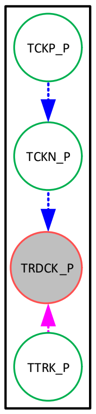


Figure 55. Clock and track repair: Differential Rx

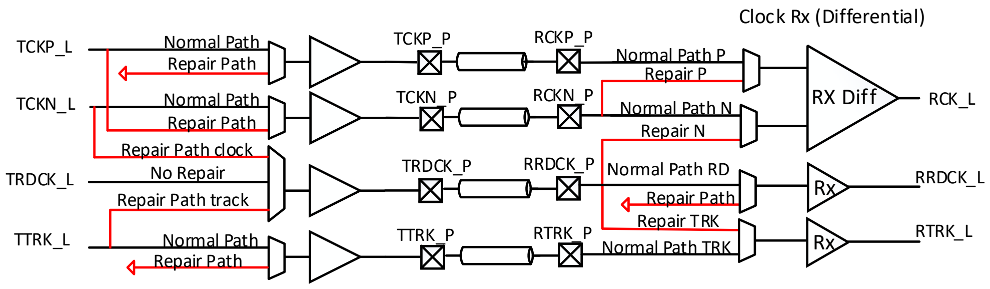
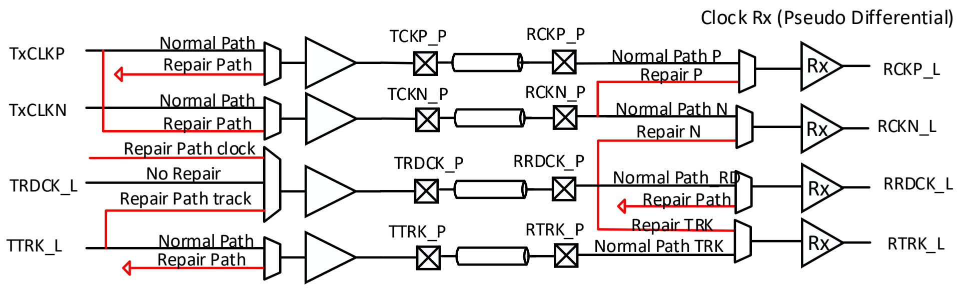


Figure 56. Clock and track repair: Pseudo Differential Rx



4.3.5 Clock and Track Lane repair implementation

Pseudo code for Clock and Track Lane repair:

```
IF failure occurs in TCKP_P:
       TCKN_P = TCKP_L AND TRDCK_P = TCKN_L
       RCKP_L = RCKN_P AND RCKN_L = RRDCK_P
ELSE IF failure occurs on TCKN_P:
       TRDCK_P = TCKN_L
       RCKN_L = RRDCK_P
ELSE IF failure occurs in TTRK_P:
       TRDCK_P = TTRK_L
       RTRK_L = RRDCK_P
```

The implementation of Clock and Track Lane remapping is shown in Figure 57(a), Figure 57(b) and Figure 57(c) respectively. The corresponding circuit level details of remapping implementation are shown in Figure 58, Figure 59 and Figure 60.

Note that the both Transmitter and Receiver on **CKRD** Lane are required during detection phase and can be tri-stated and turned off if not used for repair.

Figure 57. Clock and Track Lane repair scheme

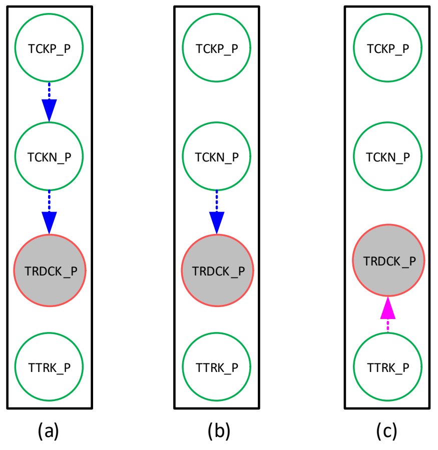


Figure 58. Clock and track repair: CKP repair

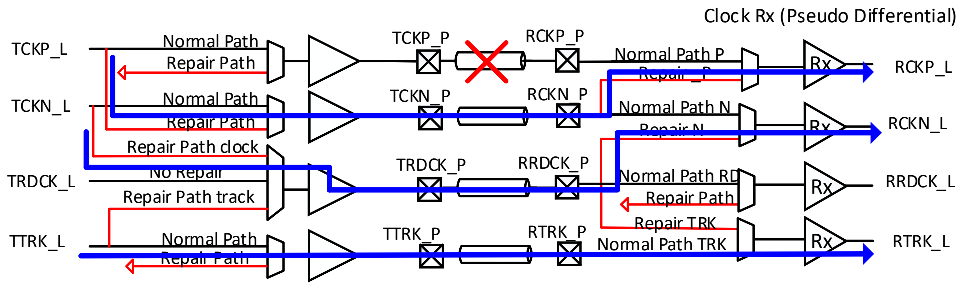


Figure 59. Clock and track repair: CKN repair

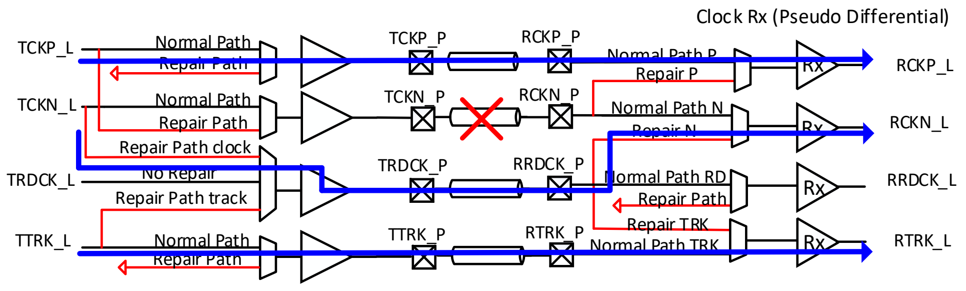
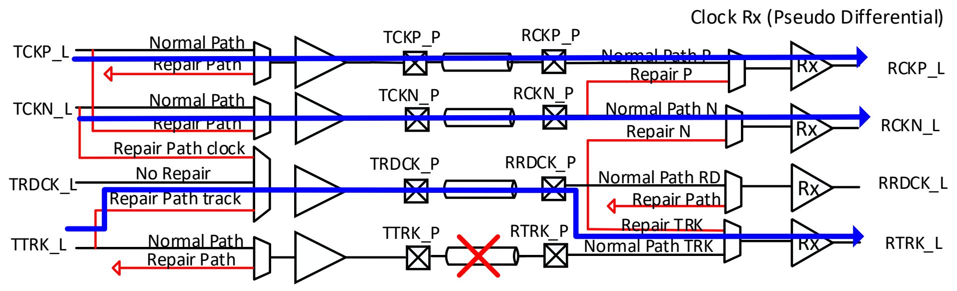


Figure 60. Clock and track repair: Track repair



4.3.6 Valid Repair and implementation

Valid Lane has a dedicated redundant Lane. If a failure is detected on the Valid physical Lane, redundant Valid physical Lane is used to send Valid. Valid failure detection and repair is performed during Link initialization and Training (section 4.5.3.3.4).

Figure 61 shows the normal path for Valid and redundant valid Lanes. Figure 62 shows the repair path for Valid Lane failure.

Figure 61. Valid repair: Normal Path

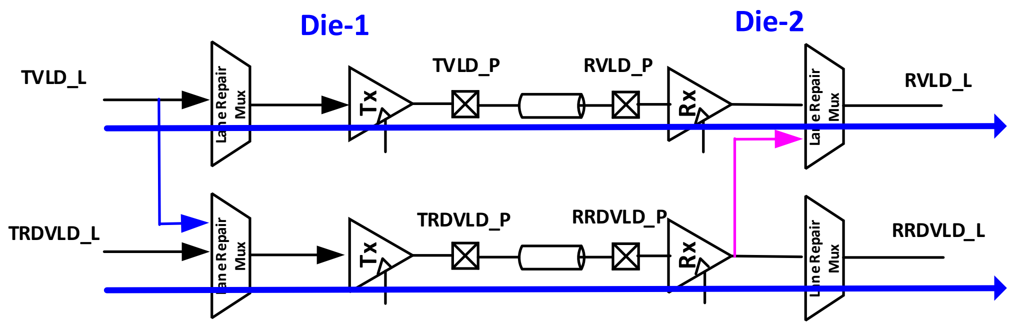
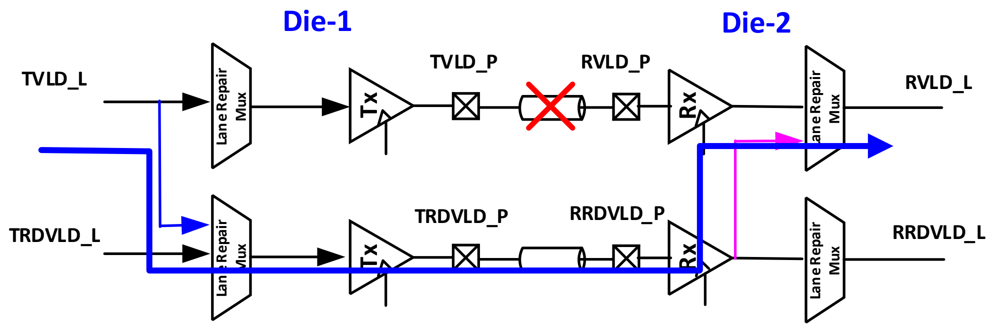


Figure 62. Valid Repair: Repair Path



4.3.7 Width Degrade in Standard Package Interfaces

In case of Standard Package x16 modules where Lane repair is not supported resilience against faulty Lanes is provided by configuring the Link to a x8 width (logical Lanes 0 to 7 or Logical Lanes 8 to 15 which exclude the faulty Lanes). For example, if one or more faulty Lanes are in logical Lane 0 to 7, the Link is configured to x8 width using logical Lanes 8 to 15. The configuration is done during Link initialization or retraining and Transmitters of the disabled Lanes go to hi-Z and Receivers are disabled.

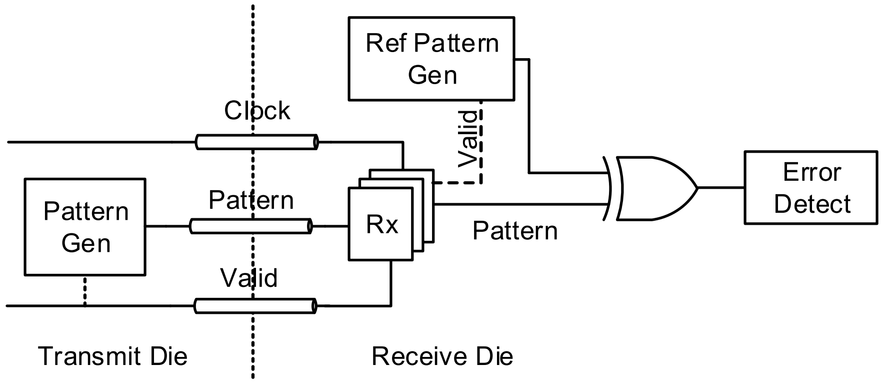
Figure 43 shows the byte to Lane mapping for a width degraded x8 interface

4.4 Data to Clock Training and Test Modes

Note: Sideband commands will be identified as {SB command}.

Figure 63 shows the infrastructure for interface training and testing. The Transmitter die and receive die implement the same Linear Feedback Shift Register (LFSR) described in section 4.4.1. The pattern sent from the Transmitter along with forwarded clock and Valid is compared with locally generated reference pattern. Both transmit and receive pattern generators must start and advance in sync. The compare circuitry checks for matching data each UI. Any mismatch between the received pattern and pattern predicted by the local pattern generator is detected as an error.

Figure 63. Test and training logic



The receive die must implement two types of comparison schemes

- **Per Lane comparison**: Per Lane comparison is used to identify of failing Lanes between the two dies. Any mismatch, above the set threshold, between received pattern and the expected pattern on each Lane will set a sticky register bit for each Lane. Once a pattern mismatch on a particular Lane is found, the sticky register bit is set for the remainder of the test. Figure 64 illustrates per Lane comparison mode. This mode will indicate per Lane errors within the module (N = 68 (64 + 4 RD) and 16 for Advanced and Standard Package modules respectively). The per Lane comparison results can be read via sideband.

- **Aggregate comparison**: In this mode, pattern mismatches each UI on any Lane within the module are accumulated into an 16 bit error counter. The Lane errors are ORed to generate a module level error and counted as shown in Figure 65. This scheme can be used for module level margin and BER. The error counter can be read via sideband.

Figure 64. Lane failure detection

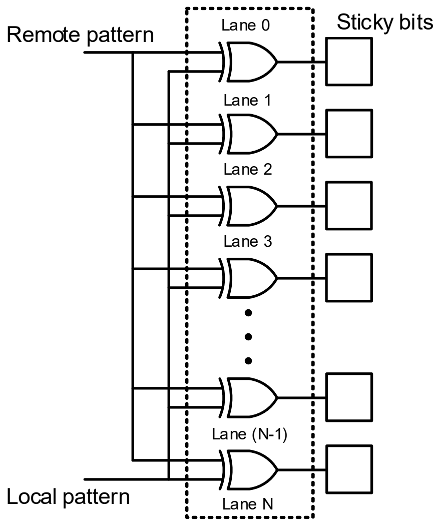
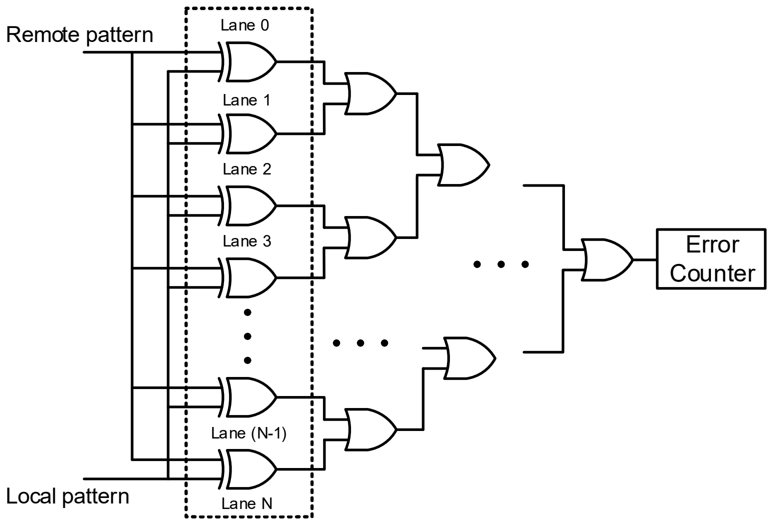


Figure 65. All Lane error detection



4.4.1 Scrambling and training pattern generation

A Linear feedback shift register (LFSR) is defined for scrambling and test pattern generation.

The LFSR uses the same polynomial as PCIe: $G(X)=X^{23} + X^{21} + X^{16} + X^{8} + X^{5} + X^{2} + 1$. Each Transmitter is permitted to implement a separate LFSR for scrambling and pattern generation. Each Receiver is permitted to implement a separate LFSR for de-scrambling and pattern comparison. The implementation is shown in Figure 66. The seed of the LFSR is Lane dependent and the seed value for Lane number is modulo 8 as shown in Table 20.

Alternatively, implementations can choose to implement one LFSR with different tap points for multiple Lanes as shown in Figure 67. This is equivalent to individual LFSR per-Lane with different seeds.

Table 20. LFSR seed values

Lane	Seed
0	23'h1DBFBC
1	23'h 0607BB
2	23'h1EC760
3	23'h18C0DB
4	23'h010F12
5	23'h19CFC9
6	23'h0277CE
7	23'h1BB807
RD0, RD2[1]	23'h1EC760
RD1, RD3[2]	23'h18C0DB

1. Same as Lane 2
2. Same as Lane 3

Figure 66. LFSR implementation

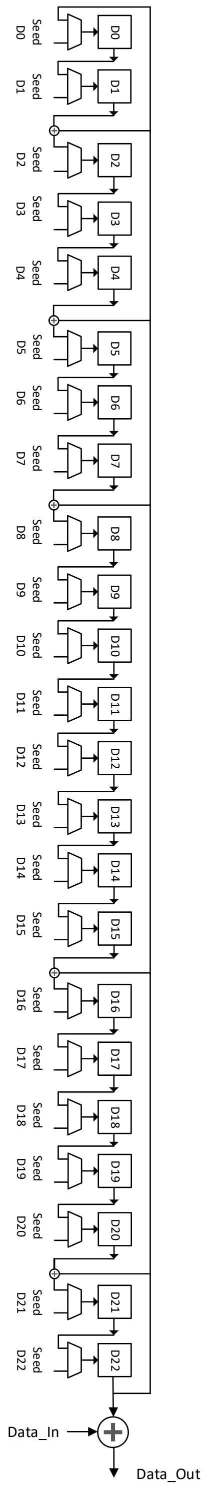
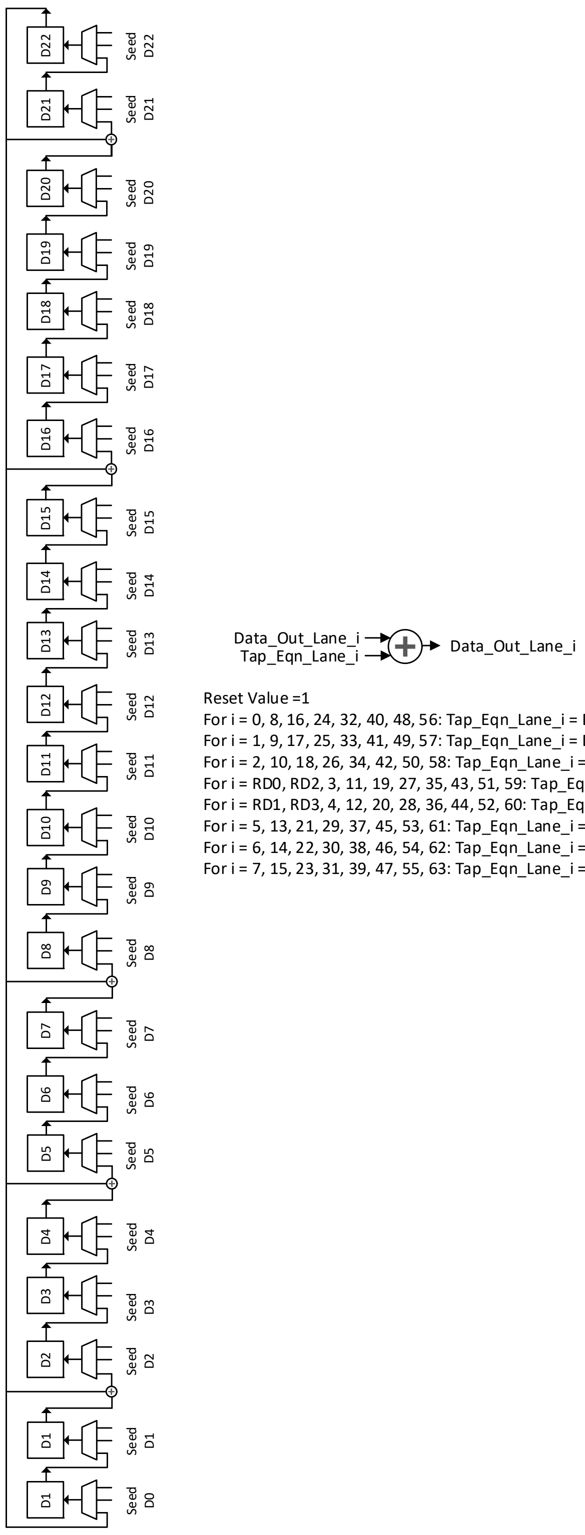


Figure 67. LFSR alternate implementation



Reset Value =1
For i = 0, 8, 16, 24, 32, 40, 48, 56: Tap_Eqn_Lane_i = D9^D13
For i = 1, 9, 17, 25, 33, 41, 49, 57: Tap_Eqn_Lane_i = D1^D13
For i = 2, 10, 18, 26, 34, 42, 50, 58: Tap_Eqn_Lane_i = D13^22
For i = RD0, RD2, 3, 11, 19, 27, 35, 43, 51, 59: Tap_Eqn_Lane_i = D1^D22
For i = RD1, RD3, 4, 12, 20, 28, 36, 44, 52, 60: Tap_Eqn_Lane_i = D3^D22
For i = 5, 13, 21, 29, 37, 45, 53, 61: Tap_Eqn_Lane_i = D1^D3
For i = 6, 14, 22, 30, 38, 46, 54, 62: Tap_Eqn_Lane_i = D3^D9
For i = 7, 15, 23, 31, 39, 47, 55, 63: Tap_Eqn_Lane_i = D1^D9

4.5 Link Initialization and Training

Link initialization and training are described at a Module level. The description of terms are used in this section are as follows:

- UCIe Module Partner: A UCIe Module partner is the corresponding UCIe module on the remote UCIe die to which the UCIe Module connects. For example, if two UCIe dies A and B are connected by a 2-module UCIe Link, modules 0 and 1; UCIe die A's module 0 (UCIe module A0) is connected to UCIe die B's module 1 (UCIe module B1); then A0 is the UCIe module partner of B1 and vice-versa.
- Phase Interpolator (PI) in this specification is used to refer any method of generating and selecting sampling clock phase.
- Timeout: Every state except RESET and TRAINERROR in the Link Training State Machine has a residency timeout of 8ms. For states with sub-states, the timeout is per sub-state; i.e., if Physical Layer is in that state for greater than 8ms, then it transitions to TRAINERROR and eventually to RESET. Physical Layer sideband handshakes for RDI state transitions with remote Link partner also timeout after 8ms. The timeout counters must be reset if a sideband message with "Stall" encoding is received.
- Electrical idle is described in section 5.12

When training during Link bring up (i.e., Physical Layer transitions out of RESET state), hardware is permitted to attempt training multiple times. The triggers for initiating Link Training are:

- Software writes 1b to Start UCIe Link Training bit in UCIe Link Control
- Adapter triggers Link Training on RDI (RDI status is Reset and there is a NOP to Active transition on the state request)
- SBINIT pattern (two consecutive iterations of 64 UI clock pattern and 32 UI low) is observed on any sideband Receiver clock/data pair

Once hardware fails training after an implementation specific number of attempts, it must transition to RESET and wait for a subsequent Link Training trigger. Physical Layer must escalate a fatal error on RDI if Software triggered or RDI triggered Link training fails or there is a Link up to Link down transition due to a Physical Layer timeout.

4.5.1 Link Training basic operations

Some basic operations in Link training are defined in this section. These will be used in main band initialization, training and margining.

4.5.1.1 Transmitter initiated Data to Clock point test

In this mode, the Transmitter initiates the data to clock training on all Lanes in the module at a single PI phase. The sequence of steps for this test are as follows for each UCIe module of the UCIe Link:

1. The UCIe Module sets up the Transmitter parameters (shown in Table 21.) sends a sideband message {Start Tx Init D to C point test req} to its UCIe Module partner and waits for a response. The data field of this message includes the required parameters, shown in Table 21The Receiver on the UCIe Module partner must enable the pattern comparison circuits to compare incoming main band data to the locally generated expected pattern. Once the data to clock training parameters for its Receiver are setup, the UCIe Module partner responds with {Start Tx Init D to C point test resp}.

2. The UCIe Module resets the LFSR (scrambler) on its mainband Transmitters and sends sideband message {LFSR clear error req}. The UCIe Module Partner resets the LFSR and clears any prior compare results on its mainband Receivers and responds with {LFSR clear error resp} sideband message.
3. The UCIe Module sends the pattern (selected through "Tx Pattern Generator Setup") for the selected number of cycles ("Pattern Count Setup") on its main band Transmitter.
4. The UCIe Module Partner performs the comparison on its Receivers for each UI during the pattern transmission based on "Rx compare setup" and logs the results.
5. The UCIe Module requests its UCIe Module Partner for the logged results in Step 4 by sending {D to C results Req} sideband message. The UCIe Module Partner stops comparison on its mainband Receivers and responds with the logged results {D to C results log} sideband message.
6. The UCIe Module stops sending the pattern on its Transmitters and sends the {End Tx Init D to C point test req} sideband message and the UCIe Module Partner responds with {End Tx Init D to C point test resp}. When a UCIe Module has sent and received the {End Tx Init D to C point test resp}, it enters electrical idle.

Table 21. Data to Clock Training Parameters

Parameter	Options
Clock sampling /PI phase control	Eye Center found by training
	Left Edge found by training
	Right Edge found by training
Tx Pattern Generator Setup	LFSR Pattern for Data Lanes
	Per-Lane ID pattern for Data Lanes as defined in Chapter 4.5.3.3.5 in Table 23
	VALTRAIN pattern four 1's followed by four 0's
Tx Pattern Count Setup	Burst Mode: Burst Count/Idle Count/Iteration Count
	Continuous Mode count
Rx Compare Setup	Maximum comparison error threshold
	Per-Lane or aggregate error compare

4.5.1.2 Transmitter initiated Data to Clock eye width sweep

In this mode, the Transmitter initiates the data to clock training on all Lanes in the module and sweeps through the sampling PI phases. This mode can also be used to check system margin. The sequence of steps for this test are as follows:

1. The UCIe Module sets up the Transmitter parameters (shown in Table 21.) sends a sideband message {Start Tx Init D to C eye sweep req} to its UCIe Module partner and waits for a response. The data field of this message includes the required parameters, shown in Table 21. The Receiver on the UCIe Module partner must enable the pattern comparison circuits to compare incoming main band data to the locally generated expected pattern. Once the data to clock training parameters for its Receiver are setup, the UCIe Module partner responds with sideband message {Start Tx Init D to C eye sweep resp}
2. The UCIe Module resets the LFSR (scrambler) on its mainband Transmitters and sends sideband message {LFSR clear error req}. The UCIe Module Partner resets

the LFSR and clears any prior compare results on its mainband Receivers and responds with {LFSR clear error resp} sideband message.

3. The UCIe Module sends the pattern (selected through “Tx Pattern Generator Setup”) for the selected number of cycles (“Pattern Count Setup”) on its main band Transmitter.
4. The UCIe Module Partner performs comparison on its Receivers for each UI during the pattern transmission based on “Rx compare setup” and logs the results.
5. The UCIe Module requests its UCIe Module Partner for the logged results in Step 4 by sending {D to C results Req} sideband message. The UCIe Module Partner stops comparison on its mainband Receivers and responds with the logged results {D to C results log} sideband message.
6. The UCIe Module sweeps through the PI phases on its forwarded clock Transmitter each time repeating Step 2 through Step 5 to find the passing PI phase range. The sweep steps and range are implementation specific.
7. The UCIe Module stops sending the pattern on its Transmitters and sends the {End Tx Init D to C eye sweep req} sideband message and the UCIe Module Partner responds with {End Tx Init D to C eye sweep resp}. When a UCIe Module has sent and received the {End Tx Init D to C point eye sweep resp}, it enters electrical idle.

4.5.1.3 Receiver initiated Data to Clock point training

In this mode, the Receiver initiates the data to clock training on all Lanes in the module at a single phase of the phase interpolator (PI). The sequence of steps for this test are as follows:

1. The UCIe Module enables the pattern comparison circuits to compare incoming main band data to the locally generated expected pattern, sets up the Receiver parameters (shown in Table 21.) sends a sideband message {Start Rx Init D to C point test req} to its UCIe Module partner and waits for a response. The data field of this message includes the required parameters, shown in Table 21. The Transmitter on the UCIe Module partner must be idle. Once the data to clock training parameters for its Transmitter are setup, the UCIe Module partner responds with sideband message {Start Rx Init D to C point test resp}
2. The UCIe Module Partner resets the LFSR (scrambler) on its mainband Transmitters and sends sideband message {LFSR clear error req}. The UCIe Module resets the LFSR and clears any prior compare results on its mainband Receivers and responds with {LFSR clear error resp} sideband message.
3. The UCIe Module Partner sends the pattern (selected through “Tx Pattern Generator Setup”) for the selected number of cycles (“Pattern Count Setup”) on its mainband Transmitter.
4. The UCIe Module performs the comparison on its mainband Receivers for each UI during the pattern transmission based on “Rx compare setup” and logs the results.
5. The UCIe Module Partner sends a sideband message {Rx Init D to C Tx count done req} sideband message once the pattern count is complete. The UCIe Module, stops comparison and responds with the sideband message {Rx Init D to C Tx count done resp}. The UCIe Module can now use the logged data for its Receiver Lanes.
6. The UCIe Module sends a {End Rx Init D to C point test req} sideband message and the UCIe Module Partner responds with {End Rx init D to C point test resp} sideband message. Then Transmitter and Receiver on the UCIe Module enter idle.

4.5.1.4 Receiver initiated Data to Clock Eye Width sweep training

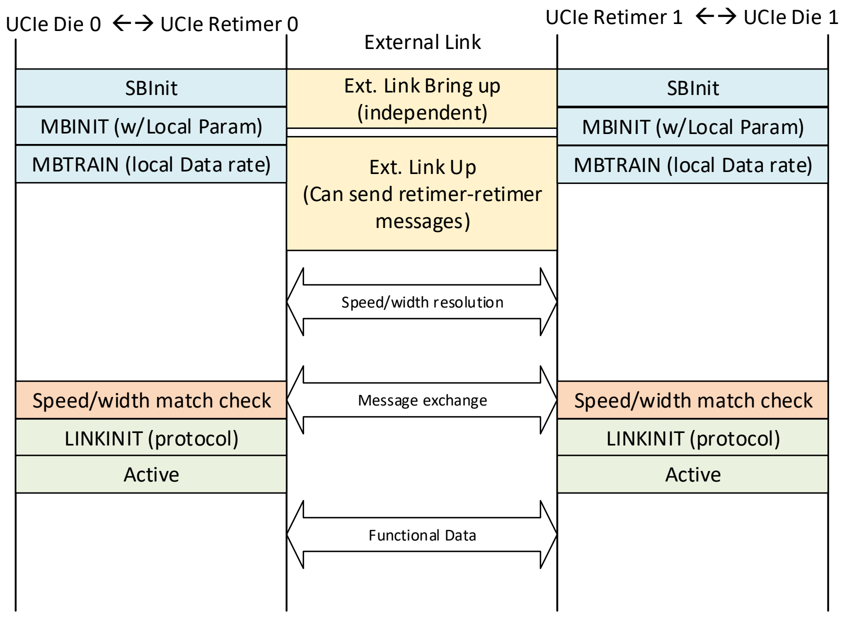
In this mode, the Receiver initiates the data to clock training on all Lanes in the module at a single phase of the phase interpolator (PI). The sequence of steps for this test are as follows:

1. The UCIe Module enables the pattern comparison circuits to compare incoming main band data to the locally generated expected pattern, sets up the Receiver parameters (shown in Table 21.) sends a sideband message {Start Rx Init D to C eye sweep req} to its UCIe Module partner and waits for a response. The data field of this message includes the required parameters, shown in Table 21. The Transmitter on the UCIe Module partner must be idle. Once the data to clock training parameters for its Transmitter are setup, the UCIe Module partner responds with sideband message {Start Rx Init D to C eye sweep resp}
2. The UCIe Module Partner resets the LFSR (scrambler) on its mainband Transmitters and sends sideband message {LFSR clear error req}. The UCIe Module resets the LFSR and clears any prior compare results on its mainband Receivers and responds with {LFSR clear error resp} sideband message.
3. The UCIe Module Partner sends the pattern (selected through "Tx Pattern Generator Setup") for the selected number of cycles ("Pattern Count Setup") on its main band Transmitter.
4. The UCIe Module performs the comparison on its mainband Receivers for each UI during the pattern transmission based on "Rx compare setup" and logs the results.
5. The UCIe Module Partner requests the UCIe Module for the logged data to clock test results through {Rx Init D to C results Req}. The UCIe Module responds with the logged results for its mainband Receiver using the sideband message {Rx Init D to C results log}. The UCIe Module Partner can determine if the pattern comparison at the PI phase passed or failed based on the results log.
6. The UCIe Module Partner sweeps through the PI phases on its forwarded clock each time repeating Step 2 through Step 5 to find the passing PI phase range. The sweep pattern and range are implementation specific.
7. The UCIe Module Partner sends a {Rx Init D to C sweep done with results} sideband message with results for its mainband Transmitter. The UCIe Module can us the sweep results for its mainband Receivers
8. The UCIe Module sends a {End Rx Init D to C eye sweep req} sideband message and the UCIe Module Partner responds with {End Rx init D to C eye sweep resp} sideband message. Then Transmitter and Receiver on the UCIe Module enter idle.

4.5.2 Link Training with Retimer

The following diagram explains the initialization flow with UCIe Retimer. As shown in Figure 68, external and UCIe (UCIe die to UCIe Retimer) Links are permitted to come up independently. When a UCIe Link trains up to local data rate and width, the remote UCIe information is requested over the external Link. If there is a data rate and width difference, each UCIe Link is permitted to be retrained to achieve a speed (data rate) and width match (if the UCIe Retimer requires this for operation, it must initiate Retraining from LinkSpeed state). This can happen multiple times during initial Link bring up or retraining. Once the UCIe Retimer has determined that the UCIe Link configuration is suitable for successful operation with remote Retimer partner, the UCIe Link proceeds to ACTIVE through protocol level Link initialization (LINKINIT)

Figure 68. Example Retimer bring up when performing speed/width match



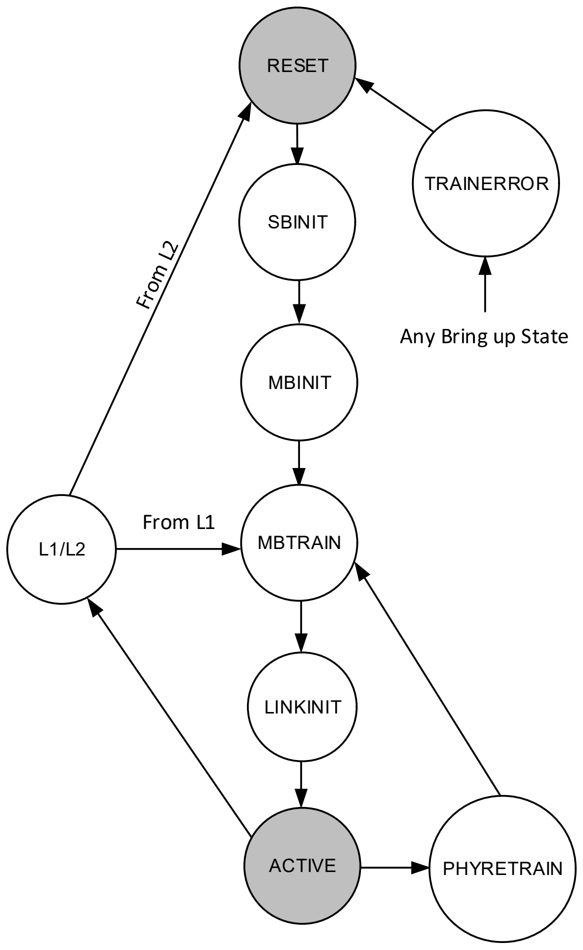
4.5.3 Link Training State Machine

A high level initialization flow is shown in Figure 69. A high level description of each state is shown in Table 22. The details and actions performed in each state are described in the subsequent sections.

Table 22. State Definitions for Initialization

State	Description
RESET	This is the state following primary reset or exit from TRAINERROR
SBINIT	Side band initialization state where the side band is detected, repaired (when applicable) and out of reset message is transmitted
MBINIT	Following sideband initialization, Main band (MB) is initialized at the lowest speed. Both dies perform on die calibration followed by interconnect repair (when applicable)
MBTRAIN	Main band (Data, Clock and Valid signals) speed of operation is set to the highest negotiated data rate. Die-to-Die training of main band is performed to center the clock with respect to Data.
LINKINIT	This state is used to exchange Adapter and Link management messages
ACTIVE	This is the state in which transactions are sent and received
L1/L2	Power Management state
PHYRETRAIN	This state is used to begin the retrain flow for the Link during runtime
TRAINERROR	State is entered when a fatal or non-fatal event occurs at any point during Link Training or operation.

Figure 69. Link Training State Machine



4.5.3.1 RESET

Physical Layer must stay in RESET for a minimum of 4ms upon every entry to RESET, to allow PLLs etc. to stabilize. The minimum conditions necessary to exit RESET are:

- Power supplies are stable
- Sideband clock is available and running at 800 MHz
- Main band and Die to Die Adapter clocks are stable and available
- Main band clock is set to the slowest IO data rate (2 GHz for 4 GT/s)
- Local SoC/Firmware not keeping the Physical Layer in RESET
- Link training trigger has occurred. The triggers were defined in the beginning of section 4.5.

4.5.3.2 Sideband Initialization (SBINIT)

In this state, the sideband (SB) interface is initialized and repaired (when applicable). The UCIe Module mainband (MB) Transmitters are tri-stated and MB Receivers are permitted to be disabled. SB initialization procedure is performed at 800 MT/s with 800 MHz sideband clock

Advanced Package interface has redundant SB clock and SB data Lanes (**`DATASBRD`**, **`CKSBRD`**) in addition to **`DATASB`** and **`CKSB`**. SBINIT sequence for **Advanced Package interface** where interconnect repair may be needed is as follows:

1. The UCIe Module must start and continue to send iterations of a 64 UI clock pattern and 32 UI low on both sideband data Transmitters (**`TXDATASB`** and **`TXDATASBRD`**). The UCIe Module must send strobes on both **`TXCKSB`** and **`TXCKSBRD`** during active data transmission and gated otherwise.
2. UCIe Module Partner must sample each incoming data patterns on its sideband Receivers with both incoming sideband clocks (this forms four Receiver/clock combinations).
3. A sideband data-clock Receiver combination detection is considered successful if two consecutive iterations of pattern in step 1 are detected.
4. If a UCIe Module Partner detects the pattern successfully on at least one of its sideband data-clock Receiver combination, it must stop sending data and clock on its sideband Transmitters after four more iterations of 64 UI clock pattern and 32 UI low. This will allow for any time differences in both UCIe Module and UCIe Module Partner coming out of RESET state. The sideband Transmitter and Receiver on the UCIe Module must now be enabled to send and receive sideband messages
5. If pattern is not detected on its sideband Receiver, the UCIe Module must continue to send the pattern on its sideband Transmitters for 1ms and idle for 1ms for 8 ms. The sideband Receiver of the UCIe Module must remain enabled during this time. Timeout occurs after 8ms and the UCIe Module enters TRAINERROR state.
6. If detection is successful on more than one sideband data/clock combination, the device can pick a combination based on a priority order. Pseudocode for sideband assignment:

```
CKSB sampling DATASB = Result[0] # 1: Detected; 0: Not detected
CKSBRD sampling DATASB = Result[1] # 1: Detected; 0: Not detected
CKSB sampling DATASBRD = Result[2] # 1: Detected; 0: Not detected
CKSBRD sampling DATASBRD = Result[3] #  1: Detected; 0: Not detected

IF (Result[3:0] == XXX1):
       Sideband = (DATASB/CKSB)
ELSE IF (Result[3:0] == XX10):
       Sideband = (DATASB/CKSBRD)
ELSE IF (Result[3:0] == X100):
       Sideband = (DATASBRD/CKSB)
ELSE IF (Result[3:0] ==1000):
       Sideband = (DATASBRD/CKSBRD)
Else:
       Sideband is not functional
```

7. If the sideband on the UCIe Module is enabled to send and receive sideband messages (Step 4), the UCIe Module must start and continue to send {SBINIT Out of Reset} sideband message on both **`TXDATASB`** and **`TXDATASBRD`** while sending both **`TXCKSB`** and **`TXCKSBRD`** until it detects the same message in its sideband Receivers or a time out occurs at 8 ms.
8. If {SBINIT Out of Reset} sideband message detection is successful on its sideband Receivers, the UCIe Module stops sending the sideband message. Before sending any further sideband messages, both UCIe Module and UCIe Module Partner must apply Sideband Data/Clock assignment (called the functional sideband) based on the information included in the {SBINIT Out of Reset} sideband message.

9. Any further sideband messages must be sent and received on the functional sideband. Any sideband message exchange can now be performed.
10. The UCIe Module sends the sideband message {SBINIT done req} and waits for a response*.* If this message is received successfully, UCIe Module Partner responds with {SBINIT done resp}. When a UCIe Module has sent and received {SBINIT done resp} it must exit to MBINIT

The next state is main band initialization (MBINIT) if sideband message exchange is successful

SBINIT sequence for **Standard Package interface** where interconnect Lane redundancy and repair are not supported is as follows:

1. The UCIe Module must start and continue to send iterations of 64 UI clock pattern and 32 UI low on its sideband Transmitter (**TXDATASB**). The UCIe Module must send strobe on its sideband clock(**TXCKSB)** during active data duration and gated otherwise
2. The UCIe Module partner must sample incoming data pattern with incoming clock.
3. Sideband pattern detection is considered successful if two consecutive iterations of pattern is step 1 are detected
4. If the UCIe Module the pattern successfully, it stops sending data and clock on its sideband Transmitters after four more iterations of pattern in step 1. This will allow for any time differences in both UCIe Modules coming out of RESET. The UCIe Module sideband Transmitter and Receiver must now be enabled to send and receive sideband messages.
5. If pattern is not detected on its sideband Receiver, the UCIe Module continues to send the pattern on its Transmitters for 1ms and idle for 1ms for 8 ms. The sideband Receiver must be enabled during this time. Timeout occurs after 8ms and the UCIe Module must exit to TRAINERROR. If a pattern is detected successfully at any time, as described in Step 3, the UCIe Module enables sideband message transmission as described in Step 4 and starts sending sideband message (Step 6)
6. Once sideband detection is successful (Step 5), the UCIe Module must start and continue to send {SBINIT Out of Reset} sideband message on **TXDATASB** while sending **TXCKSB** until it detects the same message in its sideband Receivers or a timeout occurs.
7. If {SBINIT Out of Reset} sideband message detection is successful, the UCIe module must stop sending the message. Any physical layer sideband message exchange can now be performed.
8. The UCIe Module must the sideband message {SBINIT done req}*.* If this message is received successfully, each UCIe Module Partner responds with {SBINIT done resp}. When the UCIe Module has sent and received {SBINIT done resp} it must exit to MBINIT

The next state is main band initialization (MBINIT) if sideband message exchange is successful.

4.5.3.3 MBINIT

In this state, the main band (MB) interface is initialized and repaired or degraded (when applicable). The data rate on the main band is set to the lowest supported data rate (4 GT/s)

For **Advanced Package interface** interconnect repair may be needed. Sub-state in MBINIT allows detection and repair of data, clock, track and valid Lanes. For **Standard Package interface**, where no Lane repair is needed. Sub-states is used to check functionality at lowest data rate and width degrade if needed.

4.5.3.3.1 MBINIT.PARAM

This state is used to perform exchange of parameters required to setup the maximum negotiated speed and other PHY settings. The following parameters are exchanged with UCIe Module partner:

1. "Voltage swing": The five bit value indicates the Transmitter voltage swing to the UCIe Module partner. The UCIe Module Partner must use this value and its Receiver termination information to set the reference voltage (Vref) for its Receivers. The corresponding bits in {MBINIT.PARAM configuration resp} are reserved.
2. "Maximum Data Rate": The four bit value indicates the Maximum supported Data rate to the UCIe module Partner. This value must take into consideration all the required features at the data rate (BER, CRC/Retry, quadrature clock phase support etc.). The UCIe Module Partner must compare this value with its supported maximum data rate and must respond with the maximum common data rate encoding in {MBINIT.PARAM configuration resp}. For example, a UCIe Module is 8 GT/s capable while the UCIe Module Partner advertises 16 GT/s, the UCIe module must pick 8 GT/s and send it back in response.
3. "Clock Mode": The one bit value indicates the UCIe Module's request to UCIe module partner for a strobe or continuous clock. The UCIe Module Partner must use this information to setup the clock mode on its clock Transmitter. {MBINIT.PARAM configuration resp} must reflect the same value.
4. "Clock Phase": The one bit value indicates the UCIe Module's request to UCIe Module Partner for the Clock Phase support on UCIe Module's forwarded clock. This should only be set 1b if the maximum data rate advertised is permitted to do so (refer to Table 35). The corresponding bit in {MBINIT.PARAM configuration resp} must be set to 1b if this was requested and the operational data rate allows it.
5. "Module ID": The UCIe module sends its "Module ID". This can be used by the UCIe Module Partner if in a multi-module configuration for Byte mapping, Module enable disable information etc. The corresponding bits in {MBINIT.PARAM configuration resp} are reserved

Following is the sequence for parameter exchange:

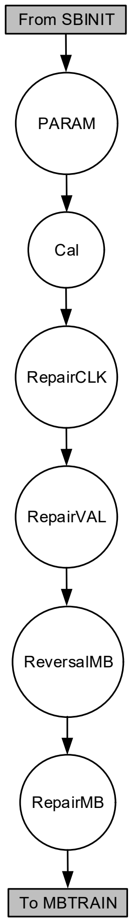
1. The UCIe Module sends sideband message {MBINIT.PARAM configuration req} to exchange parameters with the UCIe Module Partner.
2. Once {MBINIT.PARAM configuration req} is received, the UCIe Module Partner resolves and responds with {MBINIT.PARAM configuration resp} sideband message.

4.5.3.3.2 MBINIT. CAL

This state is used to perform any calibration needed (example: Tx Duty Cycle correction, Receiver offset and Vref calibration). This state is common for both **Advanced Package interface** and **Standard Package interface**

1. The UCIe Module must tri-state all its Transmitter pins in this state and is permitted to perform any required in Transmitter and Receiver calibration.
2. THe UCIe Module must send the sideband message {MBINIT.CAL Done req} and waits for a response. If this message is received successfully, the UCIe Module Partner responds with {MBINIT.CAL Done resp}. Once the UCIe Module has sent and received {MBINIT.CAL Done resp} it must exit to MBINIT. REPAIRCLK

Figure 70. MBINIT: Main band initialization and repair flow



4.5.3.3.3 MBINIT.REPAIRCLK

This sub-state is used to detect and apply repair (if needed) to clock and track Lanes for **Advanced Package interface** and for functional check of clock and track Lanes for **Standard Package interface**.

Following is the sequence for **Advanced Package interface**:

Clock repair mapping is described in section 4.3. Each clock, track and clock redundant physical Lanes (**`TCKP_P/RCKP_P`**, **`TCKN_P/RCKN_P`**,**`TTRK_P/RTRK_P`** and **`TCKRD_P/ RCKRD_P`**) is independently checked to detect possible electrical opens or electrical short between the two clock pins. single-ended clock Receivers or independent detection mechanism is required to ensure clock repair. The UCIe Module must enable Transmitters and Receivers on Clock, Track and Redundant clock Lanes. All other Transmitters are permitted to remain in tri-state.

1. The UCIe Module sends the sideband message {MBINIT.REPAIRCLK init req} and waits for a response. The UCIe Module Partner when ready to receive pattern on **`RCKP_L, RCKN_L, RTRK_L, RCKRD_L`** responds with {MBINIT.REPAIRCLK init resp}
2. The UCIe Module must now send 128 iterations of clock repair pattern (16 clock cycles followed by 8 cycles of low) on its **`TCKP_P`** only. (**`TCKN_P, TTRK_L`** and **`TCKRD_L`** are tri-stated). Clock repair pattern must not be scrambled.

3. The UCIe Module Partner detects this pattern on **RCKP_L**, **RCKN_L**, **RTRK_L** and **RCKRD_L**. Detection is considered successful if at least 16 consecutive iterations of clock repair pattern are detected. The UCIe Module Partner logs the detection result for both **RCKP_L**, **RCKN_L**, **RTRK_L** and **RCKRD_L**.
4. The UCIe Module after completing pattern transmission sends {MBINIT.REPAIRCLK result req} sideband message to get the logged result and waits for a response.
5. The UCIe Module Partner responds with {MBINIT.REPAIRCLK result resp} sideband message with log result of **RCKP_L**, **RCKN_L**, **RTRK_L** and **RCKRD_L**. If detection is successful on **RCKP_L** only and not on any of **RCKN_L**, **RTRK**, **RRDCK_L**, no repair is needed. Else if detection is successful on any two of **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**, an electrical short is implied.Else if detection is not successful on all of **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**, repair is needed on the physical Lane **TCKP_P/RCKP_P**.
6. After receiving the {MBINIT.REPAIRCLK result resp} sideband message, the UCIe Module sends the sideband message {MBINIT.REPAIRCLK init req} and waits for a response. The UCIe Module Partner when ready to receive pattern on **RCKP_L**, **RCKN_L**, **RTRK_L**, **RRDCK_L** responds with {MBINIT.REPAIRCLK init resp}.
7. After receiving the {MBINIT.REPAIRCLK init resp} sideband message, the UCIe Module must send 128 iterations of clock repair pattern (16 clock cycles followed by 8 cycles of low) on its **TCKN_L** only. (**TCKP_L**, **TTRK_L** and **TRDCK_L** are tri-stated)
8. The UCIe Module Partner detects this pattern on all **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**. Detection is considered successful if at least 16 consecutive cycles of clock repair pattern are detected. The UCIe Module Partner logs the detection result for **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**.
9. The UCIe Module after completing the pattern transmission, sends {MBINIT.REPAIRCLK result req} sideband message to get the logged result.
10. The UCIe Module Partner on receiving it responds with {MBINIT.REPAIRCLK result resp} sideband message with logged result of **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**. If detection is successful on **RCKN_L** and not on **RCKP_L**, **RTRK_L**, **RRDCK_L**, no repair is needed. Else if detection is successful on any two of **RCKN_L**, **RCKP_L**, **RTRK_L** and **RRDCK_L**, an electrical short is implied. Else if detection is not successful on any of **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**, repair is needed on the physical Lane **TCKN_P/RCKN_P**.
11. After receiving the {MBINIT.REPAIRCLK result resp} sideband message, the UCIe Module sends the sideband message {MBINIT.REPAIRCLK init req}. The UCIe Module Partner when ready to receive pattern on **RCKP_L**, **RCKN_L**, **RTRK_L**, **RRDCK_L** responds with {MBINIT.REPAIRCLK init resp}.
12. After receiving the {MBINIT.REPAIRCLK init resp} sideband message, the UCIe Module sends 128 iterations of clock repair pattern (16 clock cycles followed by 8 cycles of low) on **TRDCK_L** only. (**TCKP_L**, **TTRK_L** and **TCKN_L** tri-stated)
13. The UCIe Module Partner detects this pattern on all**RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**. Detection is considered successful if at least 16 consecutive cycles of clock repair pattern are detected. The UCIe Module Partner logs the detection result for **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**.
14. The UCIe Module device after completing the pattern transmission sends {MBINIT.REPAIRCLK result req} sideband message to get the logged result.
15. The UCIe Module Partner responds with {MBINIT.REPAIRCLK result resp} sideband message with logged result of **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**. If detection is successful only on **RRDCK_L** and not on **RCKP_L**, **RTRK_L** **RCKN_L**, no repair is needed. Else if detection is successful on any two of **RCKN_L**, **RCKP_L**, **RTRK_L** and **RRDCK_L**, an electrical short is implied. Else if detection is not successful on any of **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**, the physical Lane**TCKRD_P/RCKRD_P** is not available as a repair resource.

16. After receiving the {MBINIT.REPAIRCLK result resp} sideband message, the UCIe Module sends the sideband message {MBINIT.REPAIRCLK init req} and waits for a response. The UCIe Module Partner when ready to receive pattern on **RCKP_L**, **RCKN_L**, **RTRK_L**, **RRDCK_L** responds with {MBINIT.REPAIRCLK init resp}.
17. After receiving the {MBINIT.REPAIRCLK init resp} sideband message, the UCIe Module sends 128 iterations of clock repair pattern (16 clock cycles followed by 8 cycles of low) on TTRK_L only. (**TCKP_L**, **TCKN_L** and **TRDCK_L** are tri-stated)
18. The UCIe Module Partner detects this pattern on all **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**. Detection is considered successful if at least 16 consecutive cycles of clock repair pattern are detected. The UCIe Module Partner logs the detection result for **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**.
19. The UCIe Module after completing pattern transmission sends {MBINIT.REPAIRCLK result req} sideband message to get the logged result and waits for a response.
20. The UCIe Module Partner stops comparison and responds with {MBINIT.REPAIRCLK result resp} sideband message with logged result of **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**. If detection is successful only on RTRK_L and not on **RCKP_L**, **RCKN_L**, **RRDCK_L**, no repair is needed. Else if detection is successful on any two of **RCKN_L**, **RCKP_L**, **RTRK_L** and **RRDCK_L**, an electrical short is implied. Else if detection is not successful on any of **RCKP_L**, **RCKN_L**, **RTRK_L** and **RRDCK_L**, repair is needed on the physical Lane **TTRK_P/RTRK_P**.
21. If required,the UCIe Module applies repair on its clock/track Transmitter sends sideband message {MBINIT.REPAIRCLK apply repair req} with repair information (including no repair, and unrepairable). If a repair is needed for one of the clock or track pins (**CKP** or **CKN** or **TRK**) repair is applied as described in section 4.3. The UCIe Module Partner applies repair and sends sideband message {MBINIT.REPAIRCLK apply repair resp}. If the sideband is unrepairable (If repair is needed on any two of **RCKP_L**, **RCKN_L** and **RRDCK_L** or electrical short is detected the interconnect cannot be repaired) the UCIe Module and UCIe Module partner must exit to TRAINERROR after performing TRAINERROR handshake (section 4.5.3.8).
22. If a repair is applied, UCIe Module must check the repair success by applying clock repair pattern and checking on the Receiver.
    a. The UCIe Module sends sideband message {MBINIT.REPAIRCLK check repair init req} to initiate check repair and waits for a response. The UCIe Module Partner responds with sideband message {MBINIT.REPAIRCLK check repair init resp} and is ready to receive and check clock repair pattern.
    b. After receiving the {MBINIT.REPAIRCLK check repair init resp} sideband message, the UCIe Module sends 128 iterations of clock repair pattern (16 clock cycles followed by 8 cycles of low) on TCKP_L. The UCIe Module Partner detects this pattern **RCKN_L**, **RCKP_L**, **RTRK_L**. Detection is considered successful if at least 16 consecutive cycles of clock repair pattern are detected. The UCIe Module requests for check result request using the sideband message {MBINIT.REPAIRCLK check results req} and the UCIe Module Partner responds with the sideband message {MBINIT.REPAIRCLK check results resp}. Repair is considered successful if pattern is detected only on **RCKP_L** only. If repair is not successful, the UCIe Module ane UCIe Module Partner must exit to TRAINERROR after performing TRAINERROR handshake (section 4.5.3.8).
    c. Steps Step a and Step b are repeated for **TCKP_L** and **TTRK_L**
23. If repair is successful or repair is not required, the UCIe Module sends {MBINIT.RepairCLK done req} sideband message and the UCIe Module Partner responds with {MBINIT.RepairCLK done resp} sideband message. When the UCIe Module has sent and received {MBINIT.RepairCLK done resp}, it must exit to REPAIRVAL.

For **Standard Package interface**, clock and track Lanes are checked for functional operation at the lowest data rate. The sequence is as follows:

1. The UCIe Module sends the sideband message {MBINIT.REPAIRCLK init req} and wait for a response. When ready to receive pattern on **RCKP_L**, **RCKN_L**, **RTRK_L**, the UCIe Module Partner responds with {MBINIT.REPAIRCLK init resp}. On receiving the sideband message {MBINIT.REPAIRCLK init resp}, the UCIe Module sends 128 iterations of clock repair pattern (16 clock cycles followed by 8 cycles of low) on **TCKP_L**, **TCKN_L**, **TTRK_L**. Clock repair pattern must not be scrambled.
2. The UCIe Module Partner detects this pattern on both **RCKP_L**, **RCKN_L**, **RTRK_L** and. Detection is considered successful if at least 16 consecutive cycles of clock repair pattern are detected. The UCIe Module Partner logs the detection result for **RCKP_L**, **RCKN_L**, **RTRK_L**.
3. After completing pattern transmission, the UCIe Module sends {MBINIT.REPAIRCLK result req} sideband message to get the logged result.
4. The UCIe Module Partner stops comparison and responds with {MBINIT.REPAIRCLK result resp} sideband message with logged result of **RCKP_L**, **RCKN_L** and **RTRK_L**.
5. If detection is not successful on any one of **RCKP_L**, **RCKN_L and RTRK_L**, the UCIe Module and UCIe Module partner must exit to TRAINERROR after performing TRAINERROR handshake.
6. If detection is successful, the UCIe Module sends {MBINIT.RepairCLK done req} sideband message and the UCIe Module Partner responds with {MBINIT.RepairCLK done resp} sideband message. When the UCIe Module has sent and received the sideband message {MBINIT.RepairCLK done resp}, it must exit to REPAIRVAL.

4.5.3.3.4 MBINIT.REPAIRVAL

The UCIe Module sets the clock phase at the center of the data UI on its mainband Transmitter. The UCIe Module Partner must sample the received Valid with the received forwarded Clock. All Data Lanes must be held at low during this state

This state can be used to detect and apply repair (if needed) to Valid Lane for **Advanced Package interface** and for functional check of Valid for **Standard Package interface**.

Following is the sequence for **Advanced Package interface**:

1. The UCIe Module sends the sideband message {MBINIT.REPAIRVAL init req} and wait for a response. When ready to receive pattern on **RVLD_L** and **RRDVLD_L**, the UCIe Module Partner clears any previous comparison results and responds with {MBINIT.REPAIRVAL init resp}.
2. The UCIe Module on receiving {MBINIT.REPAIRVAL init resp} must send 128 iterations of VALTRAIN pattern (four 1's followed by four 0's) on TVLD_L along with the forwarded clock. VALTRAIN pattern must not be scrambled.
3. The UCIe Module Partner detects pattern on **RVLD_L** and **RRDVLD_L**. Detection is considered successful if at least 16 consecutive iterations of VALTRAINpattern are detected. The Receiver logs the detection result for **RVLD_L** and **RRDVLD_L**
4. After completing the pattern transmission, the UCIe Module must send {MBINIT.REPAIRVAL result req} sideband message and wait to get the logged result.
5. The UCIe Module Partner must respond with {MBINIT.REPAIRVAL result resp} sideband message with result in the previous step. If detection is successful on **RVLD_L** only and not on **RRDVLD_L**, no repair is needed. If detection is successful on both **RVLD_L** and **RRDVLD_L** or only on **RRDVLD_L** , the interconnect cannot be

repaired. If detection is not successful on both **RVLD_L** and **RRDVLD_L**, Valid Lane (**TVLD_P/RVLD_P**) needs repair.

6. After receiving {MBINIT.REPAIRVAL result resp}, Step 1 must be repeated.
7. The UCIe Module sends 128 iterations of VALTRAIN repair pattern (four 1's followed by four 0's) on **TRDVLD_L** along with the forwarded clock. VALTRAIN pattern must not be scrambled.
8. The UCIe Module Partner detects pattern on **RVLD_L** and **RRDVLD_L**. Detection is considered successful if at least 16 consecutive iterations of VALTRAIN pattern are detected. The Receiver logs the detection result for **RVLD_L** and **RRDVLD_L**
9. After completing the pattern transmission, the UCIe Module must send {MBINIT.REPAIRVAL result req} sideband message to get the logged result.
10. The UCIe Module Partner must stop comparison and respond with {MBINIT.REPAIRVAL result resp} sideband message with result in the previous step. If detection is successful on **RRDVLD_L** only and not on **RVLD_L**, is available for repair. If detection is successful on both **RVLD_L** and **RRDVLD_L** or only on **RVLD_L** , the interconnect cannot be repaired. If detection is not successful on both **RVLD_L** and **RRDVLD_L**, **TRDVLD_P/RRDVLD_P** is not available and cannot be used for Valid Lane repair.
11. If repair is required on (**TVLD_P/RVLD_P**) and repair resource is available, the UCIe module applies repair on its Valid Transmitter (section 4.3.6) and sends sideband message {MBINIT.REPAIRVAL apply repair req} with repair information (including no repair, and unrepairable). The UCIe Module Partner after receiving the message must apply repair on its Valid Receiver and must respond with sideband message {MBINIT.REPAIRVAL apply repair resp}.
12. If a repair is applied, device must check the repair success by repeating step 1 - step 4. If repair is successful or repair is not required, the UCIe Module must send {MBINIT.RepairVAL done req} sideband message and the UCIe Module Partner must respond with {MBINIT.RepairVAL done resp}. When a UCIe Module has sent and received {MBINIT.RepairVAL done resp} sideband message, it must exit to REPAIRMB.
13. Else if repair is unsuccessful or Valid Lane is unrepairable (in Step 11), the UCIe module must exit to TRAINERROR after completing the TRAINERROR handshake.

For **Standard Package interface**, Valid Lane is checked for functional operation at the lowest data rate. Following is the flow:

1. The UCIe Module must send the sideband message {MBINIT.REPAIRVAL init req} and wait for a response The UCIe Module Partner when ready to receive pattern on **RVLD_L**, must respond with {MBINIT.REPAIRVAL init resp}.
2. After receiving the sideband message {MBINIT.REPAIRVAL init resp}, the UCIe Module sends 128 iterations of VALTRAIN pattern (four 1's followed by four 0's) on **TVLD_L** along with the forwarded clock.
3. The UCIe Module Partner detects this pattern on **RVLD_L**. Detection is considered successful if at least 16 consecutive iterations of valid repair pattern are detected. The Receiver logs the detection result for **RVLD_L**
4. After completing pattern transmission, the UCIe Module must send {MBINIT.REPAIRVAL result req} sideband message and wait to get the logged result.
5. The UCIe Module Partner must stop comparison and respond with {MBINIT.REPAIRVAL result resp} sideband message with result in the previous step
6. If detection fails, the UCIe Module must exit to TRAINERROR after completing the TRAINERROR handshake.

7. If detection is successful, the UCIe Module must send {MBINIT.RepairVAL done req} sideband message and the UCIe Module Partner responds with {MBINIT.RepairVAL done resp}. When a UCIe Module has sent and received {MBINIT.RepairVAL done resp} sideband message, it must exit to REVERSALMB.

4.5.3.3.5 MBINIT.REVERSALMB

This state is entered only if Clock and Valid Lanes are functional. In this state Data Lane reversal is detected. All the Transmitters and Receivers are enabled. The UCIe Module sets the forwarded clock phase at the center of the data UI on its mainband. The UCIe Module Partner must sample the incoming Data with the incoming forwarded clock.

A 16-bit "Per Lane ID" pattern, shown in Table 23, is a Lane specific pattern using the Lane ID described in section 4.2.1. Example of "Per Lane ID" pattern for Lane 0 and Lane 31 are shown in Table 24. When "Per Lane ID" pattern is used, it must not be scrambled.

Table 23. Per Lane ID pattern pattern

Pattern	0	1	0	1	Lane ID (LSB first)								0	1	0	1
bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Table 24. Per Lane ID pattern pattern examples

Lane 1	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	1
Lane 31	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	1
bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Following is the Reversal MB sequence for **Advanced Package interface** and **Standard Package interface**:

1. The UCIe Module must send the sideband message {MBINIT.REVERSALMB init req}*.* When ready to receive "Per Lane ID" pattern and perform per Lane pattern comparison, the UCIe Module Partner must respond with {MBINIT.REVERSALMB init resp}*.*
2. On Receiving the {MBINIT.REVERSALMB init resp} sideband message the UCIe Module must send {MBINIT.REVERSALMB clear error req} sideband message. Upon receiving this message, the UCIe Module partner clears any prior errors and responds with {MBINIT.REVERSALMB clear error resp}. After receiving {MBINIT.REVERSALMB clear error resp}, the UCIe Module sends 128 iterations of Per Lane ID pattern (LSB first) on all N data Lanes with correct Valid framing on the Valid Lane (section 5.11, section 4.1.2) along with the forwarded clock. N is 68 (64 Data + 4 RD) for Advanced package interface and 16 for Standard Package interface
3. The UCIe Module Partner must perform a per Lane compare on its Receivers on all N Lanes (section 4.4). Detection on a Lane is considered successful if at least 16 consecutive iterations of "Per Lane ID" pattern are detected. The UCIe Module Partner logs the detection result for its receive Lanes to be used for Lane reversal Detection
4. After sending 128 iterations of "Per Lane ID" pattern, the UCIe Module stops sending the pattern and sends {MBINIT.REVERSALMB result req} sideband message to get the logged result.
5. The UCIe Module Partner stops comparison and must respond with {MBINIT.REVERSALMB result resp} sideband message with N-bit (68 for Advanced and 16 for Standard Package interface) per Lane result.

6. If majority of the Lanes show success (since some Lanes may need repair), Lane reversal is not needed. Skip to Step 11.
7. Else if results from Step 5 show majority of Lanes are unsuccessful, the UCIe Module applies Lane reversal on its Transmitters.
8. Following the Lane reversal application on its Transmitters, the UCIe module repeats Step 2 through Step 5..
9. If majority of Lanes show success, the Lane reversal is needed. If applied, Lane reversal preserved for rest of the device operation. Skip to step 11
10. Else if results from step 8 show majority Lanes are unsuccessful, the UCIe Module must exit to TRAINERROR after completing the TRAINERROR handshake.
11. The UCIe Module must send {MBINIT.ReversalMB done req} sideband message and the UCIe Module Partner responds with {MBINIT.RversalMB done resp}. Whenthe UCIe Module has sent and received {MBINIT.ReversalMB done resp} sideband message, it must exit to REPAIRMB.

4.5.3.3.6 MBINIT.REPAIRMB

This state is entered only after Lane reversal detection and application is successful. All the Transmitters and Receivers on a UCIe Module are enabled. The UCIe Module sets the clock phase the center of the data UI for its mainband. The UCIe Module Partner must sample the incoming Data with the incoming forwarded clock on its mainband Receivers.

In this state, the main band Lanes are detected and repaired if needed for **Advanced Package interface** and for functional check and width degrade for **Standard Package interface**.

Following is the sequence for **Advanced Package interface**:

1. The UCIe Module must send the sideband message {MBINIT.REPAIRMB start req} and waits for a response. The UCIe Module Partner responds with {MBINIT.REPAIRMB start resp}. .
2. The UCIe Module device performs Transmitter initiated data to clock point training as described in section 4.5.1.1 on its Transmitter Lanes.
   a. The transmit pattern must be set up to send 128 iterations of continuous mode “Per Lane ID” Pattern. "Per Lane ID" pattern must not be scrambled. The Receiver must be setup to perform per Lane comparison
   b. Detection on a Receiver Lane is considered successful if at least 16 consecutive iterations of “Per Lane ID “pattern are detected.
   c. LFSR Reset has no impact in MBINIT.REPAIRMB
3. The UCIe Module receives the per-Lane pass/fail information over sideband message at the end of Transmitter initiated data to clock point test Step 2.
4. If Lane repair is required and repair resources are available, the UCIe Module applies repair on its mainband Transmitters as described in section 4.3.1 and sends {MBINIT.REPAIRMB Apply repair req} sideband message. Upon receiving this sideband message, the UCIe Module Partner applies repair on its mainband Receivers as described in section 4.3.1. and sends {MBINIT.REPAIRMB Apply repair resp} sideband message . Else if the number of Lane failures are more than repair capability (section 4.3), the mainband is unrepairable and the UCIe Module must exit to TRAINERROR after performing the TRAINERROR handshake.
5. If repair is not required, perform step 7.
6. If Lane repair is applied (step 4), the applied repair is checked by UCIe Module by repeating step 2 and step 3. If post repair Lane errors are logged in step 5, the

UCIe Module must exit to TRAINERROR after performing the TRAINERROR handshake. If repair is successful, perform step 7.

The UCIe Module sends {MBINIT.REPAIRMB end req} sideband message and the UCIe Module Partner responds with {MBINIT.REPAIRMB end resp}. When UCIe Module has sent and received {MBINIT.REPAIRMB end resp}, it must exit to MBTRAIN. For **Standard Package interface**, main band is checked for functional operation at the lowest data rate. Following is the sequence of steps:

1. The UCIe Module sends the sideband message {MBINIT.REPAIRMB init req} and waits for a response. The UCIe Module Partner responds with {MBINIT.REPAIRMB init resp} when ready to receive the pattern on its mainband Receivers. .
2. The UCIe Module performs Transmitter initiated data to clock point training as described in section 4.5
   a. The transmit pattern must be set up to send 128 iterations of continuous mode "Per Lane ID" Pattern. The Receiver must be setup to perform per Lane comparison
   b. Detection on a Receiver Lane is considered successful if at least 16 consecutive iterations of "Per Lane ID "pattern are detected.
   c. LFSR Reset has no impact in MBINIT.REPAIRMB.
3. The UCIe Module receives the per-Lane pass/fail information over sideband message at the end of Transmitter initiated data to clock point test.
4. If the check is successful, the UCIe Module sends {MBINIT.REPAIRMB end req} sideband message and the UCIe Module Partner responds with {MBINIT.REPAIRMB end resp}. When the UCIe Module has sent and received {MBINIT.REPAIRMB end resp}, it must exit to MBTRAIN .
5. Else if the check fails, and width degrade is possible (section 4.3.6), the UCIe Module with faulty Transmitter Lanes must apply degrade (to both its Transmitter and Receiver) and send {MBINIT.REPAIRMB apply degrade req} including the logical Lane map shown in Table 25 to the remote Link partner. The UCIe Module Partner must apply degrade (to both its Transmitter and Receiver) and send {MBINIT.REPAIRMB apply degrade resp}. The UCIe Module must repeat from Step 2 above.
6. Else, the UCIe Module must exit to TRAINERROR after performing the TRAINERROR handshake.

Table 25. Standard Package Logical Lane map

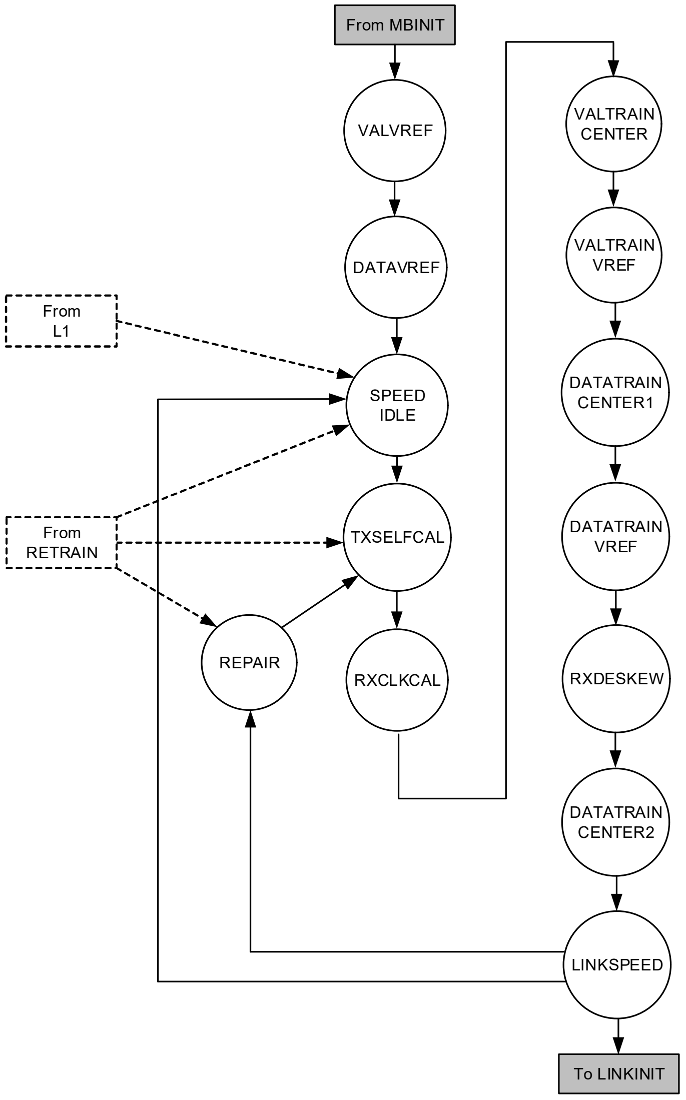
Lane map code	Functional Lanes
00b	None (Degrade not possible)
01b	Logical Lanes 0 to 7
10b	Logical Lanes 8 to 15
11b	0 - 15

4.5.3.4 MBTRAIN

MBTRAIN state is used at setup operational speed and perform clock to data centering. At higher speeds, additional calibrations like Rx clock correction, Tx and Rx de-skew may be needed to ensure Link performance. MBTRAIN allows sub-states to perform all the required calibration and training. UCIe Modules must enter each sub-state and exit of each state is through a sideband hand shake. If a particular action within a sub-state is not needed, the UCIe Module is permitted exit it though sideband handshake without performing the described operations in that sub-state.

Devices enter this state once the MBINIT is completed. This state is common for **Advanced** and **Standard Package interfaces**.

Figure 71. Main Band Training



4.5.3.4.1 MBTRAIN.VALVREF

Receiver reference voltage (Vref) to sample the incoming Valid is optimized in this state. The data rate on the main band continues to be at the lowest supported data rate (4 GT/s). The UCIe Module Partner must set the forwarded clock phase the center of the data UI on its main band Transmitters. The UCIe Module must sample the pattern on Valid signal with the forwarded clock. All data Lanes must be held low during Valid Lane reference voltage training.

The sequence for Valid Receiver reference voltage adjustment is as follows:

1. The UCIe Module must send the sideband message {MBTRAIN.VALVREF start req} and wait for a response. When {MBTRAIN.VALVREF start req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.VALVREF start resp}
2. UCIe Module optimizes Vref on its Valid Receiver by adjusting Receiver reference voltage and performing one ore more Receiver initiated point tests (section 4.5.1.3) or Receiver initiated eye width sweeps (section 4.5.1.4).
    a. The transmit pattern must be set to send 128 iterations of continuous mode "VALTRAIN" (four 1's and four zeros) pattern (Table 21). This pattern must not be scrambled. The Receiver must be setup to perform comparison on the Valid Lane
    b. Detection on a Receiver Lane is considered successful if "VALTRAIN" pattern detection errors are less than the set threshold.
    c. It should be noted that LFSR RESET has no impact in MBTRAIN.VALVREF.
3. The UCIe Module must send {MBTRAIN.VALVREF end req} sideband message after the Vref optimization (One way to perform Vref Optimization is to step through Vref and perform step 2 at each setting). When {MBTRAIN.VALVREF end req} is received, the UCIe Module Partner must respond with {MBTRAIN.VALVREF end resp}. When UCIe Module has sent and received the sideband message {MBTRAIN.VALVREF end resp}, it must exit to MBTRAIN.DATAVREF

4.5.3.4.2 MBTRAIN.DATAVREF

Receiver reference voltage (Vref) to sample the incoming data is optimized in this state. The data rate on the UCIe Module main band continues to be at the lowest supported data rate (4 GT/s). The Transmitter sets the forwarded clock phase is set at the center of the data UI

The sequence for data Receiver reference voltage adjustment is as follows:

1. The UCIe Module sends the sideband message {MBTRAIN.DATAVREF start req}. When {MBTRAIN.DATAVREF start req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.DATAVREF start resp}
2. The UCIe Module optimizes data Vref by adjusting data Receiver reference voltage and performing one or more Receiver initiated point tests (section 4.5.1.3) or Receiver initiated eye width sweeps (section 4.5.1.4).
    a. The Transmitter must be setup to send 4K UI of continuous mode LFSR pattern described in section 4.4.1. The LFSR pattern on Data must be accompanied by correct Valid framing on the Valid Lane as described in section 4.1.2. The Receiver must be setup to perform per Lane comparison.
    b. Detection on a Receiver Lane is considered successful if total error count is less than the set error threshold.
3. The UCIe Module must send {MBTRAIN.DATAVREF end req} sideband message after the Vref optimization (One way to perform Vref Optimization is to step through Vref and perform step 2 at each setting). When {MBTRAIN.DATAVREF end req} is received, the UCIe Module Partner must respond with {MBTRAIN.DATAVREF end resp}. Once the UCIe Module has sent and received the sideband message {MBTRAIN.DATAVREF end resp}, it must exit to MBTRAIN.SPEEDIDLE

4.5.3.4.3 MBTRAIN.SPEEDIDLE

This is an electrical idle state to allow frequency change.

1. If this state is entered from MBTRAIN.DATAVREF, the UCIe Module transitions to a data rate based on the highest common speed between the two devices

(section 4.5.3.3.1) . Else if this state is entered from LINKSPEED or PHYRETRAIN, the data rate is picked next lower data rate.

2. Upon completing the transition to the required data rate, the UCIe Module must send {MBTRAIN.SPEEDIDLE done req} sideband message with current Link speed information. When {MBTRAIN.SPEEDIDLE done req} sideband message is received, UCIe Module Partner responds with {MBTRAIN.SPEEDIDLE done resp}. Once the UCIe Module has sent and received the {MBTRAIN.SPEEDIDLE done resp} sideband message it must exit to MBTRAIN.TXSELFCAL.

4.5.3.4.4 MBTRAIN.TXSELFCAL

The UCIe Module calibrates its circuit parameters independent of the UCIe Module Partner. The UCIe module must maintain electrical idle on its Transmitters during this step.

1. UCIe Modules is permitted to perform any required in Transmitter related calibration.
2. Upon completion of calibration, the UCIe module must send the sideband message {MBTRAIN.TXSELFCAL Done req}*.* When {MBTRAIN.TXSELFCAL Done req} message is received , the UCIe Module Partner must respond with {MBTRAIN.TXSELFCAL Done resp}. When the UCIe module has sent and received the {MBTRAIN.TXSELFCAL Done resp} sideband message it must exit to MBINIT. RXCLKCAL.

4.5.3.4.5 MBTRAIN.RXCLKCAL

1. The UCIe Module, when ready to perform calibration on its Clock receive path, sends the sideband message {MBTRAIN.RXCLKCAL start req}*.* When the sideband message {MBTRAIN.RXCLKCAL start req} is received, the UCIe Module Partner starts sending the forwarded clock and sends the sideband message {MBTRAIN.RXCLKCAL start resp}. The Transmitter clock must be free running and all Data Lanes and Valid must be held low. The UCIe Module is permitted to use the forwarded clock to perform any clock path related calibration. The UCIe Module Partner must not adjust any circuit or PI phase parameters on its Transmitters within this state.
2. When the required calibration (if any) is performed, the UCIe Module sends {MBTRAIN.RXCLKCAL done req} message. When {MBTRAIN.RXCLKCAL done req} is received, the UCIe Module Partner stops sending forwarded clock and responds by sending {MBTRAIN.RXCLKCAL done resp} message. When a UCIe Module has sent and received {MBTRAIN.RXCLKCAL done resp} sideband message it must exit to VALTRAINCENTER.

4.5.3.4.6 MBTRAIN.VALTRAINCENTER

To ensure valid signal is functional, valid to clock training is performed before the data Lane training. The Receiver samples the pattern on Valid with the forwarded clock. Receiver reference voltage is set to the optimized value achieved through Vref training (section 4.5.3.4.1/section 4.5.3.4.2). All data Lanes are held low during valid to clock training. Following is the MBTRAIN.VALTRAINCENTER sequence

1. The UCIe Module sends a sideband message {MBTAIN.VALTRAINCENTER start req}. The UCIe Module Partner responds with {MBTAIN.VALTRAINCENTER start resp}.
2. The UCIe Module must perform a Transmitter initiated full eye width sweep (section 4.5.1.2) or one or more Transmitter initiated point tests (section 4.5.1.1) to determine the correct Valid to clock centering.

   a. The transmit pattern must be set to send 128 iterations of continuous mode "VALTRAIN" (four 1's and four zeros) pattern (Table 21). This pattern must not be scrambled. The Receiver must be setup to perform comparison on the Valid Lane
   b. Detection on a Receiver Lane is considered successful if "VALTRAIN" pattern detection errors are less than set threshold
   c. It should be noted that LFSR RESET has no impact in MBTRAIN.VALVREF.
3. The UCIe Module can use the received results log to assess valid functionality and margins. Following this, step 4 must be performed.
4. The UCIe Module must send {MBTRAIN.VALTRAINCENTER done req} sideband message. When {MBTRAIN.VALTRAINCENTER done req} is received the UCIe Module Partner responds with {MBTRAIN.VALTRAINCENTER done resp}. Once the UCIe Module has sent and received {MBTRAIN.VALTRAINCENTER done resp} sideband message, the it must exit to MBTRAIN.DATATRAINVREF.

4.5.3.4.7 MBTRAIN.VALTRAINVREF

UCIe Module is permitted to optionally optimize the reference voltage (Vref) to sample the incoming Valid at the operating data rate. The UCIe Module and UCIe Module Partner must enter this state. The sequence for Valid Receiver reference voltage adjustment is as follows:

1. The UCIe Module must send the sideband message {MBTRAIN.VALTRAINVREF start req}. When {MBTRAIN.VALTRAINVREF start req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.VALTRAINVREF start resp}
2. UCIe Module optionally optimizes Vref by adjusting Receiver reference voltage on its Valid Receiver and performing Receiver initiated eye width sweep (section 4.5.1.4) or one or more Receiver initiated point test (section 4.5.1.3). This is optional and implementation specific.
   a. If Valid centering is performed, the transmit pattern must be set to send 128 iterations of continuous mode "VALTRAIN" (four 1's and four zeros) pattern (Table 21). This pattern must not be scrambled. The Receiver must be setup to perform comparison on the Valid Lane
   b. Detection on a Receiver Lane is considered successful if "VALTRAIN" pattern detection errors are less than set threshold
   c. It should be noted that LFSR RESET has no impact in MBTRAIN.VALVREF.
3. The UCIe Module must send {MBTRAIN.VALTRAINVREF end req} sideband message after the Vref optimization is complete. When {MBTRAIN.VALTRAINVREF end req} is received, the UCIe Module Partner must respond with {MBTRAIN.VALTRAINVREF end resp}. Once the UCIe Module has sent and received the sideband message {MBTRAIN.VALTRAINVREF end resp}, it must exit to MBTRAIN.DATATRAINCENTER

4.5.3.4.8 MBTRAIN.DATATRAINCENTER1

In this state, the UCIe Module performs full data to clock training (including valid). LFSR patterns described in section 4.4.1 must be used in this state. Following is the MBTRAIN.DATATRAINCENTER sequence

1. The UCIe Module sends the sideband message {MBTRAIN.DATATRAINCENTER1 start req}. When {MBTRAIN.DATATRAINCENTER start req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.DATATRAINCENTER1 start resp}.

2. The UCIe Module must perform a Transmitter initiated full eye width sweep (section 4.5.1.2) or one or more point tests (section 4.5.1.1) to determine the correct data to clock centering and adjust Transmitter per-bit deskew (if needed).
    a. The Transmitter must be setup to send 4K UI of continuous mode LFSR pattern described in section 4.4.1. The LFSR pattern on data must be accompanied by correct valid framing on the Valid Lane as described in section 4.1.2. The Receiver must be setup to perform per Lane comparison
    b. Detection on a Receiver Lane is considered successful if total error count is less than the set error threshold
3. If the test is a success, the UCIe Module must set the clock phase to sample the data eye at the optimal point to maximize eye margins. The UCIe Module must send {MBTRAIN.DATATRAINCENTER1 end req} sideband message. When {MBTRAIN.DATATRAINCENTER1 end req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.DATATRAINCENTER1 end resp}. Once the UCIe Module has sent and received the {MBTRAIN.DATATRAINCENTER1 end resp} sideband message, it must exit to MBTRAIN.DESKEW.

Note: It is possible that the eye opening in this step are not be sufficient (test fails) and a per bit deskew may be needed on the Receiver. So, the UCIe module must exit to MBTRAIN.DESKEW.

4.5.3.4.9 MBTRAIN.DATATRAINVREF

UCIe Module is permitted to optionally optimize the reference voltage (Vref) on its data Receivers to optimize sampling of the incoming Data at the operating data rate. The UCIe Module and UCIe Module Partner must enter this state.

The sequence for data Receiver reference voltage adjustment is as follows:

1. The UCIe Module must send the sideband message {MBTRAIN.DATATRAINVREF start req}. When {MBTRAIN.DATATRAINVREF start req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.DATATRAINVREF start resp}
2. UCIe Module optionally optimizes Vref by adjusting Receiver reference voltage and performing either Receiver initiated eye width sweep (section 4.5.1.4) or one or more Receiver initiated point test (section 4.5.1.3). This is optional and implementation specific. If Data Vref optimization is performed:
    a. The Transmitter must be setup to send 4K UI of continuous mode LFSR pattern described in section 4.4.1. The LFSR pattern on data must be accompanied by correct valid framing on the Valid Lane as described in section 4.1.2. The Receiver must be setup to perform per Lane comparison
    b. Detection on a Receiver Lane is considered successful if total error count is less than the set error threshold
3. The UCIe module must send {MBTRAIN.DATATRAINVREF end req} sideband message after the Vref optimization is complete. When {MBTRAIN.DATATRAINVREF end req} is received, the UCIe Module Partner responds with {MBTRAIN.DATATRAINVREF end resp}. Once UCIe Module has sent and received the sideband message {MBTRAIN.DATATRAINVREF end resp}, it must exit to MBTRAIN.RXDESKEW

4.5.3.4.10 MBTRAIN.RXDESKEW

The UCIe Module is permitted to optionally perform per Lane deskew on its Receivers to improve timing margin in this state. The UCIe Module and UCIe Module Partner must enter this state.

1. TheUCIe Module must send the sideband message {MBTRAIN.RXDESKEW start req}. When {MBTRAIN.RXDESKEW start req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.DESKEW start resp}
2. UCIe Module optionally performs per Lane deskew on its Receivers by a Receiver initiated full eyed width sweep (section 4.5.1.4) or one or more Receiver initiated point tests (section 4.5.1.3). This is optional and implementation specific. If per Lane deskew is performed,
    a. The Transmitter must be setup to send 4K UI of continuous mode LFSR pattern described in section 4.4.1. The LFSR pattern on data must be accompanied by correct valid framing on the Valid Lane as described in section 4.1.2. The Receiver must be setup to perform per Lane comparison
    b. Detection on a Receiver Lane is considered successful if total error count is less than the set threshold
3. The UCIe Module must send {MBTRAIN.RXDESKEW end req} sideband message after the deskew is performed or skipped. When {MBTRAIN.RXDESKEW end req} is received, the UCIe Module Partner must respond with {MBTRAIN.RXDESKEW end resp}. Once UCIe Module has sent and received the sideband message {MBTRAIN.RXDESKEW end resp}, it must exit to MBTRAIN.DATATRAINCENTER

4.5.3.4.11 MBTRAIN.DATATRAINCENTER2

This state is needed for the UCIe Module to recenter clock to aggregate data in case the UCIe Module Partner's Receiver performed a per Lane deskew.

Following is the MBTRAIN.DATATRAINCENTER sequence

1. The UCIe Module sends the sideband message {MBTRAIN.DATATRAINCENTER2 start req}. When {MBTRAIN.DATATRAINCENTER start req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.DATATRAINCENTER2 start resp}.
2. The UCIe Module must perform a Transmitter initiated full eye width sweep (section 4.5.1.2) or one or more Transmitter initiated point tests (section 4.5.1.1) to determine the correct data to eye centering.
    a. The Transmitter must be setup to send 4K UI of continuous mode LFSR pattern described in section 4.4.1. The LFSR pattern on data must be accompanied by correct valid framing on the Valid Lane as described in section 4.1.2. The Receiver must be setup to perform per Lane comparison
    b. Detection on a Receiver Lane is considered successful if total error count is less than the set error threshold.
3. The UCIe Module uses the received training results to calculate the final eye center and set the clock phase to sample the data eye at the optimal point to maximize eye margins. The UCIe Module must the send {MBTRAIN.DATATRAINCENTER2 end req} sideband message. When {MBTRAIN.DATATRAINCENTER2 end req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.DATATRAINCENTER2 end resp}. Once UCIe Module has sent and received {MBTRAIN.DATATRAINCENTER2 end resp} sideband message, it must exit to MBTRAIN.LINKSPEED.

4.5.3.4.12 MBTRAIN.LINKSPEED

In this state the UCIe Module checks Link stability at the operating date rate. The following Steps must be executed sequentially.

1. The UCIe Module sends the sideband message {MBTRAIN.LINKSPEED start req}. When {MBTRAIN.TXTRAIN start req} sideband message is received, the UCIe Module Partner responds with {MBTRAIN.LINKSPEED start resp}.
2. The UCIe Module must perform point test (section 4.5.1.1) with the final clock sampling phase calculated in the previous MBTRAIN.DATACENTER2 state. LFSR pattern described in section 4.4.1 must be used in this state.
   a. The Transmitter must be setup to send 4K UI of continuous mode LFSR pattern described in section 4.4.1. The LFSR pattern on data must be accompanied by correct valid framing on the Valid Lane as described in section 4.1.2. The Receiver must be setup to perform per Lane comparison
   b. Detection on a Receiver Lane is considered successful if total error count is less than the set threshold.
3. For single Module instantiations, if errors are encountered, the UCIe Module sets its Transmitters to an electrical idle state and sends {MBTRAIN.LINKSPEED error req} sideband message. If a {MBTRAIN.LINKSPEED error req} sideband message is received, the UCIe Module Partner must complete step1 through step 2 and enters electrical idle on its Receiver and sends the sideband message {MBTRAIN.LINKSPEED error resp}.
   a. Based on the number of Lanes encountering errors, the UCIe Module checks if the failing Lanes can be repaired (for Advanced package) or Width degraded (for standard package). If Lanes can be can be repaired (for Advanced package) or Width degraded (for standard package), the UCIe Module must send {MBTRAIN.LINKSPEED exit to repair req} to the UCIe Module partner and waits for a response. The UCIe Module Partner enters MBTRAIN.REPAIR and sends the sideband message {MBTRAIN.LINKSPEED exit to repair resp}. Following this the UCIe Module must exit to MBTRAIN.REPAIR.
   b. If the Lanes cannot be repaired (for Advanced package) or degraded (for Standard package), the speed must be degraded. The UCIe Module sends {MBTRAIN.LINKSPEED exit to speed degrade req} sideband message. The UCIe Module Partner must exit to MBTRAIN.SPEEDIDLE and responds with {MBTRAIN.LINKSPEED exit to speed degrade resp}. Following this, the UCIe Module must exit to MBTRAIN.SPEEDIDLE to set data rate to next lower speed.
   c. If the UCIe Module receives a {MBTRAIN.LINKSPEED exit to speed degrade req} any outstanding {MBTRAIN.LINKSPEED exit to repair req} or {MBTRAIN.LINKSPEED done req} must be ignored and the UCIe module must respond to {MBTRAIN.LINKSPEED exit to speed degrade req}.
4. For single module instantiations, if no errors are encountered, the UCIe Module must set the clock phase on its Transmitter to sample the data eye at the optimal point to maximize eye margins. The UCIe Module must send {MBTRAIN.LINKSPEED done req} sideband message. When {MBTRAIN.LINKSPEED done req} is received, the UCIe Module partner must respond with {MBTRAIN.LINKSPEED done resp}. When UCIe Module has sent and received the sideband message {MBTRAIN.LINKSPEED done resp}: If PHY_IN_RETRAIN is not set, perform Step 7. Else if the PHY_IN_RETRAIN variable is set the following actions must be taken
   a. If a change is detected in Runtime Link Testing Control register from PHYRETRAIN entry, the UCIe Module must send {MBTRAIN.LINKSPEED exit to phy retrain req} and wait for a response. Upon receiving this message, the UCIe Module Partner must exit to PHY retrain and send {MBTRAIN.LINKSPEED exit to PHY retrain resp}. Once this sideband message is received, the UCIe module must exit to PHY retrain.
   b. Else id no change is detected, Busy bit in Runtime Link Testing Status and PHY_IN_RETRAIN variable must be cleared. Step 7 below must now be performed
   c. If the UCIe Module receives a {MBTRAIN.LINKSPEED exit to speed degrade req} or {MBTRAIN.LINKSPEED exit to repair req} any outstanding

{MBTRAIN.LINKSPEED done req} must be ignored and the UCIe module must respond to the received sideband message.

5. For multi module instantiations, after step 2, each UCIe Module reports the results of step 2 to the multi-module PHY Logic (MMPL) and wait for resolution. The multi-module PHY logic upon receiving this information, must follow the rules in section 4.7.1 and provide necessary action to each UCIe Module. The actions include:
    a. MMPL directed UCIe Module exit to LINKINIT: The UCIe Module must send {MBTRAIN.LINKSPEED done req} sideband message. When {MBTRAIN.LINKSPEED done req} is received, the UCIe Module partner must respond with {MBTRAIN.LINKSPEED done resp}. When UCIe Module has sent and received the sideband message {MBTRAIN.LINKSPEED done resp} it must exit to LINKINIT. If the UCIe Module receives a {MBTRAIN.LINKSPEED exit to speed degrade req} or {MBTRAIN.LINKSPEED exit to repair req} any outstanding {MBTRAIN.LINKSPEED done req} must be ignored and the UCIe module must respond to the received sideband message.
    b. MMPL directed UCIe Module exit to SPEEDIDLE: The UCIe Module sends {MBTRAIN.LINKSPEED exit to speed degrade req} sideband message. The UCIe Module Partner must exit to MBTRAIN.SPEEDIDLE and responds with {MBTRAIN.LINKSPEED exit to speed degrade resp}. Following this, the UCIe Module must exit to MBTRAIN.SPEEDIDLE to set data rate to next lower speed.
    c. MMPL directed UCIe Module exit to REPAIR: The UCIe Module must send {MBTRAIN.LINKSPEED exit to repair req} to the UCIe Module partner and waits for a response. The UCIe Module Partner enters MBTRAIN.REPAIR and sends the sideband message {MBTRAIN.LINKSPEED exit to repair resp}. Following this the UCIe Module must exit to MBTRAIN.REPAIR
6. If no errors are encountered and it is a Retimer, the following rules apply to achieve Link match (if required)
    a. Retimer must not send {MBTRAIN.LINKSPEED done req} unless the target Link speed and width of the remote Retimer partner resolves to current Link and width.
    b. While resolving the target Link speed and width with the remote Retimer partner, it must send {MBTRAIN.LINKSPEED done resp} with stall encoding every 4 ms. UCIe Retimer must ensure that this stall is not perpetual, and an implementation specific timeout must be included in the Retimer.
    c. If the local UCIe Link speed or width is greater than the remote Retimer UCIe Link, then it must perform the following actions in sequence:
        i. It must send the {MBTRAIN.LINKSPEED done resp} without the stall encoding.
        ii. This conditions must be treated as an error condition, and perform Step 4 or Step 5 above to with repair or speed degrade which ever is applicable.
7. Both Transmitters and Receivers are now enabled and idle and both devices exit to LINKINIT

4.5.3.4.13 MBTRAIN.REPAIR

This state can be entered from PHYRETRAIN or from MBINIT.LINKSPEED. For Advanced package interface, this state will be used to apply repair and for Standard package interface, this state will be used for Link width degrade.

For Advanced Package interface, if the number of repair resources currently available is greater than the number of Lanes encountering errors, repair must be applied

1. The UCIe Module sends the sideband message {MBTRAIN.REPAIR init req} for its Transmitter and the UCIe Module Partner responds with {MBTRAIN.REPAIR init resp}.

2. If Lane repair is possible, the UCIe Module applies repair on its Transmitter Lanes as described in section 4.3.1 and sends {MBTRAIN.REPAIR Apply repair req} sideband message. The UCIe Module Partner applies repair as described in section 4.3.1 and responds with {MBTRAIN.REPAIR Apply repair resp} sideband message once the required repair is applied.
3. The UCIe Module must send {MBTRAIN.REPAIR end req} sideband message and waits for a response. The UCIe Module Partner must then respond with {MBTRAIN.REPAIR end resp}. When a UCIe Module has sent and received {MBTRAIN.REPAIR end resp}, it must exit to MBTRAIN.TXSELFCAL

For a Standard package interface, if the number of Lanes encountering errors are all contained within Lane 0 through Lane 7 or Lane8 through Lane 15 width must be degraded to a x8 Link (Lane 0 ... Lane 7 or Lane 8 ... Lane 15)

1. The UCIe Module sends the sideband message {MBTRAIN.REPAIR init req} and the Receiver responds with {MBTRAIN.REPAIR init resp}.
2. If width degrade is possible, the UCIe Module requesting width degrade applies width degrade and sends {MBTRAIN.REPAIR Apply degrade req} sideband message. The message includes Lane reversal aware Lane map information (2b’01 for Lane 0...7 and 2b’10 for Lanes 8...15). The UCIe Module Partner will use Lanes 0...7 if Lane map 2b’01 is received and Lanes 8...15 if Lane map 2b’10 is received. Once degrade is applied the UCIe Module responds with {MBTRAIN.REPAIR Apply degrade resp}.
3. The UCIe Module must send {MBTRAIN.REPAIR end req} sideband message and wait for a response. The UCIe Module Partner must then respond with {MBTRAIN.REPAIR end resp}. When UCIe Module has sent and received {MBTRAIN.REPAIR end resp}, it must exit to MBTRAIN.TXSELFCAL.

4.5.3.5 LINKINIT

This state is used to allow complete die to die adapter to complete initial Link management before entering Active state on RDI. Refer to section 8.1.6 for more details on RDI bring up flow.

Once RDI is in Active state, the PHY will clear its copy of “Start UCIe Link training” bit from UCIe Link control register.

The LFSR must be RESET upon entering this state.

This state is common for **Advanced** and **Standard Package interfaces**.

4.5.3.6 ACTIVE

Physical layer initialization is complete, RDI is in Active stateand packets from upper layers can be exchanged between the two dies.

All data in this state is scrambled using the scrambler LFSR described in section 4.4.1.

This state is common for **Advanced** and **Standard Package interfaces**.

4.5.3.7 PHYRETRAIN

A die can enter PHY retrain for a number of reasons. The trigger for PHY to enter PHY retrain is one of the scenarios described below:

- Adapter directed PHY retrain: Adapter can direct the PHY to retrain for any reason it deems necessary.

- PHY initiated PHY retrain: Local PHY must initiate retrain on detecting a Valid framing error.
- Remote die requested PHY retrain: Local PHY must enter PHY retrain on receiving a request from the remote die.
- If a change is detected in Runtime Link Testing Control register during MBTRAIN.LINKSPEED.

A variable PHY_IN_RETRAIN must be set when entering PHYRETRAIN

Table 26. Link Test Status Register based Retrain encoding

Link Test Status Register		Retain Encoding Resolution
Start bit value	**Repair Required**	
0b	NA	TXSELFCAL
1b	Repair needed	REPAIR (If repair resources are available)
		SPEEDIDLE (if unrepairable)
1b	No Repair	TXSELFCAL

Table 27. Retrain encoding

Retrain Encoding	State	Retain Condition
001b	TXSELFCAL	No Lane errors (Valid framing errors detected by PHY)
010b	SPEEDIDLE	Lane errors & faulty Lanes cannot be repaired
100b	REPAIR	Lane errors & faulty Lanes are repairable

Table 28. Retrain exit state resolution

Retrain reg Condition		Retrain Request Encoding		Resolved State Encoding		Exit
Die 0	**Die 1**	**Die 0**	**Die 1**	**Die 0**	**Die 1**	**Both Dies**
No Lane Errors	No Lane Errors	001b	001b	001b	001b	MBTRAIN.TXSELFCAL
No Lane Errors	Repair	001b	100b	100b	100b	MBTRAIN.REPAIR
No Lane Errors	Speed Degrade	001b	010b	010b	010b	MBTRAIN.SPEEDIDLE
Repair	No Lane Errors	100b	001b	100b	100b	MBTRAIN.REPAIR
Repair	Repair	100b	100b	100b	100b	MBTRAIN.REPAIR
Repair	Speed Degrade	100b	010b	010b	010b	MBTRAIN.SPEEDIDLE
Speed Degrade	No Lane Errors	010b	001b	010b	010b	MBTRAIN.SPEEDIDLE
Speed Degrade	Repair	010b	100b	010b	010b	MBTRAIN.SPEEDIDLE
Speed Degrade	Speed Degrade	010b	010b	010b	010b	MBTRAIN.SPEEDIDLE

4.5.3.7.1 Adapter initiated PHY retrain

Following is the sequence of steps for an Adapter initiated PHY retrain:

1. UCIe Module receives retrain request from the local Adapter (RDI state req moved to Retrain). Following this, the UCIe Module must complete the stall Req/Ack (pl_stallreq; lp_stallack) hand shake on RDI as described in Chapter 8.0.
2. The UCIe Module must send sideband message {PHYRETRAIN.retrain init req} to UCIe Module Partner
3. The UCIe Module Partner on receiving the sideband message {PHYRETRAIN.retrain init req} must transition its RDI state to Retrain after completion of stall Req/Ack (pl_stallreq; lp_stallack) hand shake on its RDI interface. Following this the UCIe Module Partner responds with {PHYRETRAIN.retrain init resp}.
4. Once {PHYRETRAIN.retrain init resp} is received, the UCIe Module must transition its RDI to Retrain
5. UCIe Module must send {PHYRETRAIN. retrain start req} with retrain encoding reflecting the contents of Runtime Link Test Control register except the Start bit (if the Busy bit in Runtime Link Test Status Register is set). Following this UCIe Module Partner compares the received retrain encoding with the local retrain encoding. If received retrain encoding is the same as the local retrain encoding, the UCIe Module Partner must respond with {PHYRETRAIN.retrain start resp}. If the retrain encodings do not match, the UCIe module partner must resolve according to Retrain encodings and resolutions shown Table 26, Table 27 and Table 28 and then send {PHYRETRAIN.retrain start resp}.

6. Once UCIe Module sends and receives the sideband message {PHYRETRAIN.retrain start resp}, it must exit to corresponding training state according to the resolved retrain register encoding

4.5.3.7.2 PHY initiated PHY retrain

Following is the sequence of steps for PHY initiated PHY retrian:

1. On detecting a valid framing error, the UCIe module must assert pl_error when transmitting that flit (or flit chunk) on RDI. Following this the UCIe module (PHY) must complete the stall Req/Ack (pl_stallreq; lp_stallack) hand shake on RDI.
2. The UCIe Module must sideband message {PHYRETRAIN.retrain init req}
3. The UCIe Module Partner on receiving the sideband message {PHYRETRAIN.retrain init req} must transition its RDI to retrain after completion of stall Req/Ack (pl_stallreq; lp_stallack) hand shake on its RDI. Following that the UCIe Module Partner responds with {PHYRETRAIN.retrain init resp}
4. Once {PHYRETRAIN.retrain init resp} is received, the UCIe Module must transition its RDI to retrain
5. UCIe Module must send {PHYRETRAIN. retrain start req} with retrain encoding reflecting the contents of Runtime Link Test Control register except the Start bit (if the Busy bit in Runtime Link Test Status Register is set). Following this the UCIe Module Partner compares the received retrain encoding with the local retrain encoding. If received retrain encoding is the same as the local retrain encoding, the UCIe Module Partner must respond with {PHYRETRAIN.retrain start resp}. If the retrain encodings do not match, the UCIe module partner must resolve according to Retrain encodings and resolutions shown Table 26, Table 27 and Table 28 and then send {PHYRETRAIN.retrain start resp}.
6. Once a UCIe Module has sent and received the sideband message {PHYRETRAIN.retrain start resp}, it must exit to corresponding training state according to the retrain encoding.

4.5.3.7.3 Remote Die requested PHY retrain

1. On receiving {PHYRETRAIN.retrain init req}, the UCIe Module must transition local RDI to retrain after completion of stall Req/Ack (pl_stallreq; lp_stallack) hand shake on its RDI. Following that the UCIe Module responds with {PHYRETRAIN.retrain init resp}
2. Once {PHYRETRAIN.retrain init resp} is received, the UCIe Module Partner must transition its RDI to retrain
3. UCIe Module must send {PHYRETRAIN. retrain start req} with retrain encoding reflecting the contents of Runtime Link Test Control register except the Start bit (if the Busy bit in Runtime Link Test Status Register is set). Following this compares the received retrain encoding with the local retrain encoding. If received retrain encoding is the same as the local retrain encoding, the UCIe Module Partner responds with {PHYRETRAIN.retrain start resp}. If the retrain encodings do not match, the UCIe module partner must resolve according to Retrain encodings and resolutions shown Table 26, Table 27 and Table 28 and then send {PHYRETRAIN.retrain start resp}.
4. Once a die has sent and received the sideband message {PHYRETRAIN.retrain start resp}, it must exit to corresponding training state according to the retrain encoding.

4.5.3.7.4 PHY retrain from LINKSPEED

1. The UCIe Module must send {PHYRETRAIN. retrain start req} with retrain encoding reflecting the contents of Runtime Link Test Control register except the Start bit (if the Busy bit in Runtime Link Test Status Register is set). Following this UCIe Module Partner compares the received retrain encoding with the local retrain encoding. If received retrain encoding is the same as the local retrain encoding, the UCIe Module Partner must respond with {PHYRETRAIN.retrain start resp}. If the retrain encodings do not match, the UCIe module partner must resolve according to Retrain encodings and resolutions shown Table 26, Table 27 and Table 28 and then send {PHYRETRAIN.retrain start resp}.
2. Once a die has sent and received the sideband message {PHYRETRAIN.retrain start resp}, it must exit to corresponding training state according to the retrain encoding.

4.5.3.8 TRAINERROR

This state used as a transitional state due to any fatal or non-fatal events that need to bring the state machine back to RESET state. This can happen during initialization and training or if "Start UCIe Link training" bit from UCIe Link control register is set when state machine is not in RESET. It is also used for any events that transition the Link from a Link Up to a Link Down condition.

The exit from TRAINERROR to RESET is implementation specific. For cases when there is no error escalation (i.e., RDI is not in LinkError), it is recommended to exit TRAINERROR as soon as possible. For cases when there is error escalation (i.e., RDI is in LinkError), it is required for Physical Layer to be in TRAINERROR as long as RDI is in LinkError.

For correctable, non-fatal and fatal error escalation on RDI refer to Chapter 8.0.

This state is common for **Advanced** and **Standard Package interfaces**.

If sideband is Active, a sideband handshake must be performed for both UCIe Module and UCIe module Partner to enter TRAINERROR state from any state other than SBINIT. The following is defined as the TRAINERROR handshake

- The UCIe Module requesting exit to TRAINERROR must send {TRAINERROR Entry req} sideband message and wait for a response. The UCIe Module Partner must exit to TRAINERROR and respond with {TRAINERROR Entry resp}. Once {TRAINERROR Entry resp} sideband message is received, the UCIe Module must exit to TRAINERROR. If no response is received for 8ms, the Link Training State Machine transitions to TRAINERROR.

4.5.3.9 L1/L2

PM state allows a lower power state than dynamic clock gating in ACTIVE.

- This state is entered when RDI has transitioned to PM state as described in Chapter 8.0. The PHY power saving features in this state are implementation specific.
- When local Adapter requests Active on RDI or remote Link partner requests L1 exit the PHY must exit to MBTRAIN.SPEEDIDLE. L1 exit is coordinated with the corresponding L1 state exit transitions on RDI.
- When local Adapter requests Active on RDI or remote Link partner requests L2 exit the PHY must exit to RESET. L2 exit is coordinated with the corresponding L2 state exit transitions on RDI.

4.6 Run time Recalibration

Track signal can be used by the Receiver to perform periodic runtime calibration while in ACTIVE. Main band data must continued to be sampled properly during runtime recalibration. To request track pattern, following is the sequence.

- The UCIe Module enables the Track signal buffers on its Receiver and sends a sideband message {RECAL.track pattern init req} and waits for a response.
- The UCIe Module Partner sends {RECAL. track pattern init resp} and enables its Track signal Transmitter. Following this, Transmitter starts sending the pattern described in section 5.5.1.
- The Receiver performs the required recalibration and sends the {RECAL. track pattern done req} sideband message.
- Upon receiving this message, the Transmitter stops sending the pattern and sends the sideband message {RECAL.track pattern done resp}.
- The Receiver is permitted to disable the track Receiver upon receiving this sideband message {RECAL.track pattern done resp}

4.7 Multi-module Link

As described in Chapter 1.0, the permitted configurations for a multi-module Link are one, two and four module configurations.

4.7.1 Multi-module initialization

Each module in a multi-module configuration must initialize and train independently, using its sideband. If two or four modules are used, a separate multi-module phy logic block coordinates across the modules, as described in section 1.2.2. The multi-module phy logic (MMPL) is responsible for orchestrating data transfer across the multiple modules. Each module in a multi-module Link must operate at the same width and speed. During initialization or retraining, if any module failed to train, the MMPL must ensure multi-module configuration degrades to next degraded permitted configuration. Subsequently, any differences in speed and width between the different modules must be resolved using the following rules:

1. For Standard package multi-module configuration if width degrade is reported for any of the modules:
    a. If less than or equal to half the number of modules report width degrade at the current Link speed, the corresponding Modules must be disabled. The MMPL must ensure multi-module configuration degrades to next degraded permitted configuration. For example, if three out of four modules are active, MMPL must degrade the Link to a two module configuration.
    b. If majority of modules report report width degrade at the current Link speed refer to the pseudo code below

```
CLS: Current Link Speed
CLS-1: Next lower allowed Link Speed
M is number of active Modules
Aggregate Raw BW(M, CLS) = Common Minimum Link width * M * CLS
If modules report Width degrade:
       If CLS = 4 GT/s
              Apply Width degrade for all modules
       Else If Aggregate Raw BW(M, (CLS-1)) > Aggregate raw BW (M/2, CLS):
              Attempt Speed Degrade
       Else:
              Apply Width degrade for all modules
```

2. For Advanced or Standard package multi-module configuration if Modules report speed difference, refer to the pseudo code below:

```
IF modules report speed difference:
       CMLS: Common Maximum Link Speed
       HMLS: Highest Maximum Link Speed of next lower configuration
       IF HMLS/2 > CMLS:
              Modules degrade to next lower configuration
       Else:
              Speed for all modules degrades to CMLS
```

3. For Standard package modules if both width and speed differences are reported, the MMPL must pick the best configuration between (1) and (2) above

4.7.1.1 Sideband assignment

During Link initialization, training and retraining (section 4.5) all sideband messages are sent on individual module sideband interfaces.

Note: For all other sideband messages from upper layers or related to RDI state transitions, a single sideband is used to send and receive sideband messages. A device must send sideband messages on its Logical LSB module sideband interface (Module-0).A message sent on a Logical LSB module can be received on a different logical module on sideband Receiver.

5.0 Electrical Layer

Key attributes of electrical specification include:

- Support for 4, 8, 12, 16, 24 and 32 GT/s data rates
- Support for Advanced and Standard package interconnects.
- Support for clock and power gating mechanisms
- Single-ended unidirectional data signaling
- DC coupled point-to-point interconnect.
- Forwarded clock for transmit jitter tracking
- Matched length interconnect design within a module
- Tx driver strength control and unterminated Rx for Advanced Package. Tx termination and data rate and channel reach dependent Rx termination for Standard Package.

5.1 Interoperability

5.1.1 Data rates

A device must support 4 GT/s and all the data rates data rates between 4 GT/s and the highest supported data rate. For example, a device supporting 16 GT/s must also support 4, 8, 12 GT/s Data rates. SSC clock is allowed, but no ppm difference is allowed between Transmitter and Receiver.

5.2 Overview

5.2.1 Interface Overview

High level block diagrams of UCIe PHY are shown in Figure 72 and Figure 73. The UCIe physical interface consists of building blocks called Modules. A module using advanced packaging technology (e.g. EMIB, CoWoS) called "Advanced Package Module" consists of a pair of clocks, 64 single-ended data Lanes, a data valid Lane each direction (transmit and receive) and a Track Lane. There is a low speed sideband bus for initialization, Link training and configuration reads/writes. The sideband consists of a single-ended sideband data Lane and single-ended sideband clock Lane in both directions (transmit and receive).

The "Standard Package Module" uses a traditional Standard packaging with larger pitch. An Standard Package Module consists of a pair of clocks, 16 single-ended data Lanes, a data valid Lane and Track Lane in each direction (transmit and receive). There is a low speed sideband bus for initialization, Link training and configuration reads/ writes. The sideband consists of a single-ended sideband data Lane and single-ended sideband clock Lane in both directions (transmit and receive)

For some applications, multiple modules (2 or 4) can be aggregated to delivers additional bandwidth.

To avoid reliability issues, we recommend limiting the Transmitter output high to a maximum of 100mV above the receiving chiplet's Receiver front end circuit power supply rail. Over stress protection circuit can be implemented in Receiver when Transmitter output high is more than 100mV above Receiver power supply rail.

Figure 72. Advanced Package module

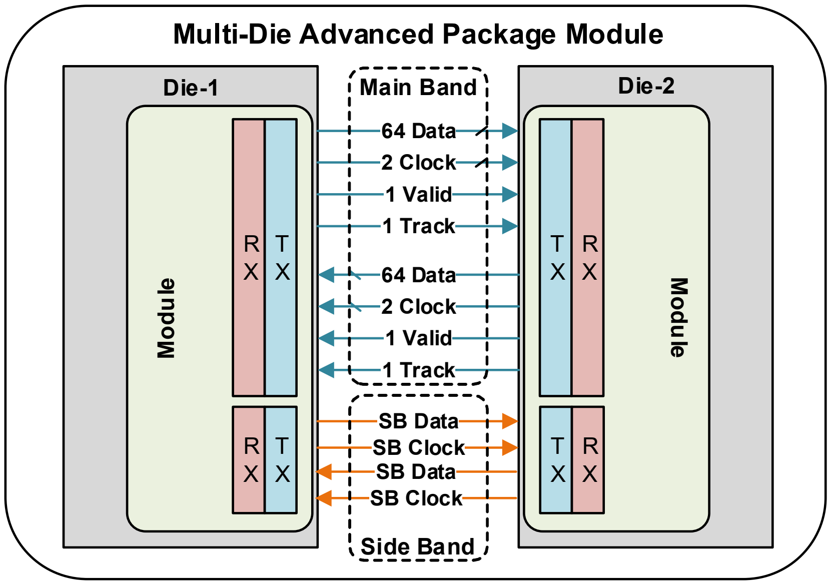
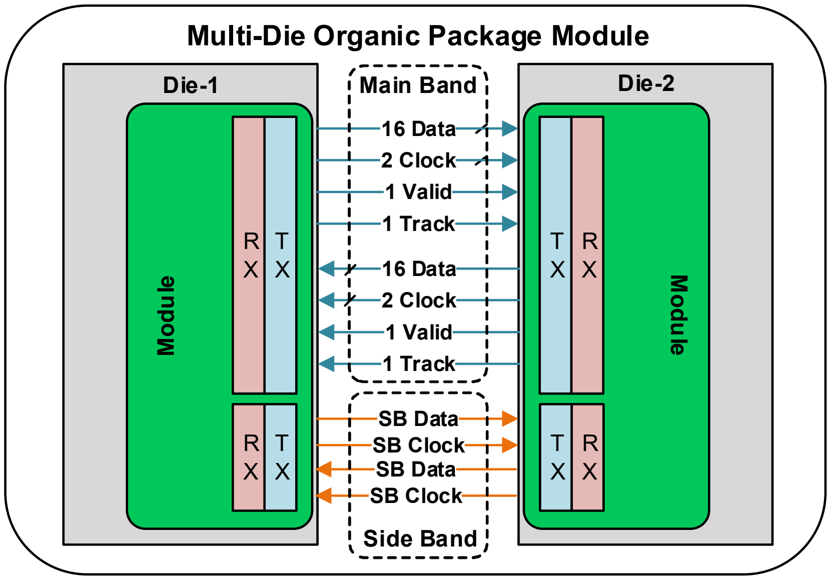


Figure 73. Standard Package module



5.2.2 Electrical summary

Table 29 defines the PHY electrical characteristics a UCIe device must adhere to.

Table 29. Electrical summary

	Standard Package				**Advanced Package**		
Data Width (per module)	16	16	16	16	64	64	64
Data Rate (GT/s)	4-16	4/8/12	16	24/32	4/8/12	16	24/32

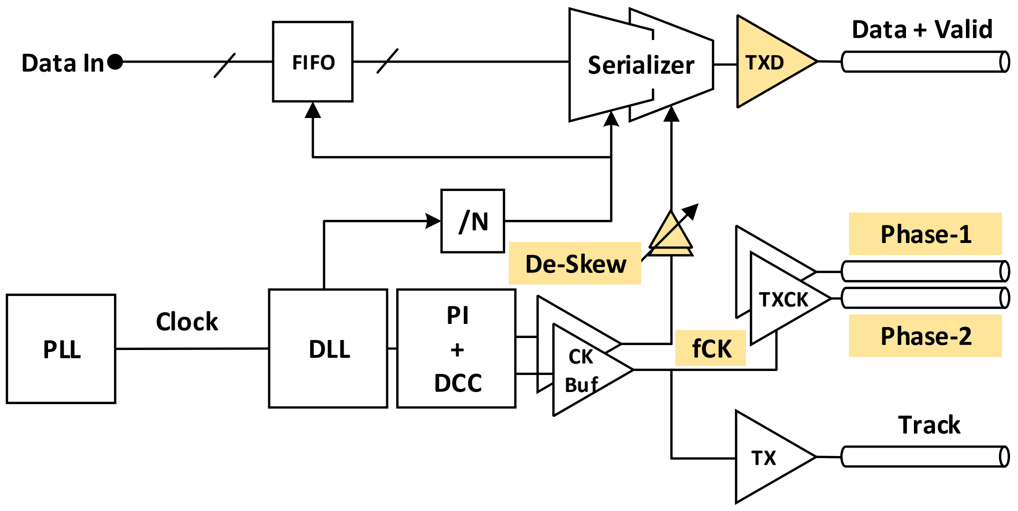
	Standard Package				Advanced Package		
Power Efficiency Target (pJ/b)	Refer to Table 6 in Chapter 1.0						
Latency Target (TX+RX) (UI)[1] (Target upper bound)	12	12	12	16	12	12	16
Idle Exit/Entry Latency (target upper bound)	0.5 ns	0.5 ns	1 ns	1 ns	0.5 ns	1 ns	1 ns
Idle Power (% of peak power) (target upper bound)	15%	15%	15%	15%	15%	15%	15%
Channel Reach (mm)	2-10	25	25	25	2	2	2
Die Edge Bandwidth Density (GB/s/mm)[2]	Refer to Table 6 in Chapter 1.0						
Bandwidth area density (GB/mm^2)	21-85	21/42/64	85	109/145	158/316/473	631	710/947
PHY dimension width (um)[3]	571.5	571.5	571.5	571.5	388.8	388.8	388.8
PHY dimension Depth (um)[4]	1320	1320	1320	1540	1043	1043	
ESD[5]	30V CDM (Anticipating going to 5-10V in Future.)						

1. Electrical PHY latency target. For overall latency target refer to Table 6 in Chapter 1.0
2. Die edge bandwidth density defined as total I/O bandwidth in GB per mm silicon die edge.
3. For compatibility, PHY dimension width must match spec for Advanced Package. Tolerance of PHY width for Standard Package can be higher since there are more routing flexibility. For best channel performance, it's recommended for width to be close to spec
4. PHY dimension depth is an informative parameter and depends on bump pitch. Number in the table is based on 45 um bump pitch for Advanced package and 110 um for Standard package
5. Reference (Industry Council on ESD Target Levels): White Paper 2: A Case for Lowering Component-level CDM ESD Specifications and Requirements

5.3 Transmitter Specification

The Transmitter topology is shown in Figure 74. Each data module consists of N single-ended data Transmitters plus a Valid signal. N is 68 (64 Data + 4 Redundant Data) for Advanced Package and 16 for Standard Package. There is a pair Transmitters for clocking and a Track signal in each module. The clock rates and phases are discussed in detail in clocking section section 5.5.

Figure 74. Transmitter



The Valid signal is used to gate the clock distribution to all data Lanes to enable fast idle exit and entry. The signal also serves the purpose of Valid framing, see section section 4.1.2 for details. TX for Valid is expected to be the same as regular data TX.

The Track signal can be used for PHY to compensate for slow changing variables such as voltage or temperature. It's a unidirectional signal similar to a data bit. The Transmitter sends a copy of Phase-1 of the clock signal when requested over the sideband by the Receiver.

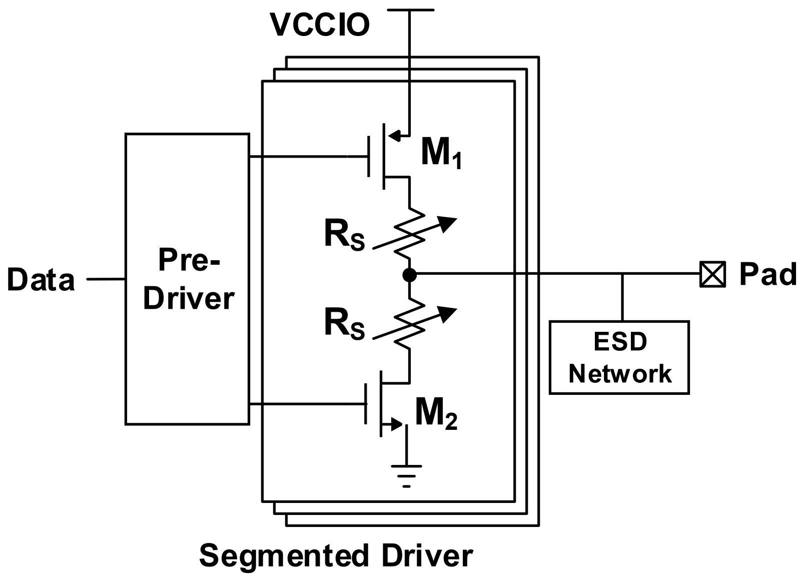
5.3.1 Driver Topology

The Transmitter is optimized for simplicity and low power operation. An example of a low power Transmitter is shown in Figure 75. Separate pull-up and pull-down network strengths are permitted to achieve optimal performance across different channel configurations.

A control loop or training is recommended to adjust output impedance to compensate for the process, voltage and temperature variations. These are implementation specific and beyond the scope of this specification. In low power states, the implementation must be capable of tri-stating the output.

It is recommended to optimize the ESD network to minimize pad capacitance. Inductive peaking technique such as T-coil may be needed at higher data rates.

Figure 75. Transmitter driver example circuit



5.3.2 Transmitter Electrical parameters

Table 30 defines the Transmitter electrical parameters.

Table 30. Transmitter Electrical Parameters

Parameter	Min	Typ	Max	Unit
Data Lane TX Swing	0.4			
Fwd Clock Tx Swing (single ended)	0.4			
Incoming Clock Slew Rate[1]	0.1	0.22	0.25	UI
Incoming Differential Clock Overlap[1]	-	-	30	mUI
Incoming Data Slew Rate[1]	-	0.35	-	UI
Driver Pull-up/down Impedance[2]	22	25	28	Ohms
Driver Pull-up/down Impedance[3]	27	30	33	Ohms
Impedance Step Size[3]	-	-	0.5	Ohms
Pull-up/down Delay Mismatch	-	-	20	mUI
1-UI Total Jitter[4]	-	-	96/113	mUI pk-pk
1-UI Deterministic Jitter (Dual Dirac)[5]	-	-	48	mUI pk-pk
Duty Cycle Error[6]	-0.02	-	0.02	UI
Lane-to-Lane Skew Correction Range (up to 16 GT/s)[2]	-0.1	-	0.1	UI
Lane-to-Lane Skew Correction Range (up to 32 GT/s)[2]	-0.15	-	0.15	UI
Lane-to-Lane Skew Correction Range (up to 16 GT/s)[3]	-0.14	-	0.14	UI
Lane-to-Lane Skew Correction Range (up to 32 GT/s)[3]	-0.22	-	0.22	UI

Table 30. Transmitter Electrical Parameters

Parameter	Min	Typ	Max	Unit
Lane-to-Lane Skew[6]	-0.02	-	0.02	UI
Clock to Mean Data Training Accuracy[7]	-0.07	-	0.07	UI
Phase adjustment step	-	-	16	mUI
TX Pad Capacitance (for all speeds)[8]	-	-	250	fF
TX Pad Capacitance (8 GT/s capable design)[3]	-	-	300	fF
TX Pad Capacitance (16 GT/s capable design)[9]	-	-	200	fF
TX Pad Capacitance (32 GT/s capable design)[9]	-	-	125	fF

1. Expected input (informative)
2. Advanced package
3. Standard package
4. At BER 1E-15/1E-27
5. Data dependent jitter excluding DCE
6. Post correction
7. Includes static and tracking error
8. Effective pad capacitance Advanced package mode
9. Effective pad capacitance Standard package mode

5.3.3 24.0, and 32.0 GT/s Transmitter Equalization

Transmitter equalization is recommended for 16 GT/s and must be supported at 24.0 GT/s and 32 GT/s data rates to mitigate the channel ISI impact. Tx equalization is de-emphasis only for all applicable Data rates.

Tx equalization coefficients for 24.0 and 32.0 GT/s are based on the FIR filter shown in Figure 76. Equalization coefficient is subject maximum unity swing constraint.

The Transmitter must support the equalization settings shown in Table 31. Determination of de-emphasis setting is based on initial configuration or training sequence, where the value with larger eye opening will be selected.

Figure 76. Transmitter de-emphasis

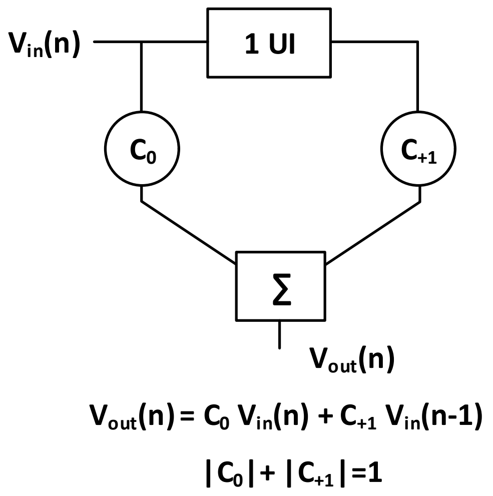


Figure 77. Transmitter de-emphasis waveform

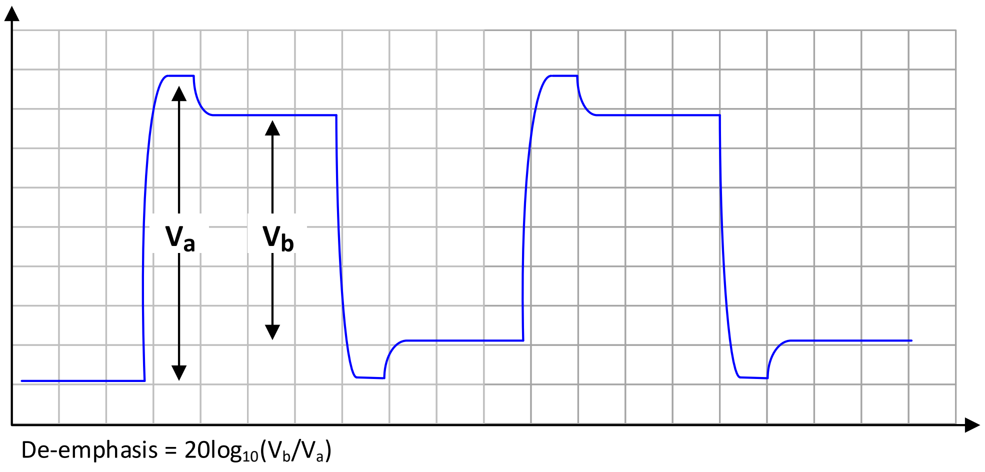


Table 31. Transmitter de-emphasis values

Setting	De-emphasis	Accuracy	C_{+1}	V_b/V_a
1	0.0 dB	-		
2	-2.2 dB	+/- 0.5 dB		

5.4 Receiver Specification

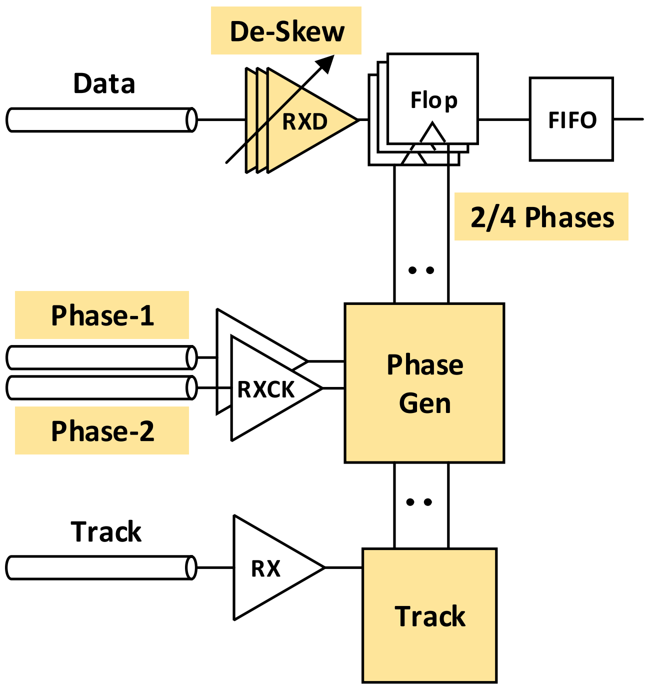
The Receiver topology is illustrated in Figure 78. Each module (Advanced Package and Standard Package) consists of clocks buffer, data Receivers and track Receiver.

The received clock is used to sample the incoming data. The Receiver must match the delays between the clock path and the data/valid path to the sampler. This is to minimize the impact of power supply noise induced jitter. The data Receivers may be implemented as 2-way or 4-way interleaved. For 4-way interleaved implementation the Receiver needs to generate required phases internally from the two phase of the forwarded clock. This may require duty cycle correction capability on the Receiver.The supported forwarded clock frequencies and phases are described in section 5.5.

At higher data rates, de-skew capability may be needed to achieve the matching requirements between the data Lanes. De-skew (when applicable) can be performed during main band training. More details are provided in the logic chapter.

Track Receiver receives a replica clock from the Transmitter and may use it to track the impact of slow varying voltage and temperature changes on sampling phase.

Figure 78. Receiver topology



5.4.1 Receiver Electrical Parameters

The specified Receiver electrical parameters are shown in Table 32

Table 32. Receiver Electrical parameters

Parameter	Min	Typ	Max	Unit
RX Input Impedance[1]	45	50	55	Ohms
Impedance Step Size[1]	-	-	1	Ohms
Data/Clock Total Differential Jitter[2]	-	-	0.03	UI
Lane-to-Lane skew (up to 16 GT/s)[3]	-0.07	-	0.07	UI
Lane-to Lane skew (> 16 GT/s)[3]	-0.12	-	0.12	UI
Phase error[4] (Including Duty cycle error and IQ mismatch)	-0.04	-	0.04	UI

Table 32. Receiver Electrical parameters

Parameter	Min	Typ	Max	Unit
Per Lane de-skew adjustment step[5]	-	-	16	mUI
Output Rise Time[6]	-	-	0.1	UI
Output Fall Time[6]	-	-	0.1	UI
RX Pad Capacitance[7]	-	-	200	fF
RX Pad Capacitance (up to 8 GT/s)[1]	-	-	300	fF
RX Pad Capacitance (up to 16 GT/s)[8,1]	-	-	200	fF
RX Pad Capacitance (24 and 32 GT/s)[8,1]	-	-	125	fF
Rx Voltage sensitivity	-	-	40	mV

1. Standard Package mode with termination
2. Based on matched architecture
3. Require Rx per Lane de-skew if limit is exceeded
4. Residual error post training and correction
5. When applicable
6. Expected output (informative)
7. Advanced Package mode
8. Effective Pad capacitance

5.4.2 Rx Termination

Rx termination is applicable only to Standard Package modules. All Receivers on Advanced Package modules must be unterminated.

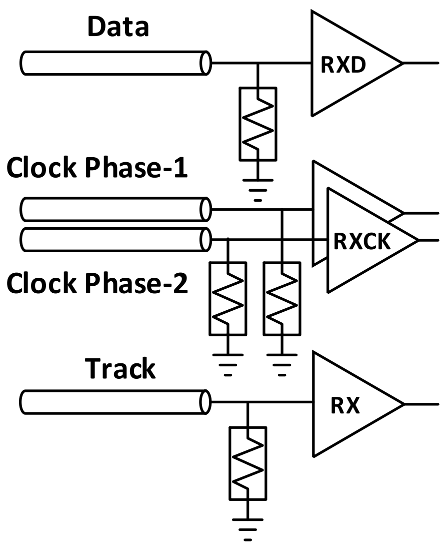
Receiver termination on Standard Package is data rate and channel dependent. Table 33 shows the maximum data rate and channel reach combinations for which the Standard Package module must remain unterminated for a minimally compliant Transmitter.

Table 33. Maximum channel reach for unterminated Receiver (minimally compliant Transmitter)

Data Rate (GT/s)	Channel Reach (mm)
12	3
8	5
4	10

Termination is required for all other combinations. Receivers must be ground-terminated when applicable as shown in Figure 79

Figure 79. Receiver termination

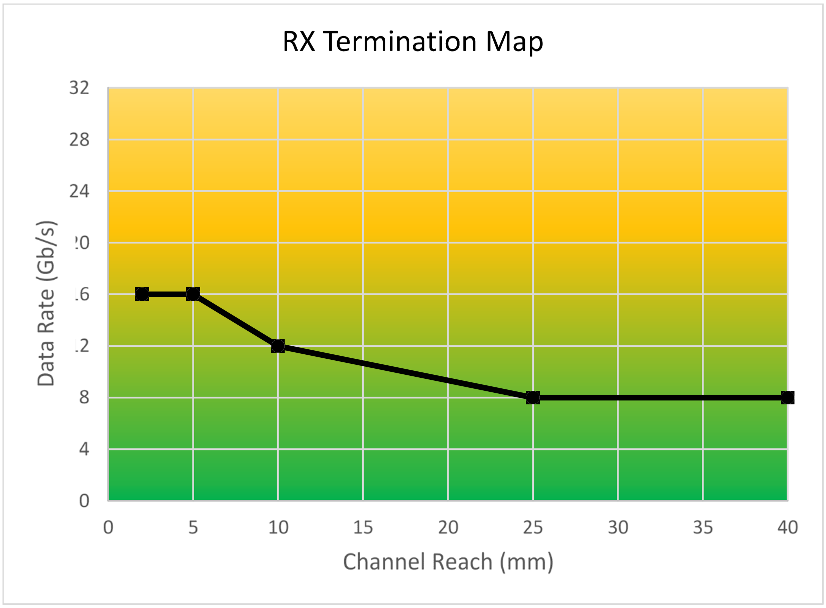


For higher Transmitter swing, unterminated Receiver can be extended to longer channel and high data rate. Table 34 shows the maximum data rate and channel reach combinations for Transmitter swing and 0.85 V (maximum recommended swing). Figure 80 shows an alternate representation of termination requirement. The area below the curve in Figure 80 shows the speed and channel reach combinations for which the Receivers in Standard package modules must remain unterminated.

Table 34. Maximum Channel reach for unterminated Receiver (TX swing = 0.85V).

Data Rate (GT/s)	Channel Reach (mm)
16	5
12	10
8 and below	All supported Lengths

Figure 80. Receiver termination map for Table 34



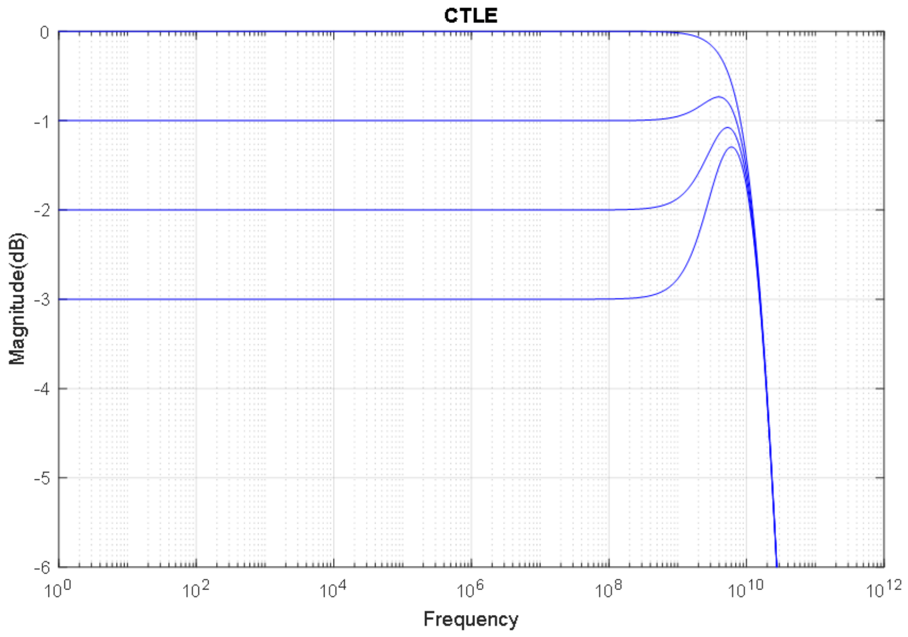
5.4.3 24.0, and 32.0 GT/s Receiver Equalization

Optional Receiver equalization may be implemented at 24.0 GT/s and 32 GT/s data rates. This enables Link operation even when TX equalization is not available. Implementation can be CTLE, inductive peaking, 1-tap DFE, or others. Expected RX equalization capability is equivalent of 1st order CTLE. Example transfer function curves of a first order CTLE are shown in Figure 81 and the corresponding equation is shown below:

$$H(s) = \omega_{p2}\left(\frac{s + A_{DC}\omega_{p1}}{(s+\omega_{p1})(s+\omega_{p2})}\right)$$

where, ω_{p2} = 2π*DataRate, ω_{p1} = 2π*DataRate /4, and A_{DC} is the DC gain.

Figure 81. Example CTLE



5.5 Clocking

Figure 82 shows the forwarded clocking architecture. Each module supports a two phase forwarded clock. It is critical to maintain matching between all data Lanes and valid signal within the module. The Receiver must provide matched delays between the receive clock distribution and data/valid Receiver path. This is to minimize the impact of power supply noise induced jitter on the Link performance. Phase adjustment is performed on the Transmitter as shown in Figure 82. Link training is required to set the position of phase adjustment to maximize the Link margin.

At higher data rates, Receiver eye margins may be small and any skew between the data Lanes (including valid) may further degrade the Link performance. Per Lane de-skew must be supported on the Transmitter at high data rates.

This specification supports quarter rate clock frequencies at very high data rates. Since quadrature clocks are required in such a case, the forwarded clock must support quadrature phases.

Table 35 shows the clock frequencies and phases that must be supported at different data rates.

5.5.1 Track

Track signal can be used to perform runtime recalibration to adjust the Receiver clock path against slow varying voltage, temperature and transistor aging conditions.

When requested by the Receiver, the Transmitter sends a copy of Phase-1 of the clock shown in Figure 82.

Figure 82. Clocking architecture

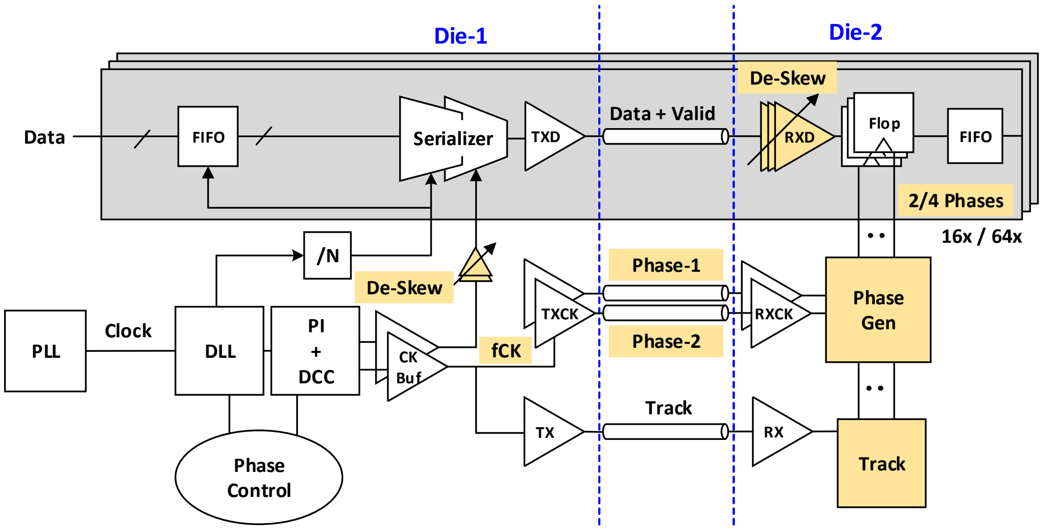


Table 35. Forwarded clock frequency and phase

Data rate (GT/s)	Clock freq. (fCK) (GHz)	Phase -1	Phase-2	De-skew (Req/Opt)
32	16	90	270	Required
	8	45	135	Required
24	12	90	270	Required
	6	45	135	Required
16	8	90	270	Required
12	6	90	270	Required
8	4	90	270	Optional
4	2	90	270	Optional

5.6 Supply noise and clock skew

I/O Vcc noise and the clock skew between data modules shall be within the range specified in table Table 36.

Table 36. IO Noise and Clock Skew

Parameter	Min	Nom	Max	Unit	
I/O Vcc noise	-	-	80	mVpp	at 8Gb/s
I/O Vcc noise	-	-	40-50	mVpp	at 16Gb/s
I/O Vcc noise	-	-	30	mVpp	at 32Gb/s
Module to module clock skew[1]	-	-	60	ps	
Note: IO VCC noise includes bandwidth above 20 MHz					

1. Applies only to multi-module instantiations

IMPLEMENTATION NOTE:

Due to different micro bump max current capacity and power delivery requirements, PHY in Advanced Package may have TX providing IO power supply to RX circuits.

Due to very low current draw, sideband supply voltage is strongly recommended to be on an always-on power domain.

5.7 Ball-out and Channel Specification

UCIe interconnect channel needs to meet the requirement of minimum rectangular eye open as specified in Table 37 under channel compliance simulation conditions with noiseless and jitter-less behavioral TX and RX models.

Figure 83. Example Eye diagram

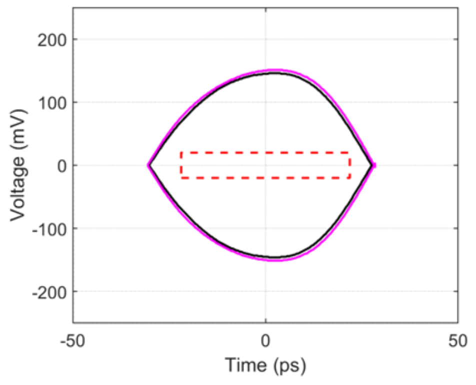


Table 37. Eye requirements

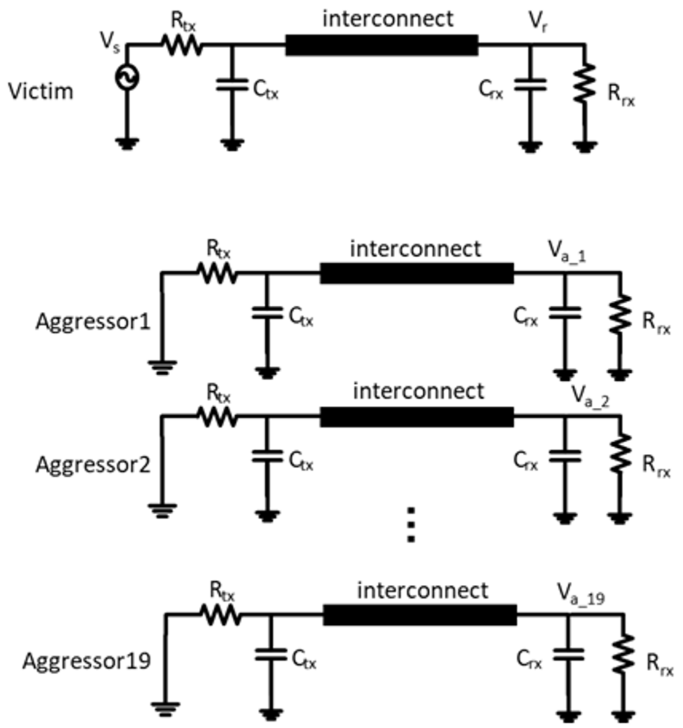
Data Rate (GT/s)	Eye Height (mV)	Eye width (UI)
4, 8, 12, 16[1]	40	0.75
24, 32[2],[3]	40	0.65

1. Rectangular mask
2. With equalization enabled
3. Based on minimum Tx swing specification

5.7.1 Voltage Transfer Function

Voltage Transfer Function (VTF) based metrics are used to define insertion loss and crosstalk. VTF metrics incorporate both resistive and capacitive components of TX and RX terminations. Figure 84 shows the circuit diagram for VTF calculations.

Figure 84. Circuit for VTF calculation



VTF loss is defined as the ratio of the Receiver voltage and the Source voltage, as shown in the equations below:

$$L(f) = 20\log 10\left|\frac{V_r(f)}{V_s(f)}\right|$$

$$L(0) = 20\log 10\left(\frac{R_{rx}}{R_{tx} + R_{channel} + R_{rx}}\right)$$

L(f) is the frequency dependent loss and L(0) is the DC loss. For unterminated channel, L(0) is 0.

VTF crosstalk is defined as the power sum of the ratios of the aggressor Receiver voltage to the source voltage. 19 aggressors are included in the calculation. Based on crosstalk reciprocity, VTF crosstalk can be expressed as

$$XT(f) = 10\log 10\left(\sum_{i=1}^{19}\left|\frac{V_{ai}(f)}{V_s(f)}\right|^2\right)$$

5.7.2 Advanced Package

Table 38. Channel Characteristics

Data Rate	4-16Gb/s	24, 32 Gb/s
VTF Loss (dB)	$L(f_N) > -3$	$L(f_N) > -5$
VTF Crosstalk (dB)[1]	$XT(f_N) < 1.5\ L(f_N) - 21.5$ and $XT(f_N) < -23$	$XT(f_N) < 1.5\ L(f_N) - 19$ and $XT(f_N) < -24$

1. Based on Voltage Transfer Function Method (Tx: 25 ohm / 0.25 pF; Rx: 0.2 pF).

f_N is the Nyquist frequency. The equations in the table form a segmented line in 2-D map of loss and crosstalk, defining the pass/fail region.

Table 39. Advanced Package module signal list

Signal Name	Count	Description
Data		
TXDATA[63:0]	64	Transmit Data
TXVLD	1	Transmit Data Valid; Enables clocking in corresponding module
TXTRK	1	Transmit track signal
TXCKP	1	Transmit Clock phase 1
TXCKN	1	Transmit Clock phase 2
TXCKRD	1	Redundant for clock and track Lane repair
TXRD[3:0]		
RXDATA[63:0]	64	Receive Data
RXVLD	1	Receive Data Valid; Enables clocking in corresponding module
RXTRK	1	Receive track.
RXCKP	1	Receive Clock phase 1
RXCKN	1	Receive Clock phase 2
RXRD[3:0]	4	Redundant for data Lane repair
RXCKRD	1	Redundant for clock Lane repair
Sideband		
TXDATASB	1	Sideband Transmit Data
RXDATASB	1	Sideband Receiver Data
TXCKSB	1	Sideband Transmit Clock
RXCKSB	1	Sideband Receive Clock
TXDATASBRD	1	Redundant Sideband Transmit Data

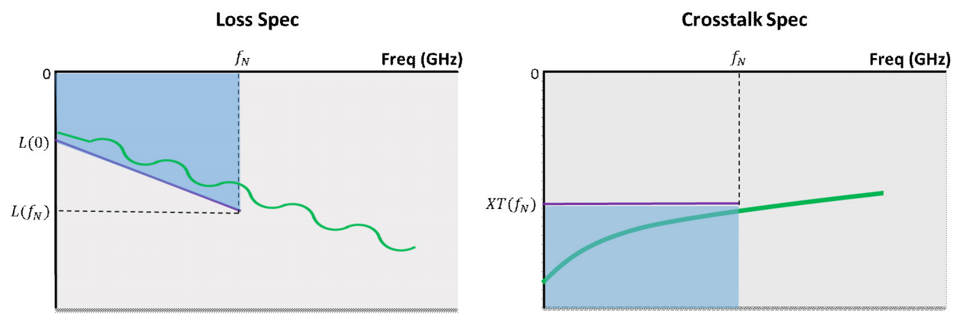
Table 39. Advanced Package module signal list

Signal Name	Count	Description
RXDATASBRD	1	Redundant Sideband Receiver Data
TXCKSBRD	1	Redundant Sideband Transmit Clock
RXCKSBRD	1	Redundant Sideband Receive Clock
Power and Voltage		
VSS		Ground Reference
VCCIO		IO supply
VCCAON		Always on Aux supply (sideband)

5.7.2.1 Loss and Crosstalk Mask

Loss and Crosstalk are specified by a mask defined by the $L(f_N)$ and $XT(f_N)$ at Nyquist frequency. It is a linear mask from DC to f_N for loss and flat mask for crosstalk, illustrated by Figure 85. Loss from DC to f_N needs to be above the spec line. Crosstalk from DC to f_N needs to be below the spec line.

Figure 85. Loss and Crosstalk mask



5.7.2.2 Advanced Package Module Bump Map

Figure 86 shows the reference bump matrix for an Advanced Package module.

It is strongly recommended to follow the bump matrix provided in Figure 86 for Advanced Package interface and the signal exit order is shown in Figure 87. The lower left corner of the bump map will be considered "origin" of a bump matrix.

The following rules must be followed for Advanced Package bump matrix.

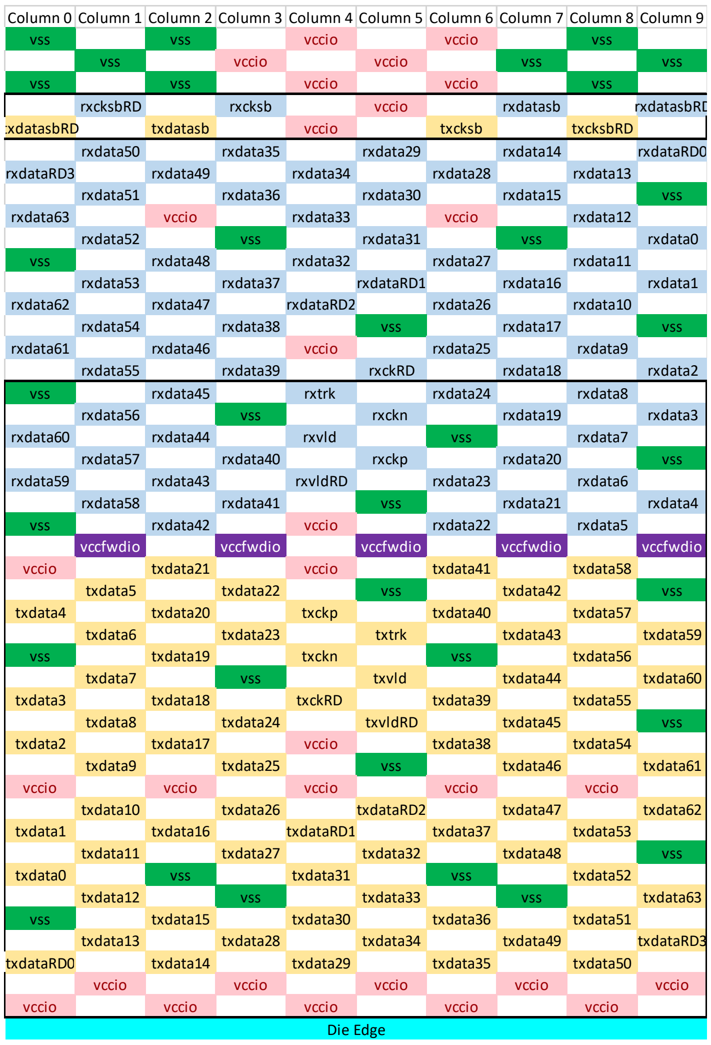
- The signals within a column must be preserved. For example, Column 0 must contain the signals: txdataRD0, txdata0, txdata1, txdata2, txdata3, txdata4, , , rxdata59, rxdata60, rxdata61, rxdata62, rxdata63, rxdataRD3 and txdatasbRD.
- It is strongly recommended to follow the supply and vss pattern shown in the bump matrices. It must be ensured that enough vss and supply bumps are provided to meet channel characteristics (FEXT and NEXT) and power delivery requirements.

Following rules must be followed for instantiating multiple modules of Advanced package bump matrix:

- When instantiating multiple modules, the modules must be stepped in the same orientation and abutted. Horizontal or vertical mirroring is not allowed
- Module stacking is not allowed
- In multi-module instantiations it is strongly recommended add one column of VSS bumps on each outside edge of the multi-module instantiation

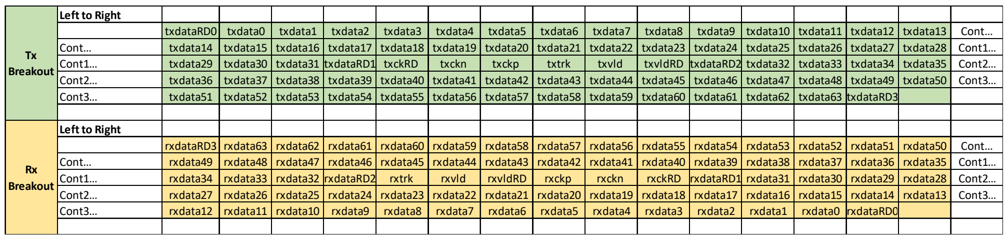
Flip die or dies from a flip wafer implementation may necessitate a jog or additional metal layers for proper connectivity.

Figure 86. Advanced Package Bump Map



Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9
vss		vss		vccio		vccio		vss	
	vss		vccio		vccio		vss		vss
vss		vss		vccio		vccio		vss	
	rxcksbRD		rxcksb		vccio		rxdatasb		rxdatasbRD
xdatasbRD		txdatasb		vccio		txcksb		txcksbRD	
	rxdata50		rxdata35		rxdata29		rxdata14		rxdataRD0
rxdataRD3		rxdata49		rxdata34		rxdata28		rxdata13	
	rxdata51		rxdata36		rxdata30		rxdata15		vss
rxdata63		vccio		rxdata33		vccio		rxdata12	
	rxdata52		vss		rxdata31		vss		rxdata0
vss		rxdata48		rxdata32		rxdata27		rxdata11	
	rxdata53		rxdata37		rxdataRD1		rxdata16		rxdata1
rxdata62		rxdata47		rxdataRD2		rxdata26		rxdata10	
	rxdata54		rxdata38		vss		rxdata17		vss
rxdata61		rxdata46		vccio		rxdata25		rxdata9	
	rxdata55		rxdata39		rxckRD		rxdata18		rxdata2
vss		rxdata45		rxtrk		rxdata24		rxdata8	
	rxdata56		vss		rxckn		rxdata19		rxdata3
rxdata60		rxdata44		rxvld		vss		rxdata7	
	rxdata57		rxdata40		rxckp		rxdata20		vss
rxdata59		rxdata43		rxvldRD		rxdata23		rxdata6	
	rxdata58		rxdata41		vss		rxdata21		rxdata4
vss		rxdata42		vccio		rxdata22		rxdata5	
	vccfwdio		vccfwdio		vccfwdio		vccfwdio		vccfwdio
vccio		txdata21		vccio		txdata41		txdata58	
	txdata5		txdata22		vss		txdata42		vss
txdata4		txdata20		txckp		txdata40		txdata57	
	txdata6		txdata23		txtrk		txdata43		txdata59
vss		txdata19		txckn		vss		txdata56	
	txdata7		vss		txvld		txdata44		txdata60
txdata3		txdata18		txckRD		txdata39		txdata55	
	txdata8		txdata24		txvldRD		txdata45		vss
txdata2		txdata17		vccio		txdata38		txdata54	
	txdata9		txdata25		vss		txdata46		txdata61
vccio		vccio		vccio		vccio		vccio	
	txdata10		txdata26		txdataRD2		txdata47		txdata62
txdata1		txdata16		txdataRD1		txdata37		txdata53	
	txdata11		txdata27		txdata32		txdata48		vss
txdata0		vss		txdata31		vss		txdata52	
	txdata12		vss		txdata33		vss		txdata63
vss		txdata15		txdata30		txdata36		txdata51	
	txdata13		txdata28		txdata34		txdata49		txdataRD3
txdataRD0		txdata14		txdata29		txdata35		txdata50	
	vccio		vccio		vccio		vccio		vccio
vccio		vccio		vccio		vccio		vccio	
Die Edge									

Figure 87. Advanced Package Bump map: Signal exit order



Tx Breakout	**Left to Right**																
		txdataRD0	txdata0	txdata1	txdata2	txdata3	txdata4	txdata5	txdata6	txdata7	txdata8	txdata9	txdata10	txdata11	txdata12	txdata13	Cont…
	Cont…	txdata14	txdata15	txdata16	txdata17	txdata18	txdata19	txdata20	txdata21	txdata22	txdata23	txdata24	txdata25	txdata26	txdata27	txdata28	Cont1…
	Cont1…	txdata29	txdata30	txdata31	txdataRD1	txckRD	txckn	txckp	txtrk	txvld	txvldRD	txdataRD2	txdata32	txdata33	txdata34	txdata35	Cont2…
	Cont2…	txdata36	txdata37	txdata38	txdata39	txdata40	txdata41	txdata42	txdata43	txdata44	txdata45	txdata46	txdata47	txdata48	txdata49	txdata50	Cont3…
	Cont3…	txdata51	txdata52	txdata53	txdata54	txdata55	txdata56	txdata57	txdata58	txdata59	txdata60	txdata61	txdata62	txdata63	txdataRD3		
Rx Breakout	**Left to Right**																
		rxdataRD3	rxdata63	rxdata62	rxdata61	rxdata60	rxdata59	rxdata58	rxdata57	rxdata56	rxdata55	rxdata54	rxdata53	rxdata52	rxdata51	rxdata50	Cont…
	Cont…	rxdata49	rxdata48	rxdata47	rxdata46	rxdata45	rxdata44	rxdata43	rxdata42	rxdata41	rxdata40	rxdata39	rxdata38	rxdata37	rxdata36	rxdata35	Cont1…
	Cont1…	rxdata34	rxdata33	rxdata32	rxdataRD2	rxtrk	rxvld	rxvldRD	rxckp	rxckn	rxckRD	rxdataRD1	rxdata31	rxdata30	rxdata29	rxdata28	Cont2…
	Cont2…	rxdata27	rxdata26	rxdata25	rxdata24	rxdata23	rxdata22	rxdata21	rxdata20	rxdata19	rxdata18	rxdata17	rxdata16	rxdata15	rxdata14	rxdata13	Cont3…
	Cont3…	rxdata12	rxdata11	rxdata10	rxdata9	rxdata8	rxdata7	rxdata6	rxdata5	rxdata4	rxdata3	rxdata2	rxdata1	rxdata0	rxdataRD0		

5.7.3 Standard Package

Interconnect channel should be design with 50 ohm characteristic impedance. Loss and crosstalk for requirement at Nyquist frequency with Receiver termination is defined by the Table 40.

Table 40. IL and xtalk for Standard Package: With Rx termination

Data Rate	4, 8 Gb/s	12, 16 Gb/s	24, 32 Gb/s
VTF Loss (dB)[1 2 3]	$L(0) > -4.5$ $L(f_N) > -7.5$	$L(0) > -4.5$ $L(f_N) > -6.5$	$L(0) > -4.5$ $L(f_N) > -7.5$
VTF Crosstalk (dB)	$XT(f_N) < 3 * L(f_N) - 11.5$ and $XT(f_N) < -25$	$XT(f_N) < 3 * L(f_N) - 11.5$ and $XT(f_N) < -25$	$XT(f_N) < 2.5 * L(f_N) - 10$ and $XT(f_N) < -26$

1. Voltage Transfer Function for 4,8Gb/s (Tx: 30 ohm / 0.3pF; Rx: 50 ohm / 0.3pF)
2. Voltage Transfer Function for 12,16Gb/s (Tx: 30 ohm / 0.2pF; Rx: 50 ohm / 0.2pF)
3. Voltage Transfer Function for 24,32Gb/s (Tx: 30 ohm / 0.125pF; Rx: 50 ohm / 0.125pF)

Loss and crosstalk for requirement at Nyquist frequency without Receiver termination is defined by Table 41. Loss and Crosstalk specifications between DC and Nyquist f_N follow the same methodology defined in section 5.7.2.1

Table 41. IL and xtalk for Standard Package: No Rx termination

Data Rate	4-12 Gb/s	16 Gb/s
VTF Loss (dB)[1 2]	$L(f_N) > -1.25$	$L(f_N) > -1.15$
VTF Crosstalk (dB)	$XT(f_N) < 7 * L(f_N) - 12.5$ and $XT(f_N) < -15$	$XT(f_N) < 4 * L(f_N) - 13.5$ and $XT(f_N) < -17$

1. Voltage Transfer Function for 4, 8Gb/s (Tx: 30 ohm / 0.3pF; Rx: 0.2 pF)
2. Voltage Transfer Function for 12,16Gb/s (Tx: 30 ohm / 0.2pF; Rx: 0.2 pF)

Table 42. Standard Package module signal list

Signal Name	Count	Description
		Data
TXDATA[15:0]	16	Transmit Data
TXVLD	1	Transmit Data Valid; Enables clocking in corresponding module
TXTRK	1	Transmit track signal
TXCKP	1	Transmit Clock phase 1
TXCKN	1	Transmit Clock phase 2
RXDATA[15:0]	16	Receive Data
RXVLD	1	Receive Data Valid; Enables clocking in corresponding module
RXTRK	1	Receive track.

Table 42. Standard Package module signal list

Signal Name	Count	Description
RXCKP	1	Receive Clock phase 1
RXCKN	1	Receive Clock phase 2
		Sideband
TXDATASB	1	Sideband Transmit Data
RXDATASB	1	Sideband Receiver Data
TXCKSB	1	Sideband Transmit Clock
RXCKSB	1	Sideband Receive Clock
		Power and Voltage
VSS		Ground Reference
VCCIO		IO supply
VCCAON		Always on Aux supply (sideband)

5.7.3.1 Standard Package Module Bump Map

Figure 88 and Figure 90 show the reference bump matrices for an x16 (one module) and x32 (two module) Standard Package interfaces respectively.

It is strongly recommended to follow the bump matrices provided in Figure 88 for one module and Figure 90 for two module Standard package. The lower left corner of the bump map will be considered “origin” of a bump matrix.

Signal exit order for x16 and x32 Standard Package bump matrices are shown in Figure 89 and Figure 91 respectively.

The following rules must be followed for Standard Package bump matrix.

- The signals within a column must be preserved. For example, for a x16 (one module Standard Package interface) shown in Figure 88, Column 1 must contain the signals: txdata0, txdata1, txdata9, txdata9 and txdatasb1.
- The signals must exit the bump field in the order shown in Figure 89. Layer 1 and Layer 2 are two different signal routing layers in a standard package.
- It is strongly recommended to follow the supply and vss pattern shown in the bump matrices. It must be ensured that enough vss and supply bumps are provided to meet channel characteristics (FEXT and NEXT) and power delivery requirements.

Following rules must be followed for instantiating multiple modules of Standard package bump matrix:

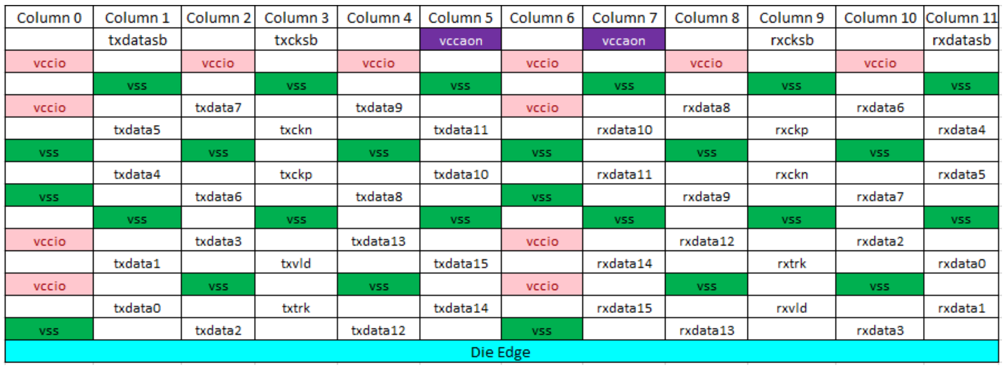
- Tx must be closer to die dwelling edge (top view)
- When instantiating multiple modules, the modules must be stepped in the same orientation and abutted. Horizontal or vertical mirroring is not allowed

If more bandwidth per shoreline is required, it is permitted to stack two modules before abutting. If two modules are stacked, the package may need to support at least four routing layers for UCIe signal routing. An example of stacked standard package modules instantiations is shown in Figure 90

- If only one stacked module is instantiated, Tx must be closer to die dwelling edge (top view)
- When instantiating multiple stacked modules, the modules must be stepped in the same orientation and abutted. Horizontal or vertical mirroring is not allowed

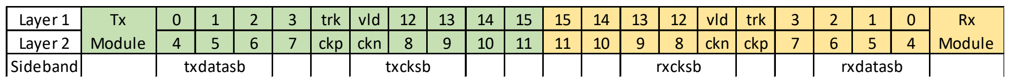
Note: An example of signal routing for stacked module is shown in Figure 92

Figure 88. Standard Package Bump Map: x16 interface



Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11
	txdatasb		txcksb		vccaon		vccaon		rxcksb		rxdatasb
vccio		vccio		vccio		vccio		vccio		vccio	
	vss		vss		vss		vss		vss		vss
vccio		txdata7		txdata9		vccio		rxdata8		rxdata6	
	txdata5		txckn		txdata11		rxdata10		rxckp		rxdata4
vss		vss		vss		vss		vss		vss	
	txdata4		txckp		txdata10		rxdata11		rxckn		rxdata5
vss		txdata6		txdata8		vss		rxdata9		rxdata7	
	vss		vss		vss		vss		vss		vss
vccio		txdata3		txdata13		vccio		rxdata12		rxdata2	
	txdata1		txvld		txdata15		rxdata14		rxtrk		rxdata0
vccio		vss		vss		vccio		vss		vss	
	txdata0		txtrk		txdata14		rxdata15		rxvld		rxdata1
vss		txdata2		txdata12		vss		rxdata13		rxdata3	
					Die Edge						

Figure 89. Standard Package x16 interface: Signal exit order



Layer 1	Tx	0	1	2	3	trk	vld	12	13	14	15	15	14	13	12	vld	trk	3	2	1	0	Rx
Layer 2	Module	4	5	6	7	ckp	ckn	8	9	10	11	11	10	9	8	ckn	ckp	7	6	5	4	Module
Sideband			txdatasb					txcksb							rxcksb					rxdatasb		

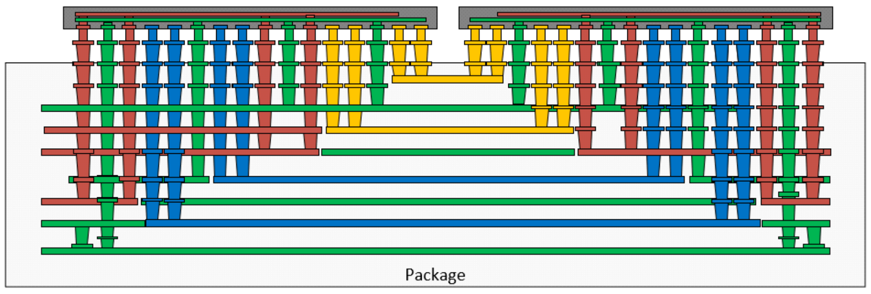
Figure 90. Standard Package Bump Map: x32 interface

Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11
	m2rxdatasb		m2rxcksb		vccaon		m2txcksb		m2txdatasb		vccaon
m1txdatasb		m1txcksb		vccaon		vccaon		m1rxcksb		m1rxdatasb	
	vccio		vccio		vccio		vccio		vccio		vccio
vss		vss		vss		vss		vss		vss	
	m2rxdata6		m2rxdata8		vss		m2txdata9		m2txdata7		vss
m2rxdata4		m2rxckp		m2rxdata10		m2txdata11		m2txckn		m2txdata5	
	vss		vss		vss		vss		vss		vss
m2rxdata5		m2rxckn		m2rxdata11		m2txdata10		m2txckp		m2txdata4	
	m2rxdata7		m2rxdata9		vss		m2txdata8		m2txdata6		vss
vss		vss		vss		vss		vss		vss	
	m2rxdata2		m2rxdata12		vss		m2txdata13		m2txdata3		vss
m2rxdata0		m2rxtrk		m2rxdata14		m2txdata15		m2txvld		m2txdata1	
	vss		vss		vss		vss		vss		vss
m2rxdata1		m2rxvld		m2rxdata15		m2txdata14		m2txtrk		m2txdata0	
	m2rxdata3		m2rxdata13		vccio		m2txdata12		m2txdata2		vccio
vccio		vccio		vccio		vccio		vccio		vccio	
	vss		vss		vccio		vss		vss		vccio
vccio		m1txdata7		m1txdata9		vccio		m1rxdata8		m1rxdata6	
	m1txdata5		m1txckn		m1txdata11		m1rxdata10		m1rxckp		m1rxdata4
vss		vss		vss		vss		vss		vss	
	m1txdata4		m1txckp		m1txdata10		m1rxdata11		m1rxckn		m1rxdata5
vss		m1txdata6		m1txdata8		vss		m1rxdata9		m1rxdata7	
	vss		vss		vss		vss		vss		vss
vccio		m1txdata3		m1txdata13		vccio		m1rxdata12		m1rxdata2	
	m1txdata1		m1txvld		m1txdata15		m1rxdata14		m1rxtrk		m1rxdata0
vccio		vss		vss		vccio		vss		vss	
	m1txdata0		m1txtrk		m1txdata14		m1rxdata15		m1rxvld		m1rxdata1
vss		m1txdata2		m1txdata12		vss		m1rxdata13		m1rxdata3	
Die Edge											

Figure 91. Standard Package x32 interface: Signal exit routing

Layer 1	Tx	0	1	2	3	trk	vld	12	13	14	15	15	14	13	12	vld	trk	3	2	1	0	Rx
Layer 2	Module 1	4	5	6	7	ckp	ckn	8	9	10	11	11	10	9	8	ckn	ckp	7	6	5	4	Module 1
Layer 3	Rx	0	1	2	3	trk	vld	12	13	14	15	15	14	13	12	vld	trk	3	2	1	0	Tx
Layer 4	Module 2	4	5	6	7	ckp	ckn	8	9	10	11	11	10	9	8	ckn	ckp	7	6	5	4	Module 2
Sideband		m1txdatasb			m2rxdatasb			m1txcksb		m2rxcksb		m2txcksb		m1rxcksb		m2txdatasb			m1rxdatasb			Sideband

Figure 92. Standard Package cross section for stacked module

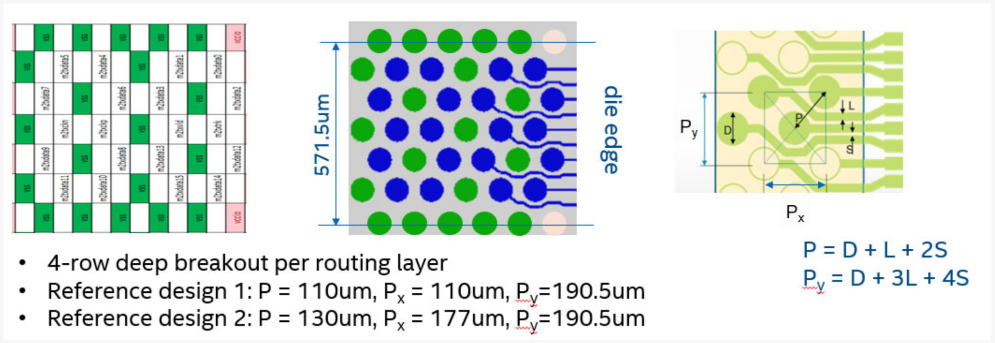


Orange – Tx signals on Die 1 talking to Rx on Die 2 in Layer 1
Orange - Tx signals on Die 1 talking to Rx on Die 2 in Layer 2
Blue - Tx signals on Die 2 talking to Rx on Die 1 in Layer 3
Blue - Tx signals on Die 2 talking to Rx on Die 1 in Layer 4
Brown – VCCIO
Green - VSS

IMPLEMENTATION NOTE:

Figure 93 is a standard package channel reference configuration noting suggested design rule ratios. The 110um dimension is reference to a diagonal pitch, the horizontal and vertical pitches can vary as long as minimum design rules aren't violated. In the example, the horizontal is a multiple of 190.5um.

Figure 93. Standard Package reference configuration



- 4-row deep breakout per routing layer
- Reference design 1: P = 110um, P_x = 110um, P_y=190.5um
- Reference design 2: P = 130um, P_x = 177um, P_y=190.5um

P = D + L + 2S
P_y = D + 3L + 4S

5.8 Tightly-coupled mode

Tightly-Coupled PHY mode is defined as when both of the following conditions are met.

- Shared Power Supply between TX and RX, or Forwarded Power Supply from TX to RX.
- Channel supports larger eye mask defined in Table 43

In this mode, there is no Receiver termination and Transmitter must provide full swing output. In this mode further optimization of PHY circuit and power reduction is possible. For example, tuned inverter can potentially used instead of a front-end amplifier. Training complexity such as voltage reference can be simplified.

Table 43. Tightly coupled mode: Eye mask

Data Rate	**4-16Gb/s**
Overall (Eye Closure due to Channel)[1]	
Eye Height	250mV
Eye Width (rectangular eye mask with specified eye height)	0.7 UI

1. With 750 mV Transmitter signal swing

Loss and Crosstalk requirement follow the same VTF method, adjusting to the eye mask defined in table Table 43. Table 44 show the specification at Nyquist frequency

Table 44. Tightly coupled mode channel for Advanced package

Data Rate	4-12 Gb/s	16 Gb/s
VTF Loss[1] (dB)	$L(f_N) > -3$	-
VTF Crosstalk[1] (dB)	$XT(f_N) < 1.5 * L(f_N) - 21.5$ and $XT(f_N) < -23$	-

1. Based on Voltage Transfer Function (Tx: 25 ohm / 0.25pF; Rx: 0.2pF)

Loss and Crosstalk specifications between DC and Nyquist f_N follow the same methodology defined in section 5.7.2.1

Primary intend for this mode is for advanced package, though it's also possible standard package when two dies are close to each other and no Receiver termination is required.

5.9 Interconnect redundancy remapping

5.9.1 Advanced Package Lane remapping

Interconnect Lane remapping is supported in Advanced Package module to improve assembly yield and recover functionality. Each module supports

- Four redundant bumps for Data
- One redundant bump for Clock and Track.
- One redundant bump for Valid

The four redundant bumps for data repair are divided into two groups of two. Figure 94 shows an illustration of redundant bump assignment for data signals. TRD_P0 and TRD_P1 are allocated to lower 32 data Lanes and TRD_P2 and TRD_P3 are allocated to upper 32 data Lanes. Each group is allowed to remap up to two Lanes.As an example, TD15 is a broken Lane in lower half and TD_P32 & TD_P40 are broken Lanes in upper 32 Lane. Figure 95 illustrates Lane remapping for the broken Lanes.

Details and implementation of Lane remapping for Data, Clock, Track and Valid are provided in section 4.3

Figure 94. Data Lane repair resources

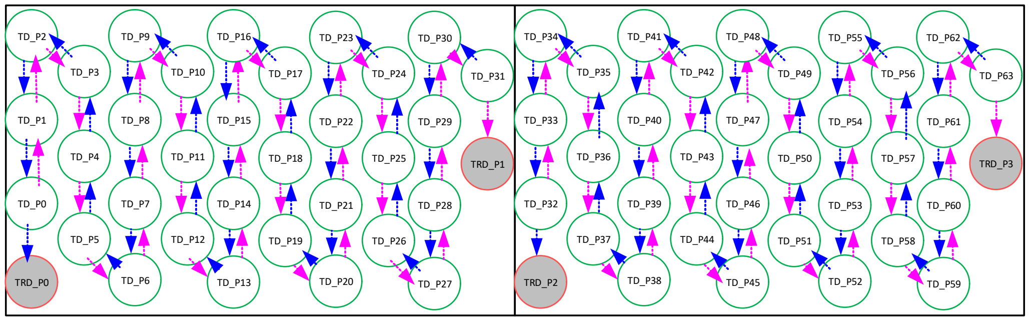
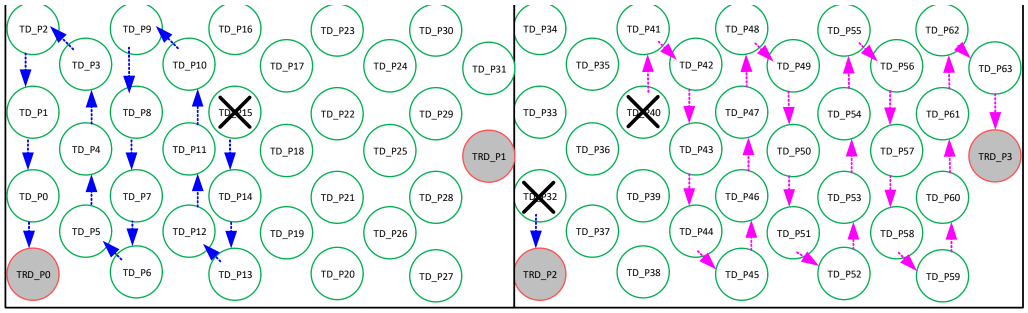


Figure 95. Data Lane repair



5.9.2 Standard Package Lane remapping

Lane repair is not supported in Standard Package modules.

5.10 BER requirements, CRC and retry

The BER requirement based on channel reach defined in section 5.7 is shown in Table 45. Error detection and correction mechanisms such as CRC and retry are required for BER for 1E-15 to achieve the required Failure In Time (FIT) rate of significantly less than 1 (1 FIT = 1 device failure in 10^9 Hours). The UCIe spec defined CRC and retry is detailed in Chapter 3.0. For the BER of 1E-27, either parity or CRC can be used and the appropriate error reporting mechanism must be invoked to ensure a FIT that is significantly less than 1.

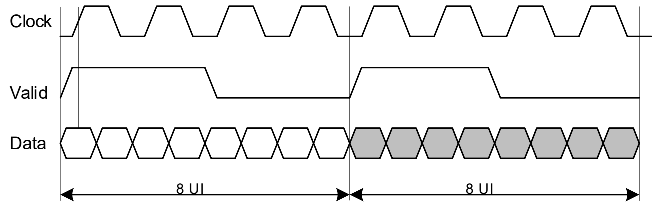
Table 45. Raw BER requirements

Data Rate (GT/s)	4	8	12	16	24	32
Advanced Package	1E-27	1E-27	1E-27	1E-15	1E-15	1E-15
Standard Package	1E-27	1E-27	1E-15	1E-15	1E-15	1E-15

5.11 Valid and Clock Gating

Valid is used to frame transmit data. For a single transmission of 8 UI data packet, Valid is asserted for the first 4 UI and de-asserted for the second 4 UI. Figure 96 shows the transfer of two 8 UI data packets back to back.

Figure 96. Valid Framing



As described in section 4.1.3, clock must be gated when Valid signal is low after providing fixed 16UI (8 cycles) of postamble clock unless free running clock mode is negotiated.

Idle state is when there is not data transmission on the main band. During Idle state, Data, Clock and Valid Lanes must hold values as follows:

- If the Link is unterminated (all Advanced Package and unterminated Standard package Links), Data Lane Transmitters must hold the last transmitted UI. Valid Lane must be held low. Clock level in idle state (after meeting postamble requirement) must alternate between differential high and differential low during consecutive clock gating events. Clock must drive a differential low for at least 1 UI or a maximum of 8 UI before normal operation. Example shown in Figure 97
- If the Link is terminated (Standard Package terminated Links), Data Lanes Transmitters must send the last UI for at least 1UI and up to 8UIs and then Hi-z. Valid Lane must be held low. Clock idle state level must alternate between differential high and differential low during consecutive clock gating events. Transmitters must precondition the Data Lanes to a 0 or 1 and clock must drive a differential low for at least 1 UI or up to a maximum of 8UIs before the normal transmission. Example shown in Figure 98.

Figure 97. Data, Clock, Valid Gated Levels: Unterminated Link

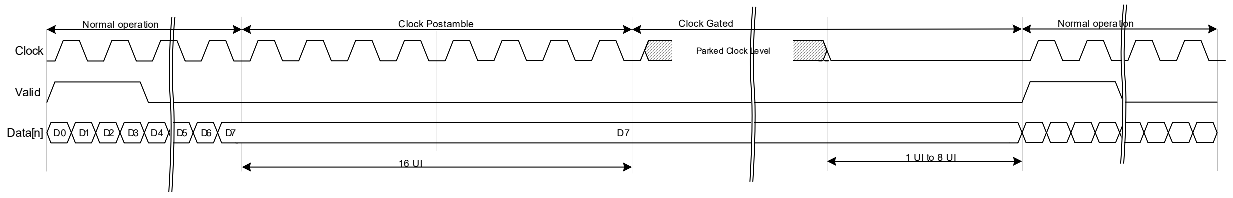
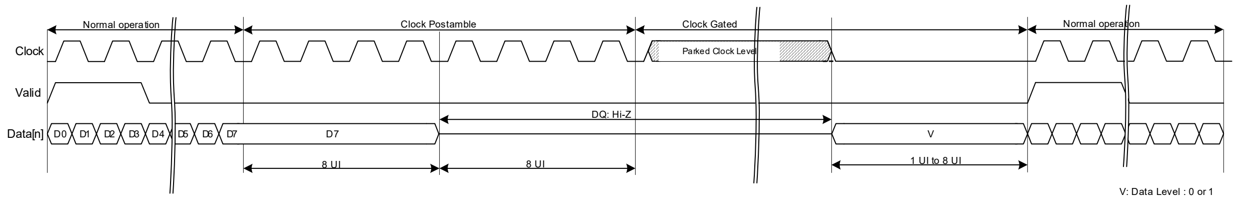


Figure 98. Data, Clock, Valid Gated Levels: Terminated Link



5.12 Electrical Idle

Some training states need electrical idle when Transmitters and Receivers are waiting for generate and receive patterns.

- Electrical idle on the mainband in this specification is describes as Transmitters and Receivers are enabled; Data, Valid and Track Lanes are held low and Clock is parked at high and low

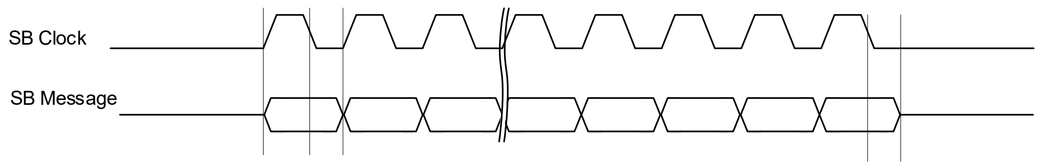
5.13 Sideband signaling

Each module supports a sideband interface. The sideband is a two signal interface for transmit and receive direction. The sideband data is a 800 MT/s single data rate signal (SDR) with 800 MHz source. Sideband must run on power supply and auxiliary clock source which are always on.

Sideband data is sent edge aligned with the strobe. The Receiver must sample the incoming data with the strobe. For example, the negative edge of the strobe can be used to sample the data as the data uses SDR signaling as shown in Figure 99.

For Advanced Package modules, redundancy is supported for the sideband interface. Sideband initialization and repair are described in section 4.5.3.2. There is no redundancy and Lane repair support on Standard Package modules.

Figure 99. Sideband signaling



5.13.1 Sideband Electrical Parameters

Table 46 shows the sideband electrical parameters.

It is strongly recommended the two sides of the sideband IO Link share the same power supply rail

Table 46. Sideband Parameters summary

Parameter	Min	Typ	Max	Unit
Supply voltage (VCCAON)[1]	0.65			V
TX Swing	0.8*VCCAON	-	-	V
Input high voltage (V_{IH})	0.7*VCCAON			V
Input low voltage (V_{IL})			0.3*VCCAON	V
Output high voltage (V_{OH})	0.9*VCCCAON			V
Output low voltage (V_{OL})			0.1*VCCAON	V
Sideband Data Setup Time	200	-	-	ps
Sideband Data Hold Time	200	-	-	ps

1. Always On power supply

6.0 Sideband

6.1 Protocol specification

The usage for the sideband Link is to provide a back-channel for Link training and an interface for sideband access of registers of the Link partner. It is also used for Link management packets and parameter exchanges with remote Link partner.

The same protocol is also used for local die sideband accesses over FDI and RDI. When relevant, FDI specific rules are pointed out using "FDI sideband:". When relevant, RDI specific rules are pointed out using "RDI sideband:". When relevant, UCIe Link specific rules are pointed out using "UCIe Link sideband:". If no prefix is mentioned, it is a common rule across FDI, RDI and UCIe Link.

The Physical Layer is responsible for framing and transporting sideband packets over the UCIe Link. Direct sideband access to remote die can originate from the Adapter or the Physical Layer. The Adapter forwards a remote die sideband access over RDI to the Physical Layer for framing and transport. These include register access requests, completions or messages.

The Protocol Layer has indirect access to remote die registers using the sideband mailbox mechanism. The mailbox registers reside in the Adapter, and it is the responsibility of the Adapter to initiate remote die register access requests when it receives the corresponding access trigger for the mailbox register over FDI.

FDI sideband: In the case of multi-protocol stacks, the Adapter must track which protocol stack sent the original request and route the completion back to the appropriate protocol stack.

FDI sideband: Since the Protocol Layer is only allowed indirect access to remote die registers, and direct access to local die registers, currently only Register Access requests and completions are permitted on the FDI interface.

All sideband requests that expect a response have an 8ms timeout. A "Stall" encoding is provided for the relevant packets for Retimers, to prevent timeouts if the Retimer needs extra time to respond to the request. When stalling for preventing timeouts, it is the responsibility of the Retimer to send the corresponding Stall response once every 4ms. The Retimer must also ensure that it does not Stall indefinitely, and escalates a Link down event after a reasonable attempt to complete resolution that required stalling the requester. If a requester receives a response with a "Stall" encoding, it resets the timeout counter.

In certain cases, it is necessary for registers to be fragmented between the different layers; i.e., certain bits of a given register physically reside in the Protocol Layer, other bits reside in the Adapter, and other bits reside in the Physical Layer. UCIe takes a hierarchical decoding for these registers. For fragmented registers, if a bit does not physically reside in a given Layer, it implements that bit as Read Only and tied to 0. Hence reads would return 0 for those bits from that Layer, and writes would have no effect on those bits. As an example, for reads, Protocol Layer would forward these requests to the Adapter on FDI and the Protocol Layer will OR the data responded by the Adapter with its local register before responding to software. The Adapter must do the same if any bits of that register reside in the Physical Layer before responding to the Protocol Layer.

6.1.1 Packet Types

Four different types of packets are permitted:

- Register Accesses: These can be Configuration (CFG) or Memory Mapped Reads or Writes and can be 32-bit or 64-bit.
- Messages without data: These can be Link Management (LM), or Vendor Defined Packets. These don't carry additional data payloads.
- Messages with data: These can be Parameter Exchange (PE), Link Training related or Vendor Defined, and carry 64b of data.

Every packet carries a 5-bit opcode, 3-bit source identifier (**`srcid`**), and a 3-bit destination identifier (**`dstid`**). The 5-bit opcode indicates the packet type, as well as whether it carries 32b of data or 64b of data.

Table 47 gives the mapping of opcode encodings to Packet Types.

Table 47. Opcode encodings mapped to Packet Types

Opcode Encoding	Packet Type
00000b	32b Memory Read
00001b	32b Memory Write
00100b	32b Configuration Read
00101b	32b Configuration Write
01000b	64b Memory Read
01001b	64b Memory Write
01100b	64b Configuration Read
01101b	64b Configuration Write
10000b	Completion without Data
10001b	Completion with 32b Data
11001b	Completion with 64b Data
10010b	Message without Data
11011b	Message with 64b Data
Other encodings	Reserved

Table 48, Table 49 and Table give the encodings of source and destination identifiers. It is not permitted for Protocol Layer from one side of the Link to directly access Protocol Layer of the remote Link partner over sideband (this should be done via main-band).

Table 48. FDI sideband: srcid and dstid encodings on FDI

Field	Description
srcid[2:0]	000b: Stack 0 Protocol Layer 100b: Stack 1 Protocol Layer other encodings are reserved.
dstid[2:0]	001b: D2D Adapter 010b: Physical Layer other encodings are reserved.

Note: FDI sideband: srcid and dstid are Reserved for completion messages transferred over FDI. The Protocol Layer must correlate the completions to original requests using the Tag field. Currently, no requests are permitted from Adapter to Protocol Layer over FDI sideband.

Table 49. RDI sideband: srcid and dstid encodings on RDI

Field	Description
srcid[2:0]	000b: Stack 0 Protocol Layer 001b: D2D Adapter 100b: Stack 1 Protocol Layer other encodings are reserved.
dstid[2]	0b: Local die terminated request 1b: Remote die terminated request
dstid[1:0]	For Local die terminated requests: 10b: Physical Layer other encodings are reserved. For Remote die terminated Register Access Requests: dstid[1:0] is Reserved For Remote die terminated Register Access Completions: 01b: D2D Adapter other encodings are reserved. For Remote die terminated messages: 01b: D2D Adapter message 10b: Physical Layer message

UCIe Link sideband: srcid and dstid encodings for UCIe Link

Field	Description
srcid[2:0]	001b: D2D Adapter 010b: Physical Layer other encodings are reserved
dstid[2]	1b: Remote die terminated request other encodings are reserved
dstid[1:0]	For Register Access requests: dstid[1:0] is Reserved. For Remote die terminated Register Access Completions: 01b: D2D Adapter other encodings are reserved. For Remote die terminated messages: 01b: D2D Adapter message 10b: Physical Layer message

6.1.2 Packet Formats

All the figures in this section show examples assuming a 32-bit interface of RDI/FDI transfer for sideband packets, hence the headers and data are shown in Phases of 32-bits.

6.1.2.1 Register Access Packets

Figure 100 shows the packet format for Register Access requests. Table 50 gives the description of the fields other than the opcode, srcid and dstid.

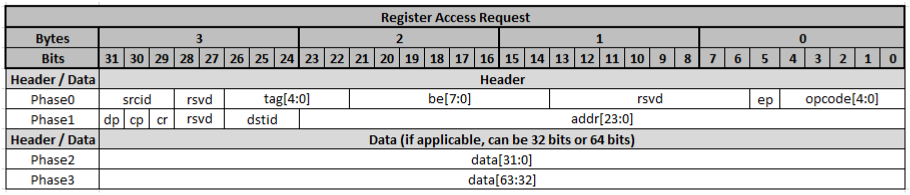
Table 50. Field descriptions for Register Access Requests

Field	Description
`CP`	Control Parity (CP) is the even parity of all the header bits excluding DP.
`DP`	Data Parity is the even parity of all bits in the data payload. If there is no data payload, this bit is set to 0b.
`Cr`	If 1b, indicates one credit return for credited sideband messages. This field is only used by the Adapter for remote Link partner's credit returns for E2E credits. It is not used for local FDI or RDI credit loops.
`Addr`[26:0]	Address of the request. Different opcodes use this field differently. See Table 51 for details. The following rules apply for the address field: For 64-bit request, Addr[2:0] is reserved. For 32-bit request, Addr[1:0] is reserved.
`BE`[7:0]	Byte Enables for the Request. It is NOT required to be contiguous. BE[7:4] are reserved if the opcode is for a 32-bit request.
`EP`	Data Poison. If poison forwarding is enabled, the completer can poison the data on internal errors.
`Tag`[4:0]	Tag is a 5-bit field generated by the requester, and it must be unique for all outstanding requests that require a completion. The original requester uses the Tag to associate returning completions with the original request.
`Data`	Payload. Can be 32 bits or 64 bits wide depending on the Opcode.

Table 51. Mapping of Addr[23:0] for different requests

Opcode	Description
Memory Reads/Writes	{RL[3:0] Offset[19:0]} Offset is the Byte Offset. RL[3:0] encodings are as follows: 0h: Register Locator 0 1h: Register Locator 1 2h: Register Locator 2 Fh: Accesses for Protocol specific MMIO registers that are shadowed in the Adapter (for example, ARB/MUX registers defined in the CXL specification). The offsets for these registers are implementation specific, and the protocol layer must translate accesses to match the offsets implemented in the Adapter. Other encodings are reserved. For accesses to Reserved RL encodings, the completer must respond with a UR.
Configuration Reads/Writes	{RL[3:0], Rsvd[7:0], Byte Offset[11:0]}, where RL[3:0] encodings are as follows: 0h: UCIe Link DVSEC Fh: Accesses for Protocol specific configuration registers that are shadowed in the Adapter (for example, ARB/MUX registers defined in the CXL specification). The offsets for these registers are implementation specific, and the protocol layer must translate accesses to match the offsets implemented in the Adapter. Other encodings are reserved. For accesses to Reserved RL encodings, the completer must respond with a UR.

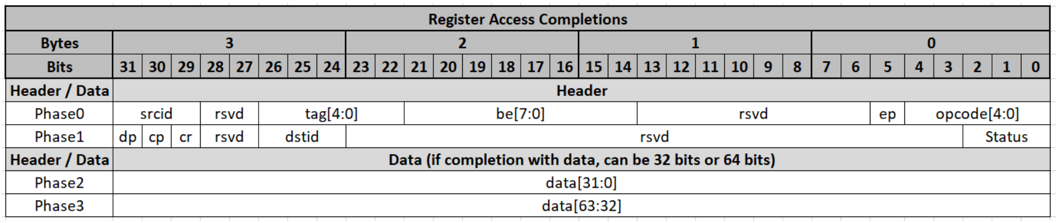
Figure 100. Format for Register Access request



Register Access Request																																
Bytes	3								2								1								0							
Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Header / Data	Header																															
Phase0	srcid			rsvd		tag[4:0]					be[7:0]								rsvd								ep	opcode[4:0]				
Phase1	dp	cp	cr	rsvd		dstid			addr[23:0]																							
Header / Data	Data (if applicable, can be 32 bits or 64 bits)																															
Phase2	data[31:0]																															
Phase3	data[63:32]																															

Figure 101 gives the format for Register Access completions.

Figure 101. Format for Register Access completions



Register Access Completions																																
Bytes	**3**								**2**								**1**								**0**							
Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Header / Data	**Header**																															
Phase0	srcid			rsvd		tag[4:0]					be[7:0]								rsvd								ep	opcode[4:0]				
Phase1	dp	cp	cr	rsvd		dstid			rsvd																					Status		
Header / Data	**Data (if completion with data, can be 32 bits or 64 bits)**																															
Phase2	data[31:0]																															
Phase3	data[63:32]																															

Table 52 gives the field descriptions for a completion.

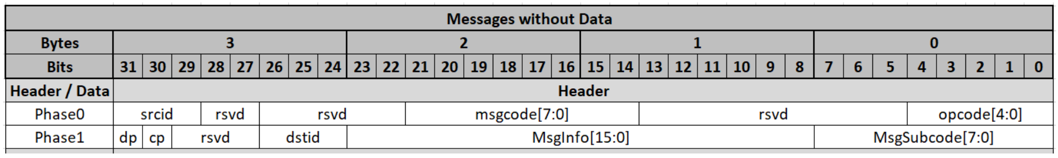
Table 52. Field Descriptions for a Completion

Field	Description
Tag[4:0]	Completion Tag associated with the corresponding Request. The requester uses this to associate the completion with the original request.
CP	Control Parity. All fields other than "DP" and "CP" in the Header are protected by Control Parity, and the parity scheme is even (including reserved bits)
DP	Data Parity. All fields in data are protected by data parity, and the parity scheme is even.
Cr	If 1b, indicates one credit return for credited sideband messages. This field is only used by the Adapter for remote Link partner's credit returns for E2E credits. It is not used for local FDI or RDI credit loops.
EP	Data Poison. If poison forwarding is enabled, the completer can poison the data on internal errors.
BE[7:0]	Byte Enables for the Request. Completer returns the same value that the original request had (this avoids the requester from having to save off the BE value). BE[7:4] are reserved if the opcode is for a 32-bit request.
Status[2:0]	Completion Status 000b - Successful Completion (SC). This can be a completion with or without data, depending on the original request (it must set the appropriate Opcode). If the original request was a write, it is a completion without data. If the original request was a read, it is a completion with data. 001b - Unsupported Request (UR). On UCIe, this is a completion with 64b Data when a request is aborted by the completer, and the Data carries the original request header that resulted in UR. This enables easier header logging at the requester. Register Access requests that timeout must also return UR status, but for those the completion is without Data. 100b - Completer Abort (CA). On UCIe, this is a completion with 64b Data, and the Data carries original request header that resulted in UR. This enables easier header logging at the requester. 111b - Stall. Receiving a completion with Stall encoding must reset the timeout at the requester. Completer must send a Stall once every 4ms if it is not ready to respond to the original request. Other encodings are reserved. An error is logged in the Sideband Mailbox Status if a CA was received or if the number of timeouts exceed the programmed threshold. For timeouts below the programmed threshold, a UR is returned to the requester.
Data	Payload. 32 bits or 64 bits depending on the Opcode.

6.1.2.2 Messages without Data

Figure 102 shows the Format for Messages without data payloads. These can be Link Management packets, NOPs or Vendor Defined message packets.

Figure 102. Format for Messages without Data



Messages without Data																																
Bytes	3								2								1								0							
Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Header / Data	Header																															
Phase0	srcid			rsvd		rsvd					msgcode[7:0]								rsvd									opcode[4:0]				
Phase1	dp	cp	rsvd			dstid			MsgInfo[15:0]																MsgSubcode[7:0]							

The definitions of opcode, srcid, dstid, dp, cp, ak and sn fields are the same as Register Access packets. Table 53 and Table 54 give the encodings of the different messages without data that are send on UCIe. Some Notes on the different message categories are listed below:

- {NOP.Crd} - These are used for E2E Credit returns. The destination must be D2D Adapter.
- {LinkMgmt.RDI.*} - These are used to coordinate RDI state transitions, the source and destination is Physical Layer.
- {LinkMgmt.Adapter0.*} - These are used to coordinate Adapter LSM state transitions for the Adapter LSM corresponding to Stack 0 Protocol Layer. The source and destination is D2D Adapter.
- {LinkMgmt.Adapter1.*} - These are used to coordinate Adapter LSM state transitions for the Adapter LSM corresponding to Stack 1 Protocol Layer. The source and destination is D2D Adapter.
- {ParityFeature.*} - This is used to coordinate enabling of the Parity insertion feature. The source and destination for this must be the D2D Adapter.
- {ErrMsg} - This is used for error reporting and escalation from the remote Link Partner. This is sent from the Retimer or Device die to the Host, and the destination must be the D2D Adapter.

Table 53. Message Encodings

Name	Msgcode	Msgsubcode	MsgInfo	Description
{Nop.Crd}	00h	00h	0000h:Reserved 0001h:1 Credit return 0002h: 2 Credit returns 0003h: 3 Credit returns 0004h: 4 Credit returns	Explicit Credit return from Remote Link partner for credited messages.

Name	Msgcode	Msgsubcode	MsgInfo	Description
{LinkMgmt.RDI.Req.Active}	01h	01h	Reserved	Active Request for RDI SM.
{LinkMgmt.RDI.Req.L1}		04h		L1 Request for RDI SM.
{LinkMgmt.RDI.Req.L2}		08h		L2 Request for RDI SM.
{LinkMgmt.RDI.Req.LinkReset}		09h		LinkReset Request for RDI SM.
{LinkMgmt.RDI.Req.LinkError}		0Ah		LinkError Request for RDI SM.
{LinkMgmt.RDI.Req.Retrain}		0Bh		Retrain Request for RDI SM.
{LinkMgmt.RDI.Req.Disable}		0Ch		Disable Request for RDI SM.
{LinkMgmt.RDI.Rsp.Active}	02h	01h	0000h: Regular Response FFFFh: Stall Response	Active Response for RDI SM.
{LinkMgmt.RDI.Rsp.PMNAK}		02h		PMNAK Response for RDI SM
{LinkMgmt.RDI.Rsp.L1}		04h		L1 Response for RDI SM.
{LinkMgmt.RDI.Rsp.L2}		08h		L2 Response for RDI SM.
{LinkMgmt.RDI.Rsp.LinkReset}		09h		LinkReset Response for RDI SM.
{LinkMgmt.RDI.Rsp.LinkError}		0Ah		LinkError Response for RDI SM.
{LinkMgmt.RDI.Rsp.Retrain}		0Bh		Retrain Response for RDI SM.
{LinkMgmt.RDI.Rsp.Disable}		0Ch		Disable Response for RDI SM.
{LinkMgmt.Adapter0.Req.Active}	03h	01h	0000h: Regular Request FFFFh: Stall	Active Request for Stack 0 Adapter LSM. The Stall encoding is provided for Retimers to avoid the Adapter LSM transition to Active timeout as described in section 7.5.3.8.
{LinkMgmt.Adapter0.Req.L1}		04h	Reserved	L1 Request for Stack 0 Adapter LSM.
{LinkMgmt.Adapter0.Req.L2}		08h		L2 Request for Stack 0 Adapter LSM.
{LinkMgmt.Adapter0.Req.LinkReset}		09h		LinkReset Request for Stack 0 Adapter LSM.
{LinkMgmt.Adapter0.Req.Disable}		0Ch		Disable Request for Stack 0 Adapter LSM.

Name	Msgcode	Msgsubcode	MsgInfo	Description
{LinkMgmt.Adapter0.Rsp.Active}	04h	01h	0000h: Regular Response FFFFh: Stall Response	Active Response for Stack 0 Adapter LSM.
{LinkMgmt.Adapter0.Rsp.PMNAK}		02h		PMNAK Response for Stack 0 Adapter LSM.
{LinkMgmt.Adapter0.Rsp.L1}		04h		L1 Response for Stack 0 Adapter LSM.
{LinkMgmt.Adapter0.Rsp.L2}		08h		L2 Response for Stack 0 Adapter LSM.
{LinkMgmt.Adapter0.Rsp.LinkReset}		09h		LinkReset Response for Stack 0 Adapter LSM.
{LinkMgmt.Adapter0.Rsp.Disable}		0Ch		Disable Response for Stack 0 Adapter LSM.
{LinkMgmt.Adapter1.Req.Active}	05h	01h	0000h: Regular Request FFFFh: Stall	Active Request for Stack 1 Adapter LSM. The Stall encoding is provided for Retimers to avoid the Adapter LSM transition to Active timeout as described in section 7.5.3.8.
{LinkMgmt.Adapter1.Req.L1}		04h	Reserved	L1 Request for Stack 1 Adapter LSM.
{LinkMgmt.Adapter1.Req.L2}		08h		L2 Request for Stack 1 Adapter LSM.
{LinkMgmt.Adapter1.Req.LinkReset}		09h		LinkReset Request for Stack 1 Adapter LSM.
{LinkMgmt.Adapter1.Req.Disable}		0Ch		Disable Request for Stack 1 Adapter LSM.
{LinkMgmt.Adapter1.Rsp.Active}	06h	01h	0000h: Regular Response FFFFh: Stall Response	Active Response for Stack 1 Adapter LSM.
{LinkMgmt.Adapter1.Rsp.PMNAK}		02h		PMNAK Response for Stack 1 Adapter LSM
{LinkMgmt.Adapter1.Rsp.L1}		04h		L1 Response for Stack 1 Adapter LSM.
{LinkMgmt.Adapter1.Rsp.L2}		08h		L2 Response for Stack 1 Adapter LSM.
{LinkMgmt.Adapter1.Rsp.LinkReset}		09h		LinkReset Response for Stack 1 Adapter LSM.
{LinkMgmt.Adapter1.Rsp.Disable}		0Ch		Disable Response for Stack 1 Adapter LSM.
{ParityFeature.Req}	07h	00h	Reserved	Parity Feature enable request.

Name	Msgcode	Msgsubcode	MsgInfo	Description
{ParityFeature.Ack}	08h	00h	0000h: Regular Response FFFFh: Stall Response	Parity Feature enable Ack.
{ParityFeature.Nak}		01h		Parity Feature enable Nak.
{ErrMsg}	09h	00h	Reserved	Correctable Error Message.
		01h		Non-Fatal Error Message.
		02h		Fatal Error Message.
--	FFh	--	Vendor ID	Vendor Defined Messages.
All other encodings not mentioned in this table are reserved.				

Table 54. Link Training State Machine related Message encodings

Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode[7:0]
{Start Tx initiated D2C point test resp}	0000h	8Ah	01h
{ LFSR_clear_error req}	0000h	85h	02h
{ LFSR_clear_error resp}	0000h	8Ah	02h
{Txinit Dto C results Req}	0000h	85h	03h
{End Tx initiated D2C point test req}	0000h	85h	04h
{End Tx initiated D2C point test resp}	0000h	8Ah	04h
{Start Tx init D to C eye sweep resp}	0000h	8Ah	05h
{ LFSR_clear_error req}	0000h	85h	02h
{ LFSR_clear_error resp}	0000h	8Ah	02h
{Txinit Dto C results Req}	0000h	85h	03h
{End Tx init D to C eye sweep req}	0000h	85h	06h
{End Tx init D to C eye sweep resp}	0000h	8Ah	06h
{Start Rx init D to C point test resp}	0000h	8Ah	07h
{ LFSR_clear_error req}	0000h	85h	02h
{ LFSR_clear_error resp}	0000h	8Ah	02h
{Tx Count Done req}	0000h	85h	08h
{Tx Count Done resp}	0000h	8Ah	08h
{End Rx init D to C point test resp}	0000h	8Ah	09h
{Start Rx init D to C eye sweep resp}	0000h	8Ah	0Ah
{ LFSR_clear_error req}	0000h	85h	02h
{ LFSR_clear_error resp}	0000h	8Ah	02h
{Rxinit D to C results Req}	0000h	85h	0Bh
{End Rx initi D to C eye sweep req}	0000h	85h	0Dh
{End Rx init D to C eye sweep resp}	0000h	8Ah	0Dh

Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode[7:0]
{SBINIT out of Reset}	{[15:4] : Reserved [3:0] : Result[3:0]}	91h	00h
{SBINIT done req}	0000h	95h	01h
{SBINIT done resp}	0000h	9Ah	01h
{MBINIT.CAL Done req}	0000h	A5h	02h
{MBINIT.CAL Done resp}	0000h	AAh	02h
{MBINIT.REPAIRCLK init req}	0000h	A5h	03h
{MBINIT.REPAIRCLK init resp}	0000h	AAh	03h
{MBINIT.REPAIRCLK result req}	0000h	A5h	04h
{MBINIT.REPAIRCLK apply repair resp}	0000h	AAh	05h
{MBINIT.REPAIRCLK check repair init req}	0000h	A5h	06h
{MBINIT.REPAIRCLK check repair init resp	0000h	AAh	06h
{MBINIT.REPAIRCLK check results req}	0000h	A5h	07h
{MBINIT.RepairCLK done req}	0000h	A5h	08h
{MBINIT.RepairCLK done resp}	0000h	AAh	08h
{MBINIT.REPAIRVAL init req}	0000h	A5h	09h
{MBINIT.REPAIRVAL init resp}	0000h	AAh	09h
{MBINIT.REPAIRVAL result req}	0000h	A5h	0Ah
{MBINIT.REPAIRVAL apply repair resp}	0000h	AAh	0Bh
{MBINIT.RepairVAL done req}	0000h	A5h	0Ch
{MBINIT.RepairVAL done resp}	0000h	AAh	0Ch
{MBINIT.REVERSALMB init req}	0000h	A5h	0Dh
{MBINIT.REVERSALMB init resp}	0000h	AAh	0Dh
{MBINIT.REVERSALMB clear error req}	0000h	A5h	0Eh
{MBINIT.REVERSALMB clear error resp}	0000h	AAh	0Eh
{MBINIT.REVERSALMB result req}	0000h	A5h	0Fh
{MBINIT.ReversalMB done req}	0000h	A5h	10h
{MBINIT.RversalMB done resp}	0000h	AAh	10h
{MBINIT.REPAIRMB start req}	0000h	A5h	11h
{MBINIT.REPAIRMB start resp}	0000h	AAh	11h
{MBINIT.REPAIRMB Apply repair resp}	0000h	AAh	12h
{MBINIT.REPAIRMB end req}	0000h	A5h	13h
{MBINIT.REPAIRMB end resp}	0000h	AAh	13h

Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode[7:0]
{MBINIT.REPAIRCLK result resp}	{[15:4] : Reserved [3] : Compare Results from RXCKRD [2] : Compare Results from RTRK_L [1] : Compare Results from RCKN_L [0] : Compare Results from RCKP_L}	AAh	04h
{MBINIT.REPAIRCLK apply repair req}	< [15:4] : Reserved, [3:0] : Repair Encoding Fh : No Repair 0h : Repair RCLKP_L 1h : Repair RCLKN_L 2h : Repair RTRK_L 7h : Unrepairable	A5h	05h
{MBINIT.REPAIRCLK check results resp	{[15:4] : Reserved [3] : Compare Results from RRDCK_L [2] : Compare Results from RTRK_L [1] : Compare Results from RCKN_L [0] : Compare Results from RCKP_L}	AAh	07h
{MBINIT.REPAIRVAL result resp}	{[15:2] : Reserved [1] : Compare Results from RRDVLD_L [0] : Compare Results from RVLD_L}	AAh	0Ah
{MBINIT.REPAIRVAL apply repair req}	{[15:2] : Reserved [1:0] : Repair Encoding 3h : No Repair 0h : Repair RVLD_L 1h : Unrepairable	A5h	0Bh
{MBINIT.REPAIRMB end req}	0000h	A5h	14h
{MBINIT.REPAIRMB end resp}	0h	AAh	14h
{MBTRAIN.VALVREF start req}	0000h	B5h	00h
{MBTRAIN.VALVREF start resp}	0000h	BAh	00h
{MBTRAIN.VALVREF end req}	0000h	B5h	01h

Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode[7:0]
{MBTRAIN.VALVREF end resp}	0000h	BAh	01h
{MBTRAIN.DATAVREF start req}	0000h	B5h	02h
{MBTRAIN.DATAVREF start resp}	0000h	BAh	02h
{MBTRAIN.DATAVREF end req}	0000h	B5h	03h
{MBTRAIN.DATAVREF end resp}	0000h	BAh	03h
{MBTRAIN.SPEEDIDLE done req}	0000h	B5h	04h
{MBTRAIN.SPEEDIDLE done resp}	0000h	BAh	04h
{MBTRAIN.TXSELFCAL Done req}	0000h	B5h	05h
{MBTRAIN.TXSELFCAL Done resp}	0000h	BAh	05h
{MBTRAIN.RXCLKCAL start req}	0000h	B5h	06h
{MBTRAIN.RXCLKCAL start resp}	0000h	BAh	06h
{MBTRAIN.RXCLKCAL done req}	0000h	B5h	07h
{MBTRAIN.RXCLKCAL done resp}	0000h	BAh	07h
{MBTAIN.VALTRAINCENTER start req}	0000h	B5h	08h
{MBTAIN.VALTRAINCENTER start resp}	0000h	BAh	08h
{MBTRAIN.VALTRAINCENTER done req}	0000h	BAh	09h
{MBTRAIN.VALTRAINCENTER done resp}	0000h	BAh	09h
{MBTAIN.VALTRAINVREF start req}	0000h	B5h	0Ah
{MBTAIN.VALTRAINVREF start resp}	0000h	BAh	0Ah
{MBTRAIN.VALTRAINVREF done req}	0000h	BAh	0Bh
{MBTRAIN.VALTRAINVREF done resp}	0000h	BAh	0Bh
{MBTRAIN.DATATRAINCENTER1 start req}	0000h	BAh	0Ch
{MBTRAIN.DATATRAINCENTER1 start resp	0000h	B5h	0Ch
{MBTRAIN.DATATRAINCENTER1 end req}	0000h	BAh	0Dh
{MBTRAIN.DATATRAINCENTER1 end resp}	0000h	B5h	0Dh
{MBTRAIN.DATATRAINVREF start req}	0000h	BAh	0Eh
{MBTRAIN.DATATRAINVREF start resp	0000h	B5h	0Fh
{MBTRAIN.DATATRAINVREF end req}	0000h	BAh	10h
{MBTRAIN.DATATRAINVREF end resp}	0000h	B5h	10h
{MBTRAIN.RXDESKEW start req}	0000h	BAh	11h
{MBTRAIN.RXDESKEW start resp	0000h	B5h	11h
{MBTRAIN.RXDESKEW end req}	0000h	BAh	12h
{MBTRAIN.RXDESKEW end resp}	0000h	B5h	12h
{MBTRAIN.DATATRAINCENTER2 start req}	0000h	BAh	13h
{MBTRAIN.DATATRAINCENTER2 start resp	0000h	B5h	13h
{MBTRAIN.DATATRAINCENTER2 end req}	0000h	BAh	14h
{MBTRAIN.DATATRAINCENTER2 end resp}	0000h	B5h	14h
{MBTRAIN.LINKSPEED start req}	0000h	BAh	15h

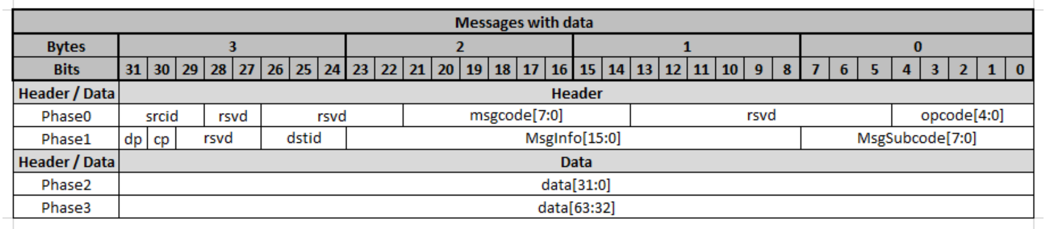
Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode[7:0]
{MBTRAIN.LINKSPEED start resp}	0000h	B5h	15h
{MBTRAIN.LINKSPEED error req}	0000h	BAh	16h
{MBTRAIN.LINKSPEED error resp}	0000h	B5h	16h
{MBTRAIN.LINKSPEED exit to repair req}	0000h	BAh	17h
{MBTRAIN.LINKSPEED exit to repair resp}	0000h	B5h	17h
{MBTRAIN.LINKSPEED exit to speed degrade req}	0000h	BAh	18h
{MBTRAIN.LINKSPEED exit to speed degrade resp}	0000h	B5h	18h
{MBTRAIN.LINKSPEED done req}	0000h: for regular response FFFFh: for stall	BAh	19h
{MBTRAIN.LINKSPEED done resp}	0000h: for regular response FFFFh: for stall	B5h	19h
{LINKSPEED.multimodule disable module req}	[15:3]: Rsvd [2:0]: Number of Active modules	BAh	1Ah
{LINKSPEED.multimodule disable module resp}	0000h	B5h	1Ah
{MBTRAIN.LINKSPEED exit to phy retrain req}	0000h	BAh	1Fh
{MBTRAIN.LINKSPEED exit to phy retrain resp}	0000h	B5h	1Fh
{MBTRAIN.REPAIR init req}	0000h	BAh	1Bh
{MBTRAIN.REPAIR init resp}	0000h	B5h	1Bh
{MBTRAIN.REPAIR Apply repair resp}	0000h	B5h	1Ch
{MBTRAIN.REPAIR end req}	0000h	BAh	1Dh
{MBTRAIN.REPAIR end resp}	0000h	B5h	1Dh
{MBTRAIN.REPAIR Apply degrade req}	[15:2]: Reserved [1:0] Standard package logical Lane map	BAh	1Eh
{MBTRAIN.REPAIR Apply degrade resp}	0000h	B5h	1Eh
{PHYRETRAIN.retrain init req}	[15:3] Reserved [2:0] : Retrain Encoding	C5h	00h
{PHYRETRAIN.retrain init resp}	[15:3] Reserved [2:0] : Retrain Encoding	CAh	00h
{PHYRETRAIN. retrain start req}	[15:3] Reserved [2:0] : Retrain Encoding	C5h	01h
{PHYRETRAIN.retrain start resp}	[15:3] Reserved [2:0] : Retrain Encoding	CAh	01h
{RECAL. track pattern init resp}	0000h	D5h	00h
{RECAL.track pattern done resp}	0000h	DAh	00h

Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode[7:0]
{TRAINERROR Entry req}	0000h	E5h	00h
{TRAINERROR Entry resp}	0000h	EAh	00h

6.1.2.3 Messages with data payloads

Figure 103 shows the formats for Messages with data payloads. The definitions of opcode, srcid, dstid, dp, cp, ak and sn fields are the same as Register Access packets.

Figure 103. Format for Messages with data payloads



Messages with data																																
Bytes	3								2								1								0							
Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Header / Data	Header																															
Phase0	srcid			rsvd		rsvd					msgcode[7:0]								rsvd									opcode[4:0]				
Phase1	dp	cp	rsvd			dstid			MsgInfo[15:0]																MsgSubcode[7:0]							
Header / Data	Data																															
Phase2	data[31:0]																															
Phase3	data[63:32]																															

Table 55 and Table give the message encodings.

Table 55. Message encodings

Name	Msgcode	Msgsubcode	MsgInfo	Data Bit Encodings	Description
{AdvCap.Adapter}	01h	00h	0000h:Reserved FFFFh: Stall	[0]: "Raw_Mode" [1]: "68B Flit Mode" [2]: "CXL 256B Flit Mode" [3]: "PCIe Flit Mode" [4]: "Streaming" [5]: "Retry" [6]: "Multi_Protocol_Enable" [7]: "Stack0_Enable" [8]: "Stack1_Enable" [9]: "CXL_LatOpt_Fmt5" [10]: "CXL_LatOpt_Fmt6" [11]: "Retimer" [20:12]: "Retimer Credits" [21]: "DP" [22]: "UP" [63:23]: Reserved	Advertised Capabilities of the D2D Adapter
{FinCap.Adapter}	02h	00h	0000h:Reserved FFFFh: Stall		Finalized Capability of the D2D Adapter

Name	Msgcode	Msgsubcode	MsgInfo	Data Bit Encodings	Description
{AdvCap.CXL }	01h	01h	0000h:Reserved FFFFh: Stall	[23:0] : Flexbus Mode negotiation usage bits as defined for Symbols 12-14 of Modified TS1/TS2 Ordered Set in the *Compute Express Link Specification*, with the following additional rules: [0]: PCIe capable/enable - this must be 1b for PCIe Non-Flit Mode. [1]: CXL.io capable/enable - this must be 0b for PCIe Non-Flit Mode. [2]: CXL.mem capable/enable - this must be 0b for PCIe Non-Flit Mode. [3]: CXL.cache capable/enable - this must be 0b for PCIe Non-Flit Mode. [4]: CXL 2.0 capable must be set for ports that support CXL protocols, as specified in the Protocol Layer interoperability requirements. [8]: Multi-Logical Device - must be set to 0b for PCIe Non-Flit Mode. [9]: CXL 68B-Enhanced Flit Mode - must be set to 0b for PCIe Non-Flit Mode. [12:10]: these bits do not apply for UCIe, must be 0b. [14]: Retimer 2 - does not apply for UCIe, must be 0b. [15]: CXL.io Throttle - must be 0b for PCIe Non-Flit Mode. [17:16]: NOP Hint Info - does not apply for UCIe, and must be 0.	Advertised Capabilities for CXL protocol.
{FinCap.CXL }	02h	01h	0000h:Reserved FFFFh: Stall		Finalized Capabilities for CXL protocol.
--	FFh	--	Vendor ID		Vendor Defined Messages.
All other encodings not mentioned in this table are reserved.					

Table 56. Link Training State Machine related encodings

Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode[7:0]	Data Field[63:0]
{Start Tx initiated D2C point test req}	[15:0]: Max Error threshold	85h	01h	{[63:59]: reserved, [58:43]: Iteration Count Settings, [42:27]: Idle Count settings, [26:11]: Burst Count settings, [10]: Pattern Mode (0: continuous mode, 1h : Burst Mode) [9:6] : Clock Phase control at Tx Device, [5:3] : Valid Pattern (0h: Functional pattern), [2:0]: Data pattern (0h: LFSR, 1h: Per Lane ID) }

{Txinit D to C results log}	{[15:6] : Reserved,
	[5]: Valid Lane comparison results
	[4]: Cumulative Results of all Lanes.
	[3:0]: Compare results from Redundant Lanes (RRD_L[3], RRD_L[2], RRD_L[1], RRD_L[0])}
		8Ah	03h	[63:0] : Compare Results of all Data Lanes
				UCIe-A { RD_L[63], RD_L[62], …., RD_L[1], RD_L[0]}
				UCIe-S { 48'h0, RD_L[15], RD_L[14], …, RD_L[1], RD_L[0]}

{Start Tx init D to C eye sweep req}	[15:0]: Max Error threshold	85h	05h	{[63:59] : reserved, [58:43] : Iteration Count Settings, [42:27] : Idle Count settings, [26:11] : Burst Count settings, [10] : Pattern Mode (0 : continuous mode, 1h : Burst Mode) [9:6] : Clock Phase control at Tx Device, [5:3] : Valid Pattern (0h : Functional pattern), [2:0] : Data pattern (0h : LFSR, 1h : Per Lane ID) }

{Txinit D to C results log}	{[15:6] : Reserved,
	[5] : Valid Lane comparison results
	[4] : Cumulative Results of all Lanes.
	[3:0]: Compare results from Redundant Lanes (RRD_L[3], RRD_L[2], RRD_L[1], RRD_L[0])}
		8Ah	03h	[63:0] : Compare Results of all Data Lanes
				UCIe-A { RD_L[63], RD_L[62], …., RD_L[1], RD_L[0]}
				UCIe-S { 48'h0, RD_L[15], RD_L[14], …, RD_L[1], RD_L[0]}

Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode [7:0]	Data Field[63:0]
{Start Rx init D to C point test req}	[15:0]: Max Error threshold	85h	07h	{[63:59] : reserved, [58:43] : Iteration Count Settings, [42:27] : Idle Count settings, [26:11] : Burst Count settings, [10] : Pattern Mode (0 : continuous mode, 1h : Burst Mode) [9:6] : Clock Phase control at Transmitter, [5:3] : Valid Pattern (0h : Functional pattern), [2:0] : Data pattern (0h : LFSR, 1h : Per Lane ID) }
{Start Rx init D to C eye sweep req}	[15:0]: Max Error threshold	85h	0Ah	{[63:59]: reserved, [58:43] : Iteration Count Settings, [42:27] : Idle Count settings, [26:11] : Burst Count settings, [10] : Pattern Mode (0 : continuous mode, 1h : Burst Mode) [9:6] : Clock Phase control at Transmitter, [5:3] : Valid Pattern (0h : Functional pattern), [2:0] : Data pattern (0h : LFSR, 1h : Per Lane ID) }

{Rxinit D to C results resp}	{[15:6]: Reserved,
	[5]: Valid Lane comparison results
	[4]: Cumulative Results of all Lanes.
	[3:0]: Compare results from Redundant Lanes (RRD_L[3], RRD_L[2], RRD_L[1], RRD_L[0])}
		8Ah	0Bh	[63:0] : Compare Results of all Data Lanes
				UCIe-A { RD_L[63], RD_L[62], …., RD_L[1], RD_L[0]}
				UCIe-S { 48'h0, RD_L[15], RD_L[14], …, RD_L[1], RD_L[0]}

{Rx Init D2C sweep done with results}	0000h	81h	0Ch	{[63:16]: reserved, [15:8]: Right Edge, [7:0]: Left Edge}
{MBINIT.PARAM Configuration req}	0000h	A5h	00h	{ [63:13]: Reserved [12:11]: Module ID: 0h: 0, 1h: 1, 2h: 2, 3h:3 [10]: Clock Phase: 0b: Differential clock, 1b: Quadrature phase [9]: Clock Mode - 0b: Strobe mode; 1b: Continuous mode [8:4]: Voltage Swing- The encodings are the same as the "Supported Vswing encodings" field of the PHY Capability register. [3:0]: Max IO Link Speed - The encodings are the same as "Max Link Speed" field of the UCIe Link Capability register}

Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode[7:0]	Data Field[63:0]
{MBINIT.PARAM configuration resp}	0000h	AAh	00h	{[63:11]: Reserved [10]: Clock Phase: 0b: Differential clock, 1b: Quadrature phase [9]: Clock Mode - 0b: Strobe mode; 1b: Continuous mode [8:4]: Reserved [3:0]: Max IO Link Speed - The encodings are the same as "Max Link Speed" field of the UCIe Link Capability register}

{MBINIT.REVERSALMB result resp}	{[15:6]: Reserved,
	[5]: Valid Lane comparison results
	[4]: Cumulative Results of all Lanes.
	[3:0]: Compare results from Redundant Lanes (RRD_L[3], RRD_L[2], RRD_L[1], RRD_L[0])}
		AAh	0Fh	[63:0] : Compare Results of all Data Lanes
				UCIe-A { RD_L[63], RD_L[62], …., RD_L[1], RD_L[0]}
				UCIe-S { 48'h0, RD_L[15], RD_L[14], …, RD_L[1], RD_L[0]}

Message	**MsgInfo[15:0]**	**MsgCode[7:0]**	**MsgSubcode [7:0]**	**Data Field[63:0]**
{MBINIT.REPAIRMB Apply repair req}	0000h	A5h	12h	{[31:24] : **Repair Address for TRD_P[3]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 20h: Invalid 21h: TD_P[33] Repaired 22h: TD_P[34] Repaired …... 3Eh: TD_P[62] Repaired 3Fh: TD_P[63] Repaired F0h: Unrepairable [23:16]: **Repair Address for TRD_P[2]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 20h: TD_P[32] Repaired 21h: TD_P[33] Repaired 22h: TD_P[34] Repaired …... 3Eh: TD_P[62] Repaired 3Fh: TD_P[63] Repaired F0h: Unrepairable [15:8]: **Repair Address for TRD_P[1]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 00h: Invalid 01h: TD_P[1] Repaired 02h: TD_P[2] Repaired …... 1Eh: TD_P[30] Repaired 1Fh: TD_P[31] Repaired F0h: Unrepairable [7:0]: **Repair Address for TRD_P[2]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 20h: TD_P[32] Repaired 21h: TD_P[33] Repaired 22h: TD_P[34] Repaired …... 3Eh: TD_P[62] Repaired 3Fh: TD_P[63] Repaired F0h: Unrepairable

Message	MsgInfo[15:0]	MsgCode[7:0]	MsgSubcode[7:0]	Data Field[63:0]
{MBTRAIN.REPAIR Apply repair req}	0000h	BAh	1Ch	{[31:24] : **Repair Address for TRD_P[3]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 20h: Invalid 21h: TD_P[33] Repaired 22h: TD_P[34] Repaired …... 3Eh: TD_P[62] Repaired 3Fh: TD_P[63] Repaired F0h: Unrepairable [23:16]: **Repair Address for TRD_P[2]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 20h: TD_P[32] Repaired 21h: TD_P[33] Repaired 22h: TD_P[34] Repaired …... 3Eh: TD_P[62] Repaired 3Fh: TD_P[63] Repaired F0h: Unrepairable [15:8]: **Repair Address for TRD_P[1]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 00h: Invalid 01h: TD_P[1] Repaired 02h: TD_P[2] Repaired …... 1Eh: TD_P[30] Repaired 1Fh: TD_P[31] Repaired F0h: Unrepairable [7:0]: **Repair Address for TRD_P[2]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 20h: TD_P[32] Repaired 21h: TD_P[33] Repaired 22h: TD_P[34] Repaired …... 3Eh: TD_P[62] Repaired 3Fh: TD_P[63] Repaired F0h: Unrepairable

6.1.3 Flow Control and Data Integrity

Sideband packets can be transferred across FDI, RDI or the UCIe sideband Link. Each of these have independent flow control.

6.1.3.1 Flow Control and Data Integrity over FDI and RDI

For each Transmitter associated with FDI or RDI, a design time parameter of the interface is used to determine the number of credits advertised by the Receiver, with a maximum of 32 credits. Each credit corresponds to 64 bits of header and 64 bits of potentially associated data. Thus, there is only one type of credit for all sideband packets, regardless of how much data they carry. Every Transmitter/Receiver pair has an independent credit loop. For example, on RDI, credits are advertised from Physical Layer to Adapter for sideband packets transmitted from the Adapter to the Physical Layer; and credits are also advertised from Adapter to the Physical Layer for sideband packets transmitted from the Physical Layer to the Adapter.

The Transmitter must check for available credits before sending Register Access requests and Messages. The Transmitter must not check for credits before sending Register Access Completions, and the Receiver must guarantee unconditional sinking for any Register Access Completion packets. Messages carrying requests or responses consume a credit on FDI and RDI, but they must be guaranteed to make forward progress by the Receiver and not get blocked behind Register Access requests.Both RDI and FDI give a dedicated signal for sideband credit returns across those interfaces.

All Receivers associated with RDI and FDI must check received messages for data or control parity errors, and these errors must be mapped to Uncorrectable Internal Errors (UIE) and transition RDI to LinkError state.

6.1.3.2 Flow Control and Data Integrity over UCIe sideband Link between dies

The BER of the sideband Link is 1e-27 or better. Hence, no retry mechanism is provided for the sideband packets. Receivers of sideband packets must check for Data or Control parity errors, and any of these errors is mapped to a fatal UIE.

6.1.3.3 End-to-End flow control and forward progress for UCIe Link sideband

It is important for deadlock avoidance to ensure that there is sufficient space at the Receiver to sink all possible outstanding requests from the Transmitter, so that the requests do not get blocked at any intermediate buffers preventing subsequent completions from making progress.

Sideband access for Remote Link partner's Adapter or Physical Layer registers is only accessible via the indirect mailbox mechanism, and the number of outstanding transactions is limited to four at a time. This is separate from local FDI or RDI accesses, and so Physical Layer and Adapter must provision for sinking at least four register access requests each from remote Adapter. Each credit corresponds to 64b of header and 64b of data. Even requests that send no data or only send 32b of data consume one credit. Register Access completions do not consume a credit and must always sink. The Adapter credit counters for register access request transmission are initialized to 4 whenever RDI is in Reset state. It is permitted to send an extra (N-4) credit returns to remote Link partner if a UCIe implementation is capable of sinking a total of N requests once RDI has transitioned to Active state. The Adapter must implement a saturating credit counter capable of accumulating at least 4 credits, and hence prevent excess credit returns from overflowing the counter.

All other messages except Vendor Defined messages must always sink and make forward progress, and not block any messages on the sideband interface behind them. All Link Management message requests have an associated response, and the source of these messages must only have one outstanding request at a time (i.e., one outstanding message per "Link Management Request" MsgCode encoding).

For vendor defined messages, there must be a vendor defined cap on the number of outstanding messages, and the Receiver must guarantee sufficient space so as to not block any messages behind the vendor defined messages on any of the interfaces.

6.1.4 Operation on RDI and FDI

The same formats and rules of operation are followed on both the RDI and FDI interfaces. The protocol is symmetric, both requests, completions and messages can be sent on **`lp_cfg`** as well as **`pl_cfg`** signals. Implementations must ensure deadlock free operation by allowing sufficient sideband packets to sink and unblock the sideband bus for other packets. At the interface, these transactions are packetized into multiple phases depending on the configuration interface width (compile time parameter). Supported interface widths are 8, 16 or 32 bits. **`lp_cfg_vld`** and **`pl_cfg_vld`** are asserted independently for each phase. They must be asserted on consecutive clock cycles for transferring consecutive phases of the same packet. They may or may not assert on consecutive clock cycles when transferring phases of different packets.

7.0 Configuration and Parameters

7.1 High level Software view of UCIe

A key goal of UCIe is to leverage all the software investments made for PCIe and CXL while still defining the interface in an extensible way for future innovative solutions. To that end, UCIe's SW view of the protocol layer is consistent with the associated protocol. For example, the host Downstream Port for UCIe that is capable of supporting CXL protocols will appear to software as a Root Port with CXL DVSEC capability and relevant PCIe capabilities. Similarly, a host downstream port for UCIe that is capable of supporting PCIe protocol only, will appear to software as a Root Port with relevant PCIe capabilities only. Host side or device side view of software for Streaming protocol is implementation-specific since the protocol itself is implementation-specific. It is though strongly recommended that ecosystem implementations define streaming solutions leveraging the SW hooks already in place for supporting CXL and PCIe. The Upstream Ports that connect to a UCIe Root Port can be a PCI Express end point, PCI Express Switch, a CXL 2.0 or above compliant endpoint-device, or a CXL Switch. This allows for UCIe solution to be fully backward compatible to pre-UCIe software. Rest of this chapter talks about SW view of UCIe when paired with PCIe or CXL protocol layers.

UCIe specification allows for a single UCIe Link layer (aka. D2D)/PHY to be shared by multiple protocol stacks. In the first generation of the spec, this sharing is limited to at most 2 identical protocol stacks. Shared Link layer is a new concept from Software perspective and requires new discovery/control mechanisms. The mechanism by which UCIe-aware SW discovers UCIe capability is described in the next section.

Table 57 shows the legal/illegal combinations of Upstream and Downstream devices/ports at a given UCIe interface, from a SW viewpoint.

Table 57. Software view of Upstream and Downstream Device at UCIe interface

Downstream Component: SW View	Upstream Component: SW View			
	PCIe RP, PCIe Switch DSP [1]	CXL-RP, CXL Switch DSP [2]	CXL Downstream Port RCRB [3]	Streaming Device
PCIe EP, PCIe Switch USP	Valid	Valid	Illegal	Vendor defined
CXL Upstream Port RCRB [4]	Illegal	Illegal	Illegal	
CXL EP	Valid	Valid	Illegal	
Streaming Device	Vendor defined			

[1] PCIe RP = As defined in PCIe Base Specification

[2] CXL RP/Switch DSP = Standard PCIe RP/Switch-DSP with additional CXL Flexbus Port DVSEC capability

[3] CXL Downstream Port RCRB = CXL Link at host or at Switch DSP that is enumerated via CXL defined Downstream Port RCRB (instead of via a Root Port)

[4] CXL Upstream Port RCRB = CXL upstream port that is enumerated via CXL defined RCRB with CXL Upstream Port RCRB and that has a RCiEP below it.

All the CXL/PCIe legacy/advanced capabilities/registers defined in the respective specifications apply to UCIe host and devices as well. Some Link and PHY layer specific registers in PCIe specification do not in UCIe context and these are listed in the appendix. In addition, two new DVSEC capabilities and three other MMIO mapped register blocks are defined to deal with UCIe-specific Adapter and Physical Layer capabilities.

7.2 SW Discovery of UCIe Links

UCIe-aware Firmware/Software may discover the presence and capabilities of UCIe Links in the system per Table 58.

Table 58. SW discovery of UCIe Links

UCIe Links	How discovered?	Salient Points
In host	Host specific Register Block called CiRB, containing UCIe Link DVSEC Capability	• CiRB is at a host defined static location. • Each UCIe Link has a separate CiRB Base address and these are enumerated to OS via UCIe Early discovery table (CIDT)[1] • Association of a UCIe Link to 1 or more Root ports is described in CIDT, allowing for UCIe-aware SW to understand the potential shared nature of the UCIe Link.
In End Points	Dev0/Fn0 of the device carries a UCIe Link DVSEC Capability.	• In multi-stack implementations, Dev0/Fn0 of the endpoint in only one of the stacks carries the UCIe Link DVSEC Capability.

Table 58. SW discovery of UCIe Links

UCIe Links	How discovered?	Salient Points
In host	Host specific Register Block called CiRB, containing UCIe Link DVSEC Capability	• CiRB is at a host defined static location. • Each UCIe Link has a separate CiRB Base address and these are enumerated to OS via UCIe Early discovery table (CIDT)[1] • Association of a UCIe Link to 1 or more Root ports is described in CIDT, allowing for UCIe-aware SW to understand the potential shared nature of the UCIe Link.
In Switch USP	Dev0/Fn0 of the USP carrying a UCIe Link DVSEC Capability	• In multi-stack implementations, Dev0/Fn0 of the USP in only one of the stacks carries the UCIe Link DVSEC Capability.
In Switch DSP	Dev0/Fn0 of the Switch USP carrying one ore more CiSRB DVSEC Capability	• UCIe Links below the switch are described in CiSRB whose base address is provided in the CiSRB DVSEC Capability • A UCIe Link DVSEC capability per downstream UCIe Link is present in the CiSRB • Association of a UCIe Link to 1 or more Switch DSPs is described as part of the UCIe Link DVSEC Capability, allowing for UCIe-aware SW to understand the potential shared nature of the UCIe interface Note: Its legal for a Switch USP to carry the CiSRB DVSEC capability but not a UCIe Link DVSEC Capability

[1] CIDT structure will be standardized as part of the ACPI specifications moving forward

7.3 Register Location Details and Access Mechanism

- 2 UCIe DVSEC capabilities (UCIe Link DVSEC, CiSRB DVSEC) and three other MMIO-mapped register blocks are defined in the first version of the specification.
- UCIe Link DVSEC capability is located in CiRB for host root ports and in CiSRB for Switch downstream ports.
- CiRB region is defined at a static location on the host side and its size is enumerated in the CIDT structure. Only UCIe Link related registers are permitted in this region and designs must not implement non-UCIe related functionality in this region.
- There is a unique CiRB base address for each UCIe Link, in the host

- CiSRB region base address is provided in the CiSRB DVSEC capability. This region is part of a BAR region of Switch Dev0/Fn0 USP.
- For scalability/flexibility reasons, multiple CiSRB DVSEC capabilities can exist in a Switch USP function. In case of multiple CiSRB DVSEC capabilities in the USP. a given DSP UCIe Link can only be described in one of the CiSRB structures.
- Configuration space registers are accessed using configuration reads and configuration writes. Register Blocks are in memory mapped regions and are accessed using standard memory reads and memory writes.
- UCIe Retimer registers are not directly accessible from host SW. They can be accessed only via a window mechanism over the sideband interface (hence the terms *SB-MMIO* and *SB-Config* in Table 59). The window mechanism is available via RP/DSP UCIe Link DVSEC Capability to access the UCIe Retimer registers on the Retimer closest to the host. For accessing UCIe Retimer registers on the far end Retimer, the same window mechanism is also available in the UCIe Link DVSEC capability of EP/USP. See section 7.5.1.10 for details of the window mechanism.
- For debug and run-time Link health monitoring reasons, host SW can also access the UCIe related registers in any partner die on the sideband interface, using the same window mechanism. For brevity purposes, that is not shown below in Table 59. Note that register accesses over side band are limited to only the UCIe related Capability registers (the two DVSECs currently defined in the spec) and the three defined UCIe Register Blocks. Nothing else on the remote die are accessible via the side band mechanism.

Table 59 summarizes the location of various register blocks in each native UCIe port/ device. Henceforth a "UCIe port/device/EP/Switch" is used to refer to a standard PCIe or CXL port/device/EP/Switch with UCIe Link DVSEC Capability.

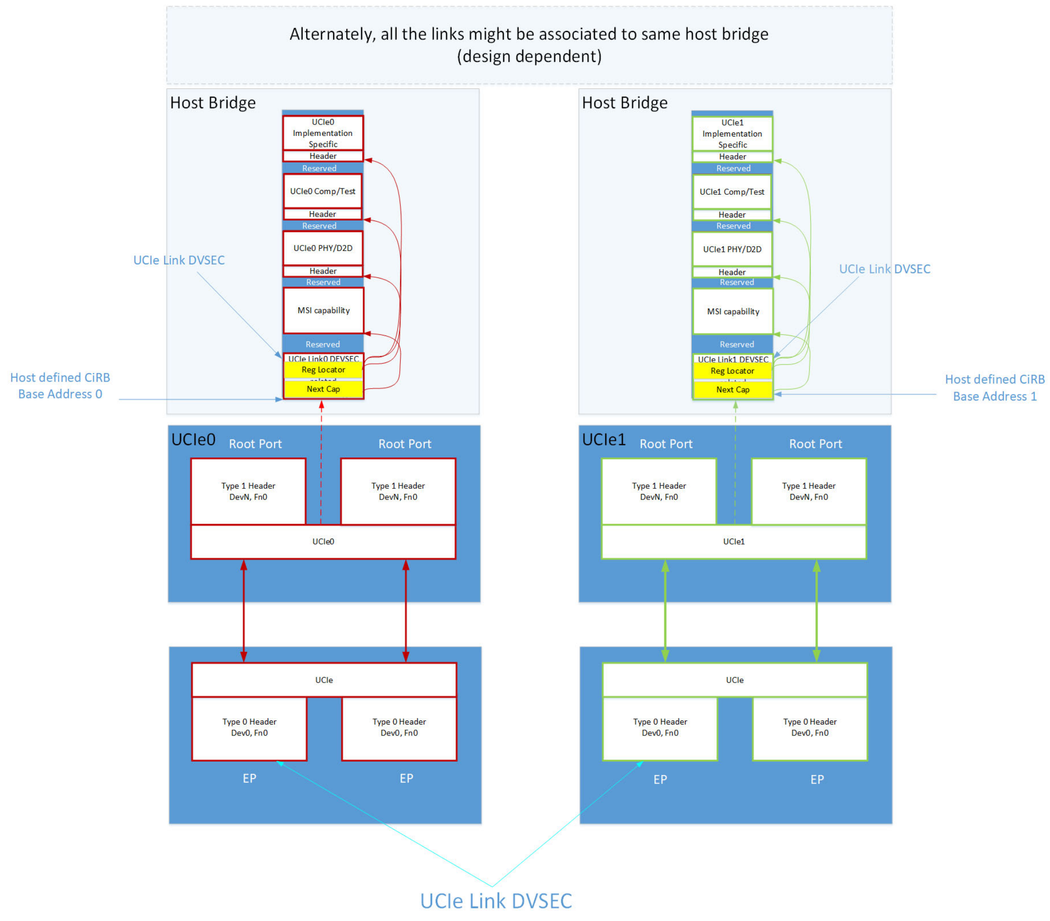
Table 59. Summary of location of various UCIe Link related registers

	Where does the register reside in ..?					**Comments**
	RP	**Switch USP**	**Switch DSP**	**EP**	**UCIe Retimer**	
UCIe Link DVSEC	CiRB	Config space	CiSRB	Config Space	Sideband Config Space	Registers that define the basic UCIe interface related details
UCIe D2D/PHY Register Block	CiRB	Switch USP-BAR Region	CiSRB	EP-BAR Region	SB-MMIO Space	Registers that define lower-level functionality for the D2D/PHY interface of a typical UCIe implementation
UCIe Test/ Compliance Register Block	CiRB	Switch USP-BAR Region	CiSRB	EP-BAR Region	SB-MMIO Space	Registers for Test/ Compliance of UCIe interface
UCIe Implementation Specific Register Block	CiRB	Switch USP-BAR Region	CiSRB	EP-BAR Region	SB-MMIO Space	Registers for vendor specific implementation

7.4 Software view Examples

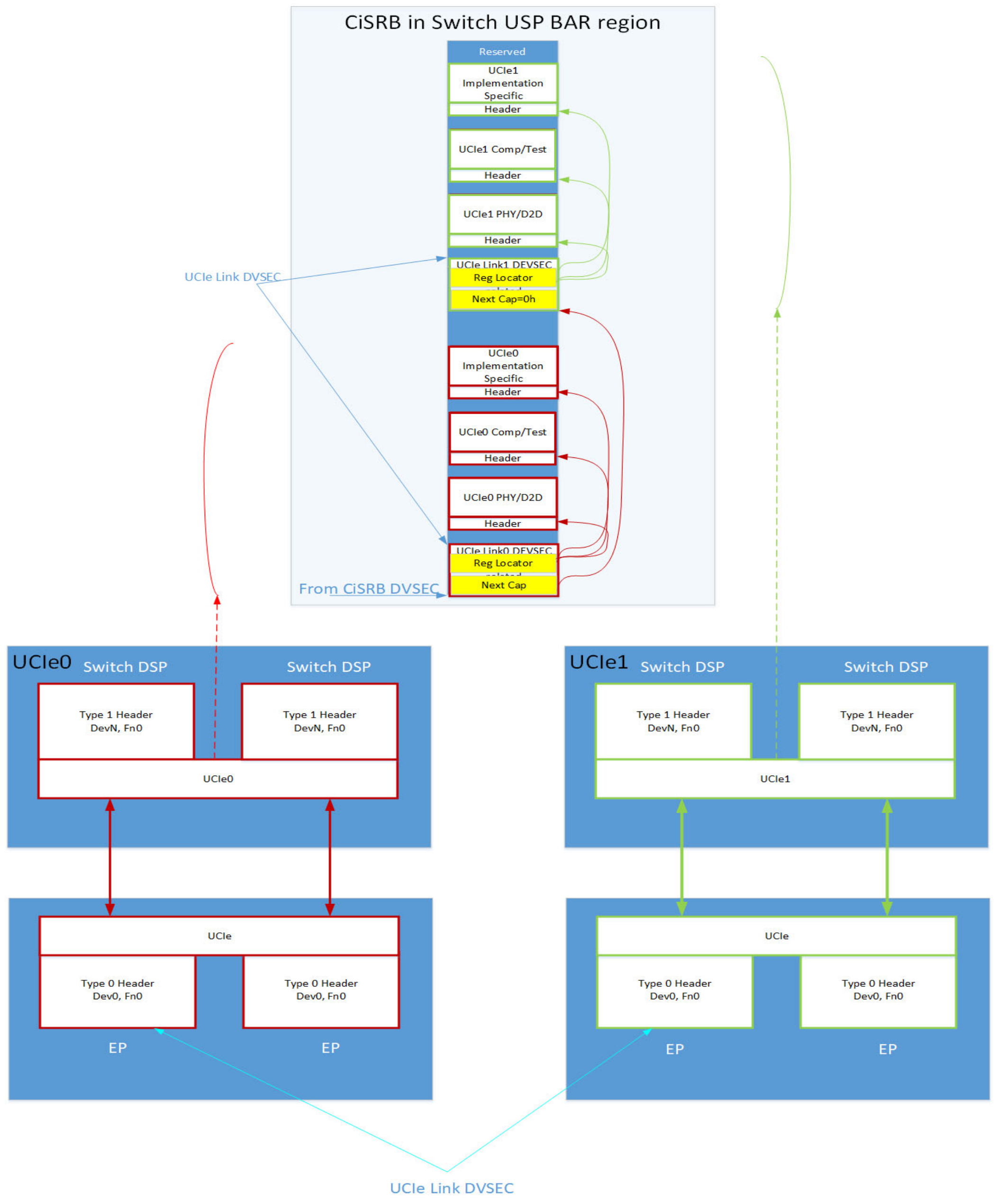
Figure 104 summarizes all the details of UCIe related DVSEC Capabilities and SW discovery of the same, for an implementation consisting of Root Ports and Endpoints. This example has a host with 2 UCIe downstream Links that each carry traffic from 2 Root Ports.

Figure 104. Software view Example with Root Ports and Endpoints



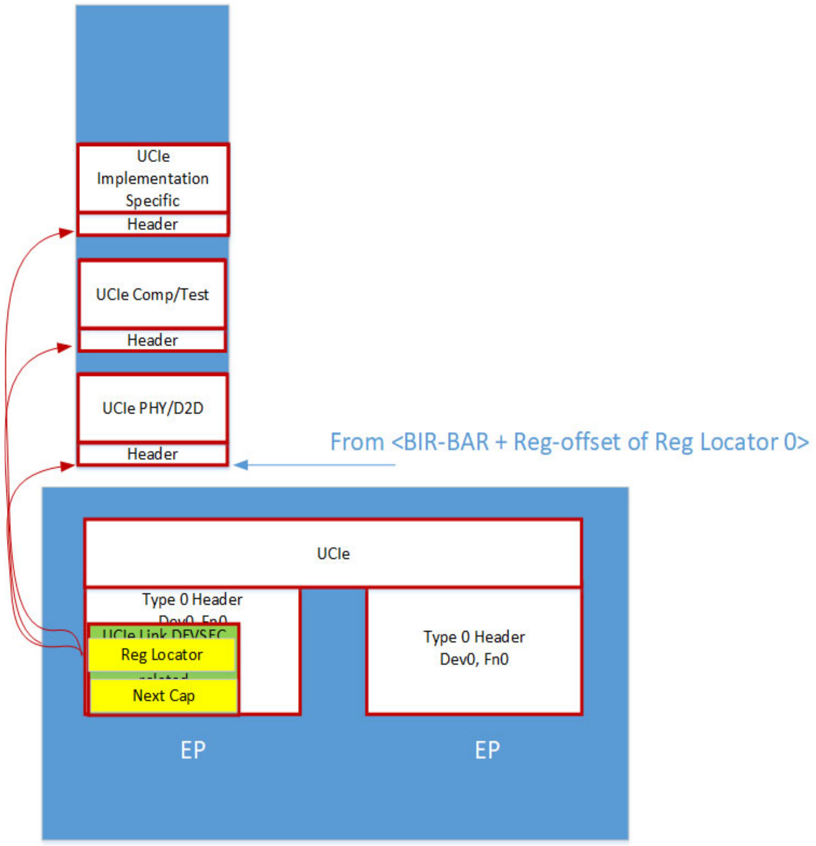
Example in Figure 105 has a Switch with 2 UCIe Links on its downstream side and each UCIe Link carries traffic from 2 Switch DSPs.

Figure 105. Software view Example with Switch and Endpoints



Example in Figure 106 shows details UCIe registers in an implementation where two EPs are sharing a common UCIe Link.

Figure 106. Software view Example of UCIe Endpoint



7.5 UCIe Registers

Figure 60 below summarizes the attributes for the register bits defined in this chapter. Unless otherwise specified, the definition of these attributes is consistent with the PCIe Base Specification and the CXL Specification.

Table 60. Register Attributes

Attribute	Description
RO	Read Only.
RW	Read-Write
RW1CS	Read-Write-One-To-Clear-Sticky. Not affected by hot reset of the Link. Otherwise, the behavior follows PCIe Base Specification.
HwInit	Hardware Initialized
RsvdP	Reserved and Preserved
RsvdZ	Reserved and Zero

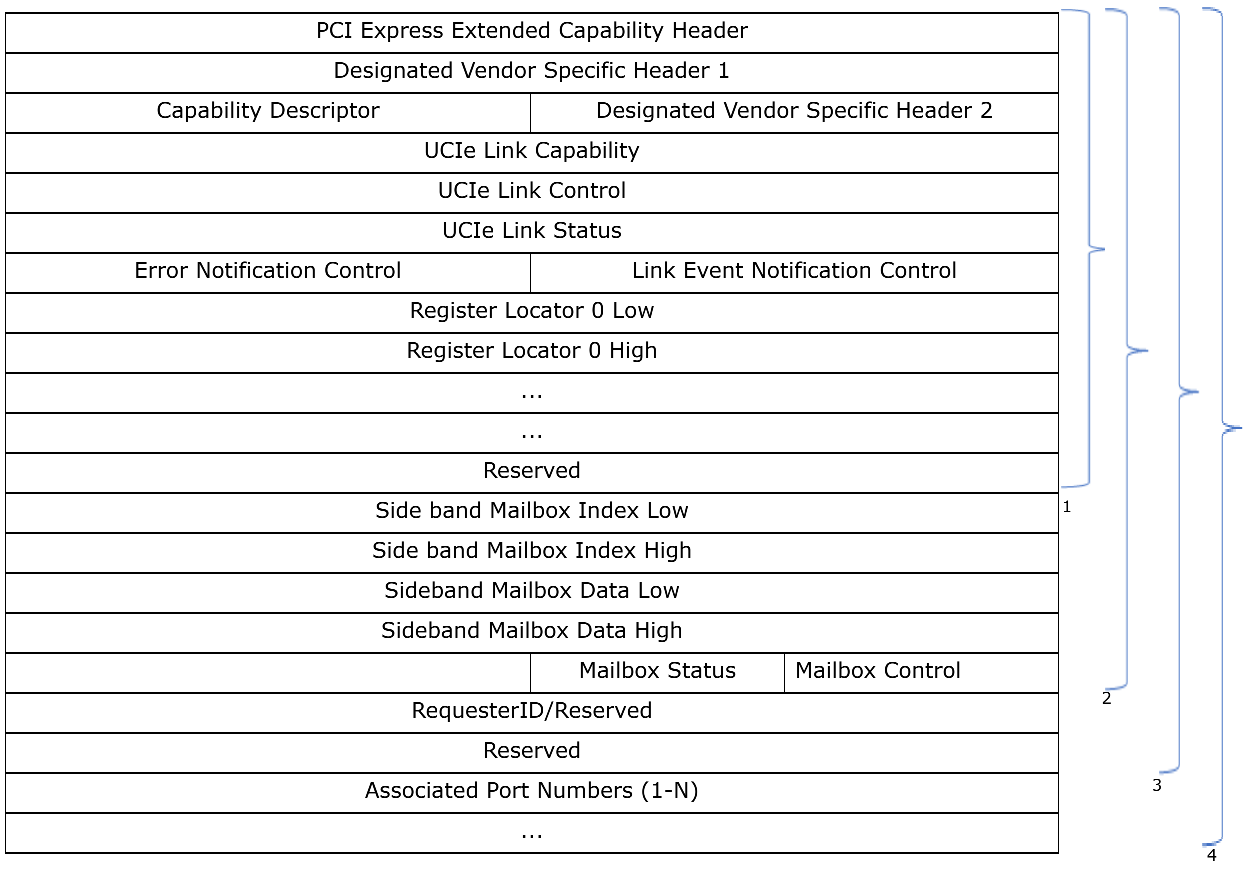
All numeric values in various data structures, individual registers and register fields defined in this chapter are always encoded in little endian format, unless stated otherwise.

7.5.1 UCIe Link DVSEC

The high-level Structure of the DVSEC is as follows

This is the basic capability register set that is required to operate a UCIe Link. And this is the one of two DVSEC capabilities defined for UCIe in the first generation. Not all the registers in the capability are applicable to all device/port types. The applicable registers for each device/port type are indicated in the right-hand side of Figure 107. Software may use the presence of this DVSEC to differentiate between a UCIe device vs. a standard PCIe or CXL device. Software may use the this DVSEC to differentiate between a UCIe Root Port and a standard PCIe or CXL Root Port.

Figure 107. UCIe Link DVSEC


PCI Express Extended Capability Header
Designated Vendor Specific Header 1
Capability Descriptor
Designated Vendor Specific Header 2
UCIe Link Capability
UCIe Link Control
UCIe Link Status
Error Notification Control
Link Event Notification Control
Register Locator 0 Low
Register Locator 0 High
...
...
Reserved
Side band Mailbox Index Low
Side band Mailbox Index High
Sideband Mailbox Data Low
Sideband Mailbox Data High
Mailbox Status
Mailbox Control
RequesterID/Reserved
Reserved
Associated Port Numbers (1-N)
...
1
2
3
4

1 applies to UCIe-EP, UCIe-USP, UCIe-Retimer

2 applies to UCIe-EP, UCIe-USP when paired with a retimer

3 applies to UCIe-RP

4 applies to UCIe-DSP

7.5.1.1 PCI Express Extended Capability Header (Offset 0x0)

Set as follows for UCIe Link DVSEC. All bits in this register are RO.

Table 61. CXL Link DVSEC - PCI Express Extended Capability Header

Field	Bit Location	Value	Comments
Capability ID	15:0	0023h	Value for PCI Express DVSEC capability
Revision ID	19:16	1h	Latest revision of the DVSEC capability
Next Capability Offset	31:20	Design Dependent	**For UCIe Link DVSEC in CiRB**: Set to point to MSI capability associated with this UCIe Link. The offset is in granularity of Bytes from the base address of CiRB. For example, if this is set to 0x100, the MSI capability is located at offset 0x100 from the base of CiRB. MSI capability must set its "Next Capability offset" to 0h to indicate end of capability chain for the specific UCIe Link. **UCIe Link DVSEC in CiSRB**: Set to point to the UCIe Link DVSEC capability of the next UCIe Link associated with a downstream port of the switch. The last UCIe Link DVSEC capability will set this offset to 0x0 indicating there are no more UCIe Links on downstream ports. The offset is in granularity of Bytes from the base address of CiSRB. For example, if this is set to 0x100, the next DVSEC capability for the next Link is located at offset of 0x100 from the base of CiSRB. **Retimer:** Set to 0x0 **Others**: design dependent

7.5.1.2 Designated Vendor Specific Header 1 (Offset 0x4), 2 (Offset 0x8)

A few things to note on the various fields described in Table 62. DVSEC Revision ID field represents the version of the DVSEC structure. The DVSEC Revision ID is incremented whenever the structure is extended to add more functionality. Backward compatibility shall be maintained during this process. For all values of n, DVSEC Revision ID n+1 structure may extend Revision ID n by replacing fields that are marked as reserved in Revision ID n, but must not redefine the meaning of existing fields. Software that was written for a lower Revision ID may continue to operate on UCIe DVSEC structures with a higher Revision ID, but will not be able to take advantage of new functionality.

All bits in this register are RO.

Table 62. UCIe Link DVSEC - Designated Vendor Specific Header 1, 2

Register	Field	Bit Location	Value
Designated Vendor-Specific Header 1 (offset 04h)	DVSEC Vendor ID	15:0	TBD
	DVSEC Revision	19:16	0h
	Length	31:20	Device dependent. See section 7.5.1.16 for some examples.
Designated Vendor-Specific Header 2 (offset 08h)	DVSEC ID	15:0	0h

7.5.1.3 Capability Descriptor (Offset 0xA)

Provides a way for SW to discover which optional capabilities are implemented by the UCIe Port/Device.

Table 63. UCIe Link DVSEC - Capability Descriptor

Bit	Attribute	Description
2:0	RO	**Number of Register locators:** 0h: 2 Register Locators 1h: 3 Register Locators .. 6h: 8 Register locators 7h: Reserved For first revision of UCIe, only values 0h and 1h are valid.
3	RO(RP/DSP), HWInit(EP/USP), RsvdP(Retimer)	**Side band mailbox Registers Present:** 0h: No side band mailbox register set present in this capability 1h: Side band mailbox register set present in this capability For RP/DSP, default value of this is 1. EP/USP must set this bit when they are paired with a retimer and must clear this bit in all other scenarios.

Table 63. UCIe Link DVSEC - Capability Descriptor

Bit	Attribute	Description
7:4	RO(DSP), RsvdP (Others)	**Number of Switch DSPs associated with this UCIe Link** Applies only to UCIe Link DVSEC in CiSRB. The specific 'port number' values of each Switch downstream port associated with this UCIe Link is called out in the Associated Port Number register(s) in this capability. 0x0 – 1 Port 0x1 – 2 ports .. 0xF – 16 ports 'Port Number' is bits 31:24 of the PCIe Link capabilities register of the downstream port. For first generation of UCIe, only values 0x0 and 0x1 are legal.
15:8	RsvdP	Reserved

7.5.1.4 UCIe Link DVSEC - UCIe Link Capability (Offset 0xC)

Basic characteristics of the UCIe Link are discovered by SW using this register.

Table 64. UCIe Link DVSEC - UCIe Link Capability

Bit	Attribute	Description
0	RO	**Raw_Mode:** If set, indicates the Link can support Raw Mode.
3:1	HWInit	**Max Link Width** 0h: x16 1h: x32 2h: x64 3h: x128 4h: x256 Others - Reserved
7:4	HWinit	**Max Link Speeds** 0h: 4GT/s 1h: 8GT/s 2h: 12GT/s 3h: 16GT/s 4h: 24GT/s 5h: 32GT/s Others: Reserved
8	RO(Retimer), RsvdP(others)	**Retimer** - Set by retimer to indicate it to SW

Table 64. UCIe Link DVSEC - UCIe Link Capability

Bit	Attribute	Description
9	RsvdP(Retimer), RO(others)	**Multi-stack capable** 0 - single stack capable 1 - multi stack capable In first rev of spec, only 2 stacks max is possible
10	RO	**Advanced Packaging** 0=Standard package mode for UCIe Link 1=Advanced package mode for UCIe Link
31:11	RsvdP	Reserved

7.5.1.5 UCIe Link DVSEC - UCIe Link Control (Offset 0x10)

Basic UCIe Link control bits are in this register.

Table 65. UCIe Link DVSEC - UCIe Link Control

Bit	Attribute	Description
0	RW(RP/DSP), HWInit(Others)	**Raw_Mode_Enable**: If set, enables the Link to negotiate Raw mode during Link training. Default value of this is 0b for RP and firmware/SW sets this bit based on system usage scenario. Switch DSP can set the default via implementation specific mechanisms like straps/FW/etc, to account of system usage scenario (like UCIe retimer). This allows for the DSP Link to train up without Software intervention and be UCIe-unaware-OS compatible.
1	RW(RP/DSP), RO(EP/DSP), RsvdP (Retimer)	**Multi-stack enable**: When set, multi stack training is enabled else not. In first gen of UCIe, only 2 stacks max are possible. Default is same as 'Multi-stack Capable' bit in UCIe Link Capability register.
5:2	RW(RP/DSP), RsvdP (Others)	**Target Link Width** 0h: Reserved 1h: Reserved 2h: x16 3h: x32 4h: x64 5h: x128 6h: x256 Others are Reserved. Default is same as 'Max Link Width' field in UCIe Link Capability Register.

Table 65. UCIe Link DVSEC - UCIe Link Control

Bit	Attribute	Description
9:6	RW(RP/DSP), RsvdP (Others)	**Target Link Speed** 0h: 4GT/s 1h: 8GT/s 2h: 12GT/s 3h: 16GT/s 4h: 24GT/s 5h: 32GT/s Others: Reserved Default is same as 'Max Link speed' field in UCIe Link Capability Register.
10	RW, with auto clear(RP/DSP), RsvdP (Others)	**Start UCIe Link training** - When set to 1, Link training starts with Link Control bits programmed in this register and with the protocol layer capabilities. Bit is automatically cleared when Link training completes with either success or error. The status register captures the final status of the Link training. Note that if the Link is up when this bit is set to 1 from 0, the Link will go through full training through Link_Down state thus resetting everything beneath the Link. Primary usage intended for this bit is for initial Link training out of reset on the host side. Note: For downstream ports of a switch with UCIe, local HW/ FW has to autonomously initiate Link training after a conventional reset, without waiting for higher level SW to start the training via this bit, to ensure backward compatibility. Default is 0.
11	RW with auto clear (RP/ DSP), RsvdP (Others)	**Retrain UCIe** Link - When set to 1, Link that is already up (Link_status=up) will be retrained without going through Link_Down state. SW can use this bit to potentially recover from Link errors. If the Link is down (Link_status=down) when this bit is set, there is no effect from this bit being set. SW should use the 'Start Link training' bit in case the Link is down. The Link_status bit in the status register can be read by software to determine whether to use this bit or not. Note that when retrain happens, the Link speed or width can change because of reliability reasons, and it will be captured through the appropriate status bit in the Link Status register. Bit is automatically cleared when Link retraining completes with either success or error (as reported via the appropriate status bits in the Link Status register) or if the Link retrain did not happen at all for the reason stated earlier. Default is 0.
12	RW	**PHY layer Clock gating enable** - When set, the dynamic clock gating of the forward clocked is enabled. Otherwise its free running. Default is 1.
31:13	RsvdP	Reserved

7.5.1.6 UCIe Link DVSEC - UCIe Link Status (Offset 0x14)

Basic UCIe Link status bits are in this register.

Table 66. UCIe Link DVSEC - UCIe Link Status

Bit	Attribute	Description
0	RO	**Raw_Mode_Enabled**: If set indicates the Adapter negotiated Raw_Mode operation with remote Link partner. This bit is only valid when Link Status bit in this register indicates 'Link Up'.
1	RsvdZ (retimer), RO(Others)	**Multi-stack enabled**: When set, multi stack training has been enabled with remote training partner. This bit is only valid when Link Status bit in this register indicates 'Link Up'.
6:2	RsvdZ	Reserved
10:7	RO	**Link Width enabled** 0h: Reserved 1h: x8 2h : x16 3h : x32 4h : x64 5h : x128 6h : x256 This has meaning only when Link status bit shows Link is up.
14:11	RO	**Link Speed enabled** 0h: 4GT/s 1h: 8GT/s 2h: 12GT/s 3h: 16GT/s 4h: 24GT/s 5h: 32GT/s Others: Reserved This field has meaning only when Link status field shows Link is up
15	RO	**Link Status** 0 - Link is down. 1 - Link is up Transitioning a Link from down to up requires a full Link retraining, which can be achieved using one of these methods 1.Start Link training via the bits in the UCIe Link Control register of the upstream device 2.Using the protocol layer reset bit associated with the Link, like the SBR bit in the BCTL register of the RP P2P space 3.Using the protocol layer Link Disable bit associated with the Link, like the Link Disable bit in the Link CTL register of the PCIe capability register in the RP P2P space, and then releasing the disable. Note if the Link is actively retraining, this bit reflects a value of 1b.

Table 66. UCIe Link DVSEC - UCIe Link Status

Bit	Attribute	Description
16	RO	**Link Training/Retraining** 1b - Currently Link is training or retraining 0b - Link is not training or retraining
17	RW1C(RP/DSP), RsvdZ (Others)	**Link Status changed** 1b - Link either transitioned from up to down or down to up. 0b - No Link status change since the last time SW cleared this bit
18	RW1C(RP/DSP), RsvdZ (Others)	**HW autonomous BW changed** UCIe autonomously changed the Link width or speed to correct Link reliability related issues
19	RW1CS	**Detected UCIe Link correctable error** Further details of specific type of correctable error is found in Table 83 register.
20	RW1CS	**Detected UCIe Link Uncorrectable Non-fatal error** Further details of specific type of correctable error is found in Table 80 register.
21	RW1CS	**Detected UCIe Link Uncorrectable Fatal error** Further details of specific type of correctable error is found in Table 80 register.
31:22	RsvdZ	Reserved

7.5.1.7 UCIe Link DVSEC - Link Event Notification Control (Offset 0x18)

Link event notification related controls are in this register.

Table 67. UCIe Link DVSEC - Link Event Notification Control

Bit	Attribute	Description
0	RW(RP/DSP), RsvdP (Others)	**'Link Status changed' UCIe Link Event Interrupt enable:** 0: Reporting of this event via interrupt is not enabled 1: Reporting of this event via interrupt is enabled. Default is 0
2	RW(RP/DSP), RsvdP (Others)	**'HW autonomous BW changed' UCIe Link Event Interrupt enable** 0: Reporting of this event via interrupt is not enabled 1: Reporting of this event via interrupt is enabled Default is 0

Table 67. UCIe Link DVSEC - Link Event Notification Control

Bit	Attribute	Description
10:2	RsvdP	Reserved
15:11	RO(RP/DSP), RsvdP(Others)	**Link Event Notification Interrupt number** This field indicates which MSI vector (for host UCIe Links), or MSI/MSI-X vector (for switch DSP UCIe Links) is used for the interrupt message generated in association with the events that are controlled via this register. For MSI, the value in this field indicates the offset between the base Message Data and the interrupt message that is generated. Hardware is required to update this field so that it is correct if the number of MSI Messages assigned to the Function changes when software writes to the Multiple Message Enable field in the Message Control Register for MSI. For first generation of UCIe, maximum 2 interrupt vectors could be requested for UCIe related functionality and the 'Link event' is one of them. For MSI-X (applicable only for interrupts from Switch DSPs with UCIe Links), the value in this field indicates which MSI-X Table entry is used to generate the interrupt message. The entry must be one of the first 32 entries even if the Function implements more than 32 entries. For a given MSI-X implementation, the entry must remain constant. For UCIe related interrupts, a switch should request its interrupt requirements from either MSI or MSI-X capability but not both.

7.5.1.8 UCIe Link DVSEC - Error Notification Control (Offset 0x1A)

Link error notification related controls are in this register.

Note: This register only controls the propagation of the error condition and it has no impact on the setting of the appropriate status bits in the Link Status register, when the relevant error happens.

Table 68. UCIe Link DVSEC - Error Notification Control

Bit	Attribute	Description
0	RW(RP/DSP), RsvdP (Others)	**'Correctable error detected' protocol layer based reporting enable** 0: Reporting of this error via protocol layer mechanism is not enabled 1: Reporting of this error via protocol layer mechanism is enabled Default is 0 When enabled, the reported PCIe/CXL protocol layer correctable error type is 'Correctable internal error'. This bit is applicable for only RP/DSP.
1	RW	**'Correctable error detected' UCIe Link Error Interrupt enable** RP/DSP 0: Reporting of this error via UCIe Link Error interrupt is not enabled 1: Reporting of this error via UCIe Link Error interrupt is enabled EP/USP 0: Reporting of this error via side band error message is not enabled 1: Reporting of this error via side band error message is enabled Note that in the case of EP/USP connected to a retimer, their side band error message targets the retimer and how the retimer sends it across to the partner retimer is vendor specific. Retimer connected to RP/DSP 0: Reporting of this error via side band error message to RP/DSP is not enabled 1: Reporting of this error via side band error message to RP/DSP is enabled Retimer connected to EP/USP 0: Reporting of this error to the partner retimer is disabled. 1: Reporting of this error to the partner retimer is enabled. The specific mechanism for reporting the error to the partner retimer is vendor-specific. Default is 0

Table 68. UCIe Link DVSEC - Error Notification Control

Bit	Attribute	Description
2	RW(RP/DSP), RsvdP (Others)	**'Uncorrectable non-fatal error detected' protocol layer based reporting enable** 0: Reporting of this error via protocol layer mechanism is not enabled 1: Reporting of this error via protocol layer mechanism is enabled Default is 0 This bit is applicable for only RP/DSP.
3	RW	**'Uncorrectable non-fatal error detected' UCIe Link Error Interrupt enable** RP/DSP 0: Reporting of this error via UCIe Link Error interrupt is not enabled 1: Reporting of this error via UCIe Link Error interrupt is enabled EP/USP 0: Reporting of this error via side band error message is not enabled 1: Reporting of this error via side band error message is enabled Note that in the case of EP/USP connected to a retimer, their side band error message targets the retimer and how the retimer sends it across to the partner retimer is vendor specific. Retimer connected to RP/DSP 0: Reporting of this error via side band error message to RP/DSP is not enabled 1: Reporting of this error via side band error message to RP/DSP is enabled Retimer connected to EP/USP 0: Reporting of this error to the partner retimer is disabled. 1: Reporting of this error to the partner retimer is enabled. The specific mechanism for reporting the error to the partner retimer is vendor specific. Default is 0

Table 68. UCIe Link DVSEC - Error Notification Control

Bit	Attribute	Description
4	RW (RP/DSP), RsvdP (Others)	**'Uncorrectable fatal error detected' protocol layer based reporting enable** 0: Reporting of this error via protocol layer mechanism is not enabled 1: Reporting of this error via protocol layer mechanism is enabled Default is 0 When enabled, the reported PCIe/CXL protocol layer uncorrectable error type is 'Uncorrectable internal error' This bit is applicable for only RP/DSP.
5	RW	**'Uncorrectable fatal error detected' UCIe Link Error Interrupt enable** RP/DSP 0: Reporting of this error via UCIe Link Error interrupt is not enabled 1: Reporting of this error via UCIe Link Error interrupt is enabled EP/USP 0: Reporting of this error via side band error message is not enabled 1: Reporting of this error via side band error message is enabled Note that in the case of EP/USP connected to a retimer, their side band error message targets the retimer and how the retimer sends it across to the partner retimer is vendor specific. Retimer connected to RP/DSP 0: Reporting of this error via side band error message to RP/DSP is not enabled 1: Reporting of this error via side band error message to RP/DSP is enabled Retimer connected to EP/USP 0: Reporting of this error to the partner retimer is disabled. 1: Reporting of this error to the partner retimer is enabled. The specific mechanism for reporting the error to the partner retimer is vendor specific. Default is 0

Table 68. UCIe Link DVSEC - Error Notification Control

Bit	Attribute	Description
10:7	RsvdP	Reserved
15:11	RW	**Link Error Notification Interrupt number** This field indicates which MSI vector (for host UCIe Links), or MSI/MSI-X vector (for switch DSP UCIe Links) is used for the interrupt message generated in association with the events that are controlled via this register. For MSI, the value in this field indicates the offset between the base Message Data and the interrupt message that is generated. Hardware is required to update this field so that it is correct if the number of MSI Messages assigned to the Function changes when software writes to the Multiple Message Enable field in the Message Control Register for MSI. For first generation of UCIe, maximum 2 interrupt vectors could be requested for UCIe related functionality and the 'Error' is one of them. For MSI-X (applicable only for interrupts from Switch DSPs with UCIe Links), the value in this field indicates which MSI-X Table entry is used to generate the interrupt message. The entry must be one of the first 32 entries even if the Function implements more than 32 entries. For a given MSI-X implementation, the entry must remain constant. For UCIe related interrupts, a switch should request its interrupt requirements from either MSI or MSI-X capability but not both.

7.5.1.9 UCIe Link DVSEC - Register Locator 0, 1, 2 (Starts at offset 0x1E)

The starting address of the MMIO mapped register blocks for D2D/PHY, Compliance/Test and Implementation-specifics are located by SW via these registers.

Note: All register blocks start with a header section that indicate the size of the block in multiples of 4KB.

Table 69. UCIe Link DVSEC - Register Locator 0, 1, 2

Bit	Attribute	Description
2:0	RO	**Register BIR** For UCIe DVSEC capability in host CiRB, Switch CiSRB and in UCIe Retimer, this field is reserved. For others, its defined as follows: Indicates which one of a Dev0/Fn0 Base Address Registers, located beginning at 10h in Configuration Space, or entry in the Enhanced Allocation capability with a matching BAR Equivalent Indicator (BEI), is used to map the UCIe Register blocks into Memory Space. Defined encodings are: o 0 Base Address Register 10h o 1 Base Address Register 14h o 2 Base Address Register 18h o 3 Base Address Register 1Ch o 4 Base Address Register 20h o 5 Base Address Register 24h All other Reserved. The Registers block must be wholly contained within the specified BAR. For a 64-bit Base Address Register, the Register BIR indicates the lower DWORD.
6:3	RO	**Register Block Identifier** Identifies the type of UCIe register blocks. Defined encodings are: o 0h UCIe D2D/PHY Register Block o 1h UCIe Test/Compliance Register Block o 2h Implementation specific register block o Others Reserved The same register block identifier value cannot be repeated in multiple Register Locator entries.
11:7	RsvdP	Reserved
63:12	RO	**Register Block Offset** A[63:12] 4KB aligned offset from the starting address of the Dev0/Fn0 BAR pointed to by the Register BIR field (for EP, Switch USP) or from the start of CiRB/CiSRB region (for hosts/ Switch). This field is reserved for retimers.

7.5.1.10 UCIe Link DVSEC - Sideband Mailbox Index (Offset is design dependent)

Mailbox registers are to be implemented by all hosts with UCIe Links. Switches with downstream UCIe Links and EP/USP when paired with UCIe Retimer, should also implement this register. Note that accesses to mailbox are inherently non-atomic in nature and hence its upto higher level software to coordinate access to any mailbox related register so that one agent does not step on another agent using the mailbox mechanism. Those mechanisms for software coordination are outside the scope of the specification.

Table 70. UCIe Link DVSEC - Sideband Mailbox Index

Bit	Attribute	Description
4:0	RW	**Opcode** 00000b 32b Memory Read 00001b 32b Memory Write 00100b 32b Configuration Read 00101b 32b Configuration Write 01000b 64b Memory Read 01001b 64b Memory Write 01100b 64b Configuration Read 01101b 64b Configuration Write OthersReserved Default 00100
12:5	RW	**BE[7:0]** Default 0x0F
39:13	RW	**Address[26:0]** Format for this field is as defined in the sideband interface definition in Chapter 6. Note: The address offset defined as part of this address field is DWORD aligned for 32bit accesses and QWORD aligned for 64bit accesses. Default is 0
63:40	RsvdP	Reserved

7.5.1.11 UCIe Link DVSEC - Sideband Mailbox Data (Offset is design dependent)

Table 71. UCIe Link DVSEC - Sideband Mailbox Data

Bit	Attribute	Description
63:0	RW	For sideband write opcodes, this carries the write data to the destination For sideband read opcodes, this carries the data read from the destination when Write/Read Trigger bit in the Mailbox Control register is clear, after it was initially set. Till the Write/Read trigger bit is clear on reads, this field's value is undefined. For 32 bit writes/reads, only bottom 32bits of this register are valid.

7.5.1.12 UCIe Link DVSEC - Sideband Mailbox Control (Offset is design dependent)

Table 72. UCIe Link DVSEC - Sideband Mailbox Control

Bit	Attribute	Description
0	RW, with auto clear	**Write/Read trigger**: When this bit is written to a 1 from a value of 0, the mailbox generates traffic on the sideband interface, using the contents of the Mailbox Header and Data registers. This bit automatically clears when the write or read access triggered by this bit being set, is complete on the side band bus. SW can poll this bit to know when the write/read has actually completed at the destination. It can then go read the Mailbox data register for the read data.

7.5.1.13 UCIe Link DVSEC - Sideband Mailbox Status (Offset is design dependent)

Table 73. UCIe Link DVSEC - Sideband Mailbox Status

Bit	Attribute	Description
1:0	RW1C(RP/DSP), RW1C(EP/USP), when implemented	**Write/Read status:** 00 - CA received 01 - UR received 01 - Reserved 11 - Success This bit has valid value only when the Write/Read Trigger bit is cleared from being a 1 prior to it.
7:2	RsvdZ	Reserved

7.5.1.14 UCIe Link DVSEC - Requester ID (Offset is design dependent)

Table 74. UCIe Link DVSEC - Requester ID

Bit	Attribute	Description
23:0	RW(RP)/RsvdP (Others)	Applicable only for host side UCIe Links. Segment No: Bus No: Dev No: Fn No for MSIs triggered on behalf of the associated UCIe Link Note: For MSI's issued on behalf of UCIe Links on downstream ports of switches, the Switch USP BDF is used. UCIe Link DVSEC capabilities in CiSRB implement this as RO 0.
31:24	RsvdP	Reserved

7.5.1.15 UCIe Link DVSEC - Associated Port Numbers (Offset is design dependent)

These registers apply only to UCIe Link DVSEC capabilities present in CiSRB.

Table 75. UCIe Link DVSEC - Associated Port Numbers

Bit	Attribute	Description
7:0	RO	**Port Number 1** - 'Port number' of the 1st switch DSP associated with this UCIe. This value is from the Link Capabilities register of that switch DSP.
15:8	RO	**Port Number 2** - 'Port number' of the 2nd switch DSP associated with this UCIe, if any. If there is no 2nd switch DSP associated with this UCIe Link, this register is treated as reserved and should not be included as part of the "length" field of the 'Designated Vendor specific Header 1' register and SW should not consider this as part of the DVSEC capability. Note max 2 Port numbers can be associated with a UCIe Link in the first revision of the specification.

7.5.1.16 Examples of setting the Length field in DVSEC for various Scenarios

Example#1: UCIe EP supporting 2 Register Locators and not associated with a UCIe-Retimer, would set the length field in DVSEC capability to indicate 48B.

Example#2: Host CiRB supporting 3 register locators would set the length to indicate 84B.

Example#3: Switch CiSRB supporting 3 register locators and associated with just 1 DSP port to a UCIe Link, would set the length to indicate 85B.

7.5.2 UCIe Switch Register Block (CiSRB) DVSEC Capability

This capability can only be present in the config space of the upstream port of a Switch. There can be multiple of these in the same USP config space.

7.5.2.1 PCI Express Extended Capability Header (Offset 0x0)

Set as follows for UCIe Switch Register Block DVSEC. All bits in this register are RO.

Table 76. CiSRB DVSEC - PCI Express Extended Capability Header

Field	Bit Location	Value	Comments
Capability ID	15:0	0023h	Value for PCI Express DVSEC capability
Revision ID	19:16	1h	Latest revision of the DVSEC capability
Next Capability Offset	31:20	Design Dependent	

7.5.2.2 Designated Vendor Specific Header 1 (Offset 0x4), 2 (Offset 0x8)

A few things to note on the various fields described in Table 62. DVSEC Revision ID field represents the version of the DVSEC structure. The DVSEC Revision ID is incremented whenever the structure is extended to add more functionality. Backward compatibility shall be maintained during this process. For all values of n, DVSEC Revision ID n+1 structure may extend Revision ID n by replacing fields that are marked as reserved in Revision ID n, but must not redefine the meaning of existing fields. Software that was written for a lower Revision ID may continue to operate on UCIe DVSEC structures with a higher Revision ID, but will not be able to take advantage of new functionality.

All bits in this register are RO.

Table 77. CiSRB DVSEC - Designated Vendor Specific Header 1, 2

Register	Field	Bit Location	Value
Designated Vendor-Specific Header 1 (offset 04h)	DVSEC Vendor ID	15:0	TBD
	DVSEC Revision	19:16	0h
	Length	31:20	14h
Designated Vendor-Specific Header 2 (offset 08h)	DVSEC ID	15:0	1h

7.5.2.3 UCIe Switch Register Block (CiSRB) Base Address (Offset 0xC)

All bits in this register are RO.

Table 78. CiSRB DVSEC - CiSRB Base Address

Bit	Attributes	Description
0	RO	**Register BIR** Indicates which one of a Switch USP Function's Base Address Registers, located beginning at 10h in Configuration Space, or entry in the Enhanced Allocation capability with a matching BAR Equivalent Indicator (BEI), is used to locate the UCIe Switch Register Block. Defined encodings are: o 0 Base Address Register 10h o 1 Base Address Register 14h o All other Reserved. The Registers block must be wholly contained within the specified BAR. For a 64-bit Base Address Register, the Register BIR indicates the lower DWORD.
11:1	RsvdP	Reserved
63:12	RO	**Register Block Offset** A[63:12] 4KB-aligned offset from the starting address of the Switch USP BAR indicated by the Register BIR field. The BAR value + Offset indicated in this register is where the UCIe Switch Register Block (CiSRB) starts. Ex: If this register is 100, CiSRB starts at the <64bit BAR value + 0x100000>

7.5.3 D2D/PHY Register Block

These registers occupy of 8KB of register space. The first 4KB are for the D2D Adapter, and the next 4KB are for the Physical Layer. The D2D Adapter registers are enumerated below. The location of these registers in system MMIO region is as described in section 7.3.

7.5.3.1 UCIe Register Block Header

Table 79. D2D/PHY Register Block - UCIe Register Block Header (Offset 0x0)

Bit	Attributes	Description
00h	15:0	**Vendor ID:** Default is set to Vendor ID assigned for UCIe Consortium - TBD
31:16	RO	**Vendor ID Register Block** Set to 0h to indicate D2D/PHY register block
35:32	RO	**Vendor Register Block Version** Set to 0h
63:36	RsvdP	Reserved
95:64	RO	**Vendor Register Block Length** - The number of bytes in the register block including the UCIe Register block header. Default is 2000h.
127:96	RsvdP	Reserved

7.5.3.2 Uncorrectable Error Status Register (Offset 0x010)

Table 80. Uncorrectable Error Status Register

Bit	Attribute	Description
0	RW1CS	**Adapter Timeout:** Set to 1b by hardware if greater than 8ms has elapsed for Adapter handshakes with its remote Link partner. The Header Log 2 register captures the reason for a timeout. This error will bring the main Link Down. Default Value is 0b.
1	RW1CS	**Receiver Overflow**: Set to 1b by hardware if Receiver overflow errors are detected. The Header Log 2 register captures the encoding to indicate the type of Receiver overflow. This error will bring the Link Down. Default Value is 0b.
2	RW1CS	**Internal Error**: Set to 1b by hardware if an internal Data path error is detected. Examples of such errors include (but not limited to) uncorrectable error correcting code (ECC) error in the Retry buffer, sideband parity errors etc. This error will bring the Link Down. It includes fatal error indicated by the Physical Layer that brought the Link Down. Default Value is 0b.
3	RW1CS(RP/ DSP/Retimer), RsvdZ(Others)	**Side band Fatal Error Message received**: Set to 1b by hardware if the Adapter received a Fatal {ErrMsg} on sideband. Default Value is 0b.
4	RW1CS(RP/ DSP/Retimer), RsvdZ(Others)	**Side band Non-Fatal Error Message received**: Set to 1b by hardware if the Adapter received a Non-Fatal {ErrMsg} on sideband. Default Value is 0b.
5	RW1CS	**Invalid Parameter Exchange**: Set to 1b if the Adapter was not able to determine a valid protocol or Flit format for operation.
31:6	RsvdZ	Reserved

7.5.3.3 Uncorrectable Error Mask Register (Offset 0x014)

The Uncorrectable Error Mask Register controls reporting of individual errors. When a bit is 1b, the corresponding error status bit in Uncorrectable Error Status Register is not forwarded to the Protocol Layer for logging or escalation.

Table 81. Uncorrectable Error Mask Register

Bit	Attribute	Description
0	RWS	**Adapter Timeout Mask** Default Value is 1b.
1	RWS	**Receiver Overflow Mask** Default Value is 1b.
2	RWS	**Internal Error Mask** Default Value is 1b.
3	RWS	**Side band Fatal Error Message received Mask** Default Value is 1b.
4	RWS	**Side band Non-Fatal Error Message received Mask** Default Value is 1b.
5	RWS	**Invalid Parameter Exchange Mask** Default Value is 1b.
31:6	RsvdP	Reserved

7.5.3.4 Uncorrectable Error Severity Register (Offset 0x018)

The Uncorrectable Error Severity register controls whether an individual error is reported as a Non-fatal or Fatal error. An error is reported as a fatal uncorrectable error when the corresponding bit in the severity register is 1b. If the bit is 0b, the corresponding error is reported as a non-fatal uncorrectable error.

Table 82. Uncorrectable Error Severity Register

Bit	Attribute	Description
0	RWS	**Adapter Timeout Severity** Default Value is 1b.
1	RWS	**Receiver Overflow Severity** Default Value is 1b.
2	RWS	**Internal Error Severity** Default Value is 1b.
3	RWS	**Side band Fatal Error Message received Severity** Default Value is 1b.
4	RWS	**Side band Non-Fatal Error Message received Severity** Default Value is 0b.
5	RWS	**Invalid Parameter Exchange Severity** Default Value is 1b
31:6	RsvdP	Reserved

7.5.3.5 Correctable Error Status Register (Offset 0x01C)

Table 83. Correctable Error Status Register

Bit	Attribute	Description
0	RW1CS	**CRC Error Detected**: Set to 1b by hardware if the Adapter detected a CRC Error when Adapter Retry was negotiated with remote Link partner. Default Value is 0b.
1	RW1CS	**Adapter LSM transition to Retrain**: Set to 1b by hardware if the Adapter LSM transitioned to Retrain state. Default Value is 0b.
2	RW1CS	**Correctable Internal Error**: Set to 1b by hardware if an internal correctable Data path error is detected. Examples of such errors include (but not limited to) correctable error correcting code (ECC) error in the Retry buffer, Physical Layer indicated correctable error on RDI etc. LinkError state transition on RDI and FDI must also be logged in this bit. Default Value is 0b.
3	RW1CS(RP/ DSP/Retimer), RsvdZ(Others)	**Side band Correctable Error Message received**: Set to 1b by hardware if the Adapter received a Correctable {ErrMsg} on sideband with Device origin encoding in the message information. Default Value is 0b.
31:4	RsvdZ	Reserved

7.5.3.6 Correctable Error Mask Register (Offset 0x020)

The Correctable Error Mask Register controls the reporting of individual errors. When a bit is 1b in this register, the corresponding error status bit is not set upon the error event, and the error is not signaled.

Table 84. Correctable Error Mask Register

Bit	Attribute	Description
0	RWS	**CRC Error Detected Mask** Default Value is 1b.
1	RWS	**Adapter LSM transition to Retrain Mask** Default Value is 1b.
2	RWS	**Correctable Internal Error Mask** Default Value is 1b.
3	RWS	**Device Correctable Error Message received Mask** Default Value is 1b.
4	RWS	**Retimer Correctable Error Message received Mask** Default Value is 1b.
31:5	RsvdP	Reserved

7.5.3.7 Header Log 1 Register (Offset 0x024)

Table 85. Header Log 1 Register

Bit	Attribute	Description
63:0	ROS	**Header Log 1:** This logs the header for the first register accesses that received a completion with Completer Abort status or received a completion with Unsupported Request. Note that register accesses that time out are not required to be logged at the requester. If the ’Received UR’ and ’Received CA’ status bits are both cleared in the ’Correctable Error Status’ register, this field’s value is undefined. Default Value is 0.

7.5.3.8 Header Log 2 Register (Offset 02Ch)

Table 86. Header Log 2 Register

Bit	Attribute	Description
3:0	ROS	**Adapter Timeout encoding**: Captures the reason for the first Adapter Timeout that was logged in Uncorrectable Error Status. Default Value is 0000b. The encodings are interpreted as follows: 0001b: Parameter Exchange flow timed out 0010b: Adapter LSM request to remote Link partner did not receive a response after 8ms. Bits 11:9 capture the specific state request that did not receive a response. Bit 12 of this register captures which Adapter LSM timed out. 0011b: Adapter LSM transition to Active timeout. This is recorded in case the Adapter never received Active Request from remote Link partner for 8ms after sending an Active Request on sideband even though it received an Active Response. Bit 12 of this register captures which Adapter LSM timed out. 0100b: Retry Timeout - no Ack or Nak received after 8ms, when Retry was enabled. Timeout counter is only incremented while RDI is in Active and Adapter's Retry buffer is not empty. 0101b: Local sideband access timeout 0110b: Retimer credit return timeout - no Retimer credit received for greater than 8ms if one or more Retimer credits have been consumed by the Adapter. This timer is only counting during Active state. If RDI moves to Retrain, this timer must be Reset since the Retimer credits are also Reset. 0111b: Remote Register Access timeout. This is triggered when if the Adapter has observed N timeouts for Register Accesses where N is >= register access timeout threshold. other encodings are reserved. If the Adapter Timeout status bit is cleared in the 'Uncorrectable Error Status' register, this field's value is undefined.
6:4	ROS	**Receiver Overflow encoding**: Captures the encoding for the first Receiver overflow error that occurred. Default value is 000b. The encodings are interpreted as follows: 001b: Transmitter Retry Buffer overflow 010b: Retimer Receiver Buffer overflow 011b: FDI sideband buffer overflow 100b: RDI sideband buffer overflow other encodings are reserved. If the Receiver overflow status bit is cleared in the 'Uncorrectable Error Status' register, this field's value is undefined.

Bit	Attribute	Description
9:7	ROS	**Adapter LSM response type:** 001b: Active 010b: L1 011b: L2 100b: LinkReset 101b: Disable Other encodings are reserved If the Adapter Timeout status bit is cleared in the 'Uncorrectable Error Status' register, this field's value is undefined.
10	ROS	**Adapter LSM id :** 0b : Adapter LSM 0 timed out 1b : Adapter LSM 1 timed out
12:11	RsvdZ	Reserved
13	RO	**Parameter Exchange Successful** : Hardware updates this bit to 1b after successful Parameter exchange with remote Link partner.
17:14	ROS	**Flit Format**: This field logs the negotiated Flit format, it is the current snapshot of the format the Adapter is informing to the Protocol Layer. Refer to Chapter 3.0 for the definitions of these formats. The encodings are: 0001b - Format 1 0010b - Format 2 0011b - Format 3 0100b - Format 4 0101b - Format 5 0110b - Format 6 0111b - Format 7 Other encodings are Reserved
22:18	ROS	**First Fatal Error Indicator**: 5-bit encoding that indicates which bit of Uncorrectable Error Status errors was logged first. If no bits are 1b in the Uncorrectable Error Status register, then the value of this field has no meaning.
31:23	RsvdZ	Reserved

7.5.3.9 Error and Link Testing Control Register (0ffset 0x030)

Table 87. Error and Link Testing Control Register

Bit	Attribute	Description
3:0	RW	**Remote Register Access Threshold**: Indicates the number of consecutive timeouts for remote register accesses that must occur before the Register Access timeout is logged and the error escalated to a Link_Status=Down condition. Default Value is 0100b.
4	RW	**Runtime Link Testing Tx Enable**: Software writes to this bit to enable Parity byte injections in the data stream as described in section 3.8. Runtime Link Rx Enable must be set to 1b for remote Link Partner for successful enabling of this mode. Default Value is 0b.
5	RW	**Runtime Link Testing Rx Enable**: Software writes to this bit to enable Parity byte checking in the data stream as described in section 3.8. Runtime Link Tx Enable must be set to 1b for remote Link Partner for successful enabling of this mode. Default Value is 0b.
8:6	RW	**Number of 64 Byte Inserts**: Software writes to this to indicate the number 64 Byte inserts are done at a time for Runtime Link Testing. Software programs this based on the current Link width (taking into account how many modules per Adapter are currently in operation). The encodings are: 000b: one 64B insert 001b: two 64B inserts 010b: four 64B inserts Other encodings are reserved. Default value is 000b.
9	RW1C	**Parity Feature Nak received**: Hardware updates this bit if it receives a Nak from remote Link partner when attempting to enable Runtime Link Testing.
12:10	RsvdP	Reserved
14:13	RW	**CRC Injection Enable** : Software writes to this bit to trigger CRC error injections, The error is injected by inverting 1, 2 or 3 bits in the CRC bytes. The specific bits inverted are implementation specific. The CRC injection must not happen for Flits that are already inverting CRC bits for Viral handling. The encodings are interpreted as : 00b : CRC Injection is Disabled. 01b : 1 bit is inverted 10b : 2 bits are inverted 11b : 3 bits are inverted. Default Value is 00b.

Bit	Attribute	Description
16:15	RW	**CRC Injection Count** : Software writes to this bit to program the number of CRC injections. It only takes effect if CRC injection Enable is not Disabled. 00b : Single Flit is corrupted. CRC Injection Busy is reset to 0b after single Flit corruption. 01b: A CRC error is injected every 8 Flits. Hardware continues to inject a CRC error every 8 Flits until CRC Injection Enable is 00b. CRC Injection Busy is reset to 0b only after CRC Injection Enable is 00b. 10b: A CRC error is injected every 16 Flits. Hardware continues to inject a CRC error every 16 Flits until CRC Injection Enable is 00b. CRC Injection Busy is reset to 0b only after CRC Injection Enable is 00b. 11b: A CRC error is injected every 64 Flits. Hardware continues to inject a CRC error every 64 Flits until CRC Injection Enable is 00b. CRC Injection Busy is reset to 0b only after CRC Injection Enable is 00b.
17	RO	**CRC Injection Busy** : Hardware loads a 1b to this bit once it has begun CRC Injection. Software is permitted to poll on this bit. See CRC Injection Count description to see how this bit returns to 0b.
31:18	RsvdP	Reserved

7.5.3.10 Runtime Link Testing Parity Log 0 (Offset 0x034)

Table 88. Runtime Link Testing Parity Log 0 Register

Bit	Attribute	Description
63:0	RW1C	**Parity Log for Module 0**: Hardware updates the bit corresponding to the parity error byte with error over the period when Runtime Link Testing was enabled at Rx. Default Value is 0.

7.5.3.11 Runtime Link Testing Parity Log 1 (Offset 0x03C)

Table 89. Runtime Link Testing Parity Log 1 Register

Bit	Attribute	Description
63:0	RW1C	**Parity Log for Module 1**: Hardware updates the bit corresponding to the parity error byte with error over the period when Runtime Link Testing was enabled at Rx. Default Value is 0. This is register is only applicable if the Adapter is designed for handling two or more Physical Layer modules. It is reserved otherwise.

7.5.3.12 Runtime Link Testing Parity Log 2 (Offset 0x044)

Table 90. Runtime Link Testing Parity Log 2 Register

Bit	Attribute	Description
63:0	RW1C	**Parity Log for Module 2**: Hardware updates the bit corresponding to the parity error byte with error over the period when Runtime Link Testing was enabled at Rx. Default Value is 0. This is register is only applicable if the Adapter is designed for handling four Physical Layer modules. It is reserved otherwise.

7.5.3.13 Runtime Link Testing Parity Log 3 (Offset 0x04C)

Table 91. Runtime Link Testing Parity Log 3 Register

Bit	Attribute	Description
63:0	RW1C	**Parity Log for Module 3**: Hardware updates the bit corresponding to the parity error byte with error over the period when Runtime Link Testing was enabled at Rx. Default Value is 0. This is register is only applicable if the Adapter is designed for handling four Physical Layer modules. It is reserved otherwise.

7.5.3.14 Advertised Adapter Capability Log (Offset 0x054)

Table 92. Advertised Adapter Capability Log Register

Bit	Attribute	Description
63:0	RW1C	**Advertised Adapter Capability**: Hardware updates the bits corresponding to the data bits it sent in the {AdvCap.Adapter} sideband message. Default Value is 0.

7.5.3.15 Finalized Adapter Capability Log (Offset 0x05C)

Table 93. Finalized Adapter Capability Log Register

Bit	Attribute	Description
63:0	RW1C	**Finalized Adapter Capability**: Hardware updates the bits corresponding to the data bits it sent (DP) or received (UP) in the {FinCap.Adapter} sideband message. Default Value is 0.

7.5.3.16 Advertised CXL Capability Log (Offset 0x064)

Table 94. Advertised CXL Capability Log Register

Bit	Attribute	Description
63:0	RW1C	**Advertised CXL Capability**: Hardware updates the bits corresponding to the data bits it sent in the {AdvCap.CXL} sideband message. Default Value is 0.

7.5.3.17 Finalized CXL Capability Log (Offset 0x06C)

Table 95. Finalized CXL Capability Log Register

Bit	Attribute	Description
63:0	RW1C	**Finalized CXL Capability**: Hardware updates the bits corresponding to the data bits it sent (DP) or received (UP) in the {FinCap.CXL} sideband message. Default Value is 0.

7.5.3.18 PHY Capability (Offset 1000h)

This register is global, and not per module.

Table 96. Physical Layer Capability Register

Bit	Attribute	Description
2:0	RO	**Module to Byte mapping** 00h: no Module reversal (LSB Module to LSB Module) 01h: MSB Module to LSB module connectivity Reserved"
3	RO	**Terminated Link**: If set to 1b, it indicates that the Receiver is terminated.
4	RO	**TX Equalization support:** 0: TXEQ not supported 1: TXEQ supported"
10:5	RO	**Supported Vswing encodings** 01h: 0.4V 02h: 0.45V 03h: 0.5V 04h: 0.55V 05h: 0.6V 06h: 0.65V 07h: 0.7V 08h: 0.75V 09h: 0.8V 0Ah: 0.85V 0Bh: 0.9V 0Ch: 0.95V 0Dh: 1.0V 0Eh: 1.05 0Fh: 1.1V 10h: 1.15V Others Reserved
12:11	RO	**Clock Mode support:** 0h: Supports both free running and strobe m0des 1h: Strobe mode only 2h: Free running mode only
14:13	RO	**Clock phase support:** 0h: Differential clock only 1h: Quadrature clock only 2h: Both differential and quadrature clock
15	RO	**Package type** 0h: Advanced Package 1h: Standard Package
16	RO	**Tightly coupled mode (TCM) support** 0h: TCM not Supported 1h: TCM supported
31:17	RsvdP	Reserved

7.5.3.19 PHY Control (Offset 1004h)

This register is global, and not per module.

Table 97. Physical Layer Control Register

Bit	Attribute	Description
2:0	RW	**Module to Byte mapping** 00h: no Module reversal (LSB Module to LSB Module) 01h: MSB Module to LSB module connectivity Others: Reserved Module to Byte mapping hardware writes it post hardware training Default is 0h
3	RW	**Rx Terminated Control** 0b: Rx Termination disabled 1b: Rx Termination enabled Default is same as 'Terminated Link' bit in PHY capability register
4	RW	**Tx Eq Enable:** 0b: Eq Disabled 1b: Eq Enabled Default is 0
5	RW	**Clock Mode Select** 0h: Strobe Mode 1h: Free running mode Default is 0
6	RW	**Clock phase support select:** 0h: Differential clock 1h: Quadrature clock
31:7	RsvdP	Reserved

7.5.3.20 PHY Status (Offset 1008h)

This register is global and not per module.

Table 98. Physical Layer Status Register

Bit	Attribute	Description
2:0	RO	**Module to Byte mapping** 00h: no Module reversal (LSB Module to LSB Module) 01h: MSB Module to LSB module connectivity Others: Reserved Module to Byte mapping hardware writes it post hardware training Default is 0h
3	RO	**Rx Terminated Status** 0b: Rx Termination disabled 1b: Rx Termination enabled Default is same as 'Terminated Link' bit in PHY capability register
4	RO	**Tx Eq Status:** 0b: Eq Disabled 1b: Eq Enabled Default is 0
5	RO	**Clock Mode Status:** 0h: Strobe Mode 1h: Free running mode Default is 0
6	RO	**Clock phase Status:** 0h: Differential clock 1h: Quadrature clock
7	RO	**Lane Reversal within Module:** Indicates if Lanes within a module are reversed 0b: Lanes within module not reversed 1b: Lanes within module are reversed
31:7	RsvdP	Reserved

7.5.3.21 PHY Initialization and Debug (Offset 100Ch)

This register is global, and not per module.

Table 99. Phy Init and Debug Register

Bit	Attribute	Description
2:0	RW	**Initialization control :** 0h: Initialize to Active 1h: Initialize to MBINIT (Debug mode) 2h: Initialize to MBTRIAN (Debug/compliance mode) Others: Reserved Default is 0
31:3	RsvdP	Reserved

7.5.3.22 Training Setup 1 (Offset 1010h)

This register is replicated per module. Offsets 1010h to 101Ch are used in 4B increments for multi-module scenarios

Table 100. Training Setup 1 Register

Bit	Attribute	Description
2:0	RW	**Data pattern used during training** 0h: Per Lane LFSR pattern 1h: Per Lane ID pattern Reserved Default = 0
5:3	RW	**Valid Pattern used during training** 0h: Functional valid pattern (1111 0000 (lsb first)) "Reserved Default = 0
9:6	RW	**Clock Phase control** 0h: Clock PI center found by Transmitter 1h: Left edge found through Data to clock training 2h: Right edge found through Data to clock training Reserved Default = 0
10	RW	**Training mode**: 0b: Continuous mode 1b: Burst Mode Default = 0
26:11	RW	**16 bit Burst Count**: Indicates the duration of selected pattern (UI count) Default = 4h
31:27	RsvdP	Reserved

7.5.3.23 Training Setup 2 (Offset 1020h)

This register is replicated per module. Offsets 1020h to 102Ch are used in 4B offset increments for multi-module scenarios.

Table 101. Training Setup 2 Register

Bit	Attribute	Description
15:0	RW	**16 bit idle count**: Indicates the duration of low following the burst (UI count) Default = 4h"
31:16	RW	**16 bit iterations**: Indicates the repeatation count of burst followed by idle(UI count) Default = 4h

7.5.3.24 Training Setup 3 (Offset 1030h)

This register is replicated per module. Offsets 1030h to 1048h are used in 8B offset increments for multi-module scenarios.

Table 102. Training Setup 3 Register

Bit	Attribute	Description
63:0	RW	**Lane mask**: Indicated the Lanes to mask during Rx comparison. Example 1h = Lane 0 is masked during comparison Default = 0 (no mask)

7.5.3.25 Training Setup 4 (Offset 1050h)

This register is replicated per module. Offsets 1050h to 105Ch are used in 4B offset increments for multi-module scenarios.

Table 103. Training Setup 4 Register

Bit	Attribute	Description
3:0	RW	**Repair Lane mask**: Indicated the Redunddant Lanes to mask during Rx comparison. Example 1h =RD0 is masked during comparison 2h: RD1 mask Default = 0 (no mask)
15:4	RW	**Max error Threshold in per Lane comparison** : Indicates threshold for error counting to start Default = 0 (all errors are counted)
31:16	RW	**Max error Threshold in aggregate comparison** : Indicates threshold for error counting to start Default = 0 (all errors are counted)

7.5.3.26 Current Lane Map Module 0 (Offset 1060h)

Table 104. Current Lane Map Module 0 Register

Bit	Attribute	Description
63:0	RW	**Current Rx Lane map (CLM) for Module-0** : If a bit is 1 it indicates the corresponding Lane is operational. For Standard package modules 16:63 are not applicable. Default Values is 0

7.5.3.27 Current Lane Map Module 1 (Offset 1068h)

Table 105. Current Lane Map Module 1 Register

Bit	Attribute	Description
63:0	RW	**Current Rx Lane map (CLM) for Module-1** : If a bit is 1 it indicates the corresponding Lane is operational. For Standard package modules 16:63 are not applicable. Default Values is 0 This register is reserved if module 1 is not present

7.5.3.28 Current Lane Map Module 2 (Offset 1070h)

Table 106. Current Lane Map Module 2 Register

Bit	Attribute	Description
63:0	RW	**Current Rx Lane map (CLM) for Module-2** : If a bit is 1 it indicates the corresponding Lane is operational. For Standard package modules 16:63 are not applicable. Default Values is 0 This register is reserved if module 1 is not present

7.5.3.29 Current Lane Map Module 3 (Offset 1078h)

Table 107. Current Lane Map Module 3 Register

Bit	Attribute	Description
63:0	RW	**Current Rx Lane map (CLM) for Module-3** : If a bit is 1 it indicates the corresponding Lane is operational. For Standard package modules 16:63 are not applicable. Default Values is 0 This register is reserved if module 1 is not present

7.5.3.30 Error Log 0 (Offset 1080h)

This register is replicated per module. Offsets 1080h to 108Ch are used in 4B offset increments for multi-module scenarios.

Table 108. Error Log 0 Register

Bit	Attribute	Description
7:0	ROS	**State N**: Captures the current Link training state machine status. State Encodings are given by: 00h RESET 01h SBINIT 02h MBINIT.PARAM 03h MBINIT.CAL 04h MBINIT.RepairCLK 05h MBINIT.RepairVAL 06h MBINIT.ReversalMB 07h MBINIT.RepairMB 08h MBTRAIN.VALVREF 09h MBTRAIN.DATAVREF 0Ah MBTRAIN.SPEEDIDLE 0Bh MBTRAIN.TXSELFCAL 0Ch MBTRAIN.RXSELFCAL 0Dh MBTRAIN.VALTRAINCENTER 0Eh MBTRAIN.DATATRAINVREF1 0Fh MBTRAIN.RXDESKEW 10h MBTRAIN.DATATRAINCENTER2 11h MBTRAIN.LINKSPEED 12h MBTRAIN.REPAIR 13h PHYRETRAIN 14h LINKINIT 15h ACTIVE 16h TRAINERROR 17h L1/L2 Default is 0
8	ROS	**Lane Reversal**: 1b indicates Lane Reversal within the module. Default is 0
9	ROS	**Width Degrade**: 1b indicates Moudule width Degrade. Applicabe to Standard package only. Default is 0
15:10	RsvdZ	Reserved
23:16	ROS	**State (N-1)**: Captures the state before State N was entered for Link training state machine. State encodings are the same as State N field. Default is 0
31:24	ROS	**State (N-2)**: Captures the state before State (N-1) was entered for Link training state machine. State encodings are the same as State N field. Default is 0

7.5.3.31 Error Log 1 (Offset 109Ch)

This register is replicated per module. Offsets 10900h to 109Ch are used in 4B offset increments for multi-module scenarios.

Table 109. Error Log 1 Register

Bit	Attribute	Description
7:0	ROS	**State (N-3)**: Captures the state status before State (N-2) was entered. State encodings are the same as State N field. Default is 0
8	ROS	**State Timeout Occurred**: Hardware sets this to 1b if a Link Training State machine state or sub-state timed out and it was escalated as a fatal error. Default value is 0b.
9	ROS	**Sideband Timeout Occurred**: Hardware sets this to 1b if a sideband handshake timed out, for example, if a RDI request did not get a response for 8ms. Sideband handshakes related to Link Training messages are not included here. Default value is 0b.
10	ROS	**Remote LinkError received**: Hardware sets this to 1b if remote Link partner requested LinkError transition through RDI sideband. Default value is 0b.
11	ROS	**Internal Error**: Hardware sets this to 1b if any implementation specific internal error occurred in the Physical Layer. Default value is 0b.
31:8	RsvdZ	Reserved

7.5.3.32 Runtime Link Test Control (Offset 1100h)

Table 110. Runtime Link Test Control

Bit	Attribute	Description
0	RW	**Apply Valid Repair**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Valid repair in the next Retrain cycle. This bit should have no effect for hardware on Standard Package. Default is 0
1	RW	**Apply Clock Repair**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Clock repair in the next Retrain cycle. This bit should have no effect for hardware on Standard Package. Default is 0
2	RW	**Apply Module 0 Lane Repair**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Lane repair for this logical module id at the next Retrain cycle, if this Module is operational. For Standard Package, this bit will trigger a width degrade for logical Module 0, if possible and relevant. Default value is 0b
3	RW	**Apply Module 1 Lane Repair**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Lane repair for this logical module id at the next Retrain cycle, if this Module is operational. For Standard Package, this bit will trigger a width degrade for logical Module 1, if possible and relevant. Default value is 0b These bits are reserved if Module 1 is not present.
4	RW	**Apply Module 2 Lane Repair**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Lane repair for this logical module id at the next Retrain cycle, if this Module is operational. For Standard Package, this bit will trigger a width degrade for logical Module 2, if possible and relevant. Default value is 0b These bits are reserved if Module 2 is not present.
5	RW	**Apply Module 3 Lane Repair**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Lane repair for this logical module id at the next Retrain cycle, if this Module is operational. For Standard Package, this bit will trigger a width degrade for logical Module 3, if possible and relevant. Default value is 0b These bits are reserved if Module 3 is not present.
6	RW	**Start**: Software writes to this bit before hitting Link Retrain bit to inform hardware that the contents of this register are valid.
7	RW	**Inject Stuck-at fault** : Software writes 1b to this bit to indicate hardware must inject a stuck at fault for the Lane id identified in Lane Repair id for the corresponding field. Injecting the fault at Tx or Rx is implementation specific. Default value is 0b.

Bit	Attribute	Description
14:8	RW	**Module 0 Lane repair id**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Lane repair for this logical transmit Lane id in logical Module 0 at the next Retrain cycle, if this Module is operational. For Standard Package, this bit will trigger a width degrade for logical Module 0, if possible and relevant. Default is 0. These bits are reserved if Module 0 is not present.
21:15	RW	**Module 1 Lane repair id**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Lane repair for this logical transmit Lane id in logical Module 1 at the next Retrain cycle, if this Module is operational. For Standard Package, this bit will trigger a width degrade for logical Module 1, if possible and relevant. Default is 0. These bits are reserved if Module 1 is not present.
28:22	RW	**Module 2 Lane repair id**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Lane repair for this logical transmit Lane id in logical Module 2 at the next Retrain cycle, if this Module is operational. For Standard Package, this bit will trigger a width degrade for logical Module 2, if possible and relevant. Default is 0. These bits are reserved if Module 2 is not present.
35:29	RW	**Module 3 Lane repair id**: For Advanced Package, software programs this bit to inform Physical Layer hardware to apply Lane repair for this logical transmit Lane id in logical Module 3 at the next Retrain cycle, if this Module is operational. For Standard Package, this bit will trigger a width degrade for logical Module 3, if possible and relevant. Default is 0. These bits are reserved if Module 3 is not present.
63:36	RsvdP	Reserved

7.5.3.33 Runtime Link Test Status (Offset 1108h)

Table 111. Runtime Link Test Status Register

Bit	Attribute	Description
0	RO	**Busy**: Hardware loads 1b to this bit once Start bit is written by software. Hardware loads 0b to this bit once it has attempted to complete the actions requested in Runtime Link Test Control register. Default is 0
31:1	RsvdZ	Reserved

7.5.3.34 Mainband Data Repair (Offset 110Ch)

This register is replicated per module. Offsets 110Ch to 112Ch are used in 8B offset increments for multi-module scenarios.

Table 112. Mainband Data Repair Register

Bit	Attribute	Description
7:0	RO	**Repair Address for TRD_P[0]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 00h: TD_P[0] Repaired 01h: TD_P[1] Repaired 02h: TD_P[2] Repaired …... 1Eh: TD_P[30] Repaired 1Fh: TD_P[31] Repaired F0h: Unrepairable
15:8	RO	**Repair Address for TRD_P[1]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 00h: Invalid 01h: TD_P[1] Repaired 02h: TD_P[2] Repaired …... 1Eh: TD_P[30] Repaired 1Fh: TD_P[31] Repaired F0h: Unrepairable
23:16	RO	**Repair Address for TRD_P[2]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 20h: TD_P[32] Repaired 21h: TD_P[33] Repaired 22h: TD_P[34] Repaired …... 3Eh: TD_P[62] Repaired 3Fh: TD_P[63] Repaired F0h: Unrepairable
31:24	RO	**Repair Address for TRD_P[3]**: Indicates the physical Lane repaired when TRD_P[0] is used in remapping scheme FFh: No Repair 20h: Invalid 21h: TD_P[33] Repaired 22h: TD_P[34] Repaired …... 3Eh: TD_P[62] Repaired 3Fh: TD_P[63] Repaired F0h: Unrepairable

Bit	Attribute	Description
39:32	RO	**Repair Address for RRD_P[0]**: Indicates the physical Lane repaired when RRD_P[0] is used in remapping scheme FFh: No Repair 00h: RD_P[0] Repaired 01h: RD_P[1] Repaired 02h: RD_P[2] Repaired …... 1Eh: RD_P[30] Repaired 1Fh: RD_P[31] Repaired F0h: Unrepairable
47:40	RO	**Repair Address for RRD_P[1]**: Indicates the physical Lane repaired when RRD_P[1] is used in remapping scheme FFh: No Repair 00h: Invalid 01h: RD_P[1] Repaired 02h: RD_P[2] Repaired …... 1Eh: RD_P[30] Repaired 1Fh: RD_P[31] Repaired F0h: Unrepairable
55:48	RO	**Repair Address for RRD_P[2]**: Indicates the physical Lane repaired when RRD_P[2] is used in remapping scheme FFh: No Repair 20h: RD_P[32] Repaired 21h: RD_P[33] Repaired 22h: RD_P[34] Repaired …... 3Eh: RD_P[62] Repaired 3Fh: RD_P[63] Repaired F0h: Unrepairable
63:56	RO	**Repair Address for RRD_P[3]**: Indicates the physical Lane repaired when RRD_P[3] is used in remapping scheme FFh: No Repair 20h: Invalid 21h: RD_P[33] Repaired 22h: RD_P[34] Repaired …... 3Eh: RD_P[62] Repaired 3Fh: RD_P[63] Repaired F0h: Unrepairable

7.5.3.35 Clock, Track, Valid and Sideband Repair (Offset 1134h)

This register is replicated per module. Offsets 1134h to 1140h are used in 4B offset increments for multi-module scenarios.

Table 113. Clock, Track, Valid and Sideband Repair Register

Bit	Attribute	Description
3:0	RO	**Repair Address for TRDCK_P**: Indicates the physical Lane repaired when TRDCK_P is used in remapping scheme Fh: No Repair 0h: TCKP_P Repaired 1h: TCKN_P Repaired 2h: TTRK_P Repaired 3h to 6h: Reserved 7h: Unrepairable
7:4	RO	**Repair Address for RRDCK_P**: Indicates the physical Lane repaired when RRDCK_P is used in remapping scheme Fh: No Repair 0h: RCKP_P Repaired 1h: RCKN_P Repaired 2h: RTRK_P Repaired 3h to 6h: Reserved 7h: Unrepairable
9:8	RO	**Repair Address for TRDVLD_P**: Indicates the physical Lane repaired when TRDVLD_P is used in remapping scheme 3h: No Repair 0h: TVLD_P Repaired 1h: Unrepairable 2h: Reserved
15:10	RsvdP	Reserved
19:16	RO	**Repair Address for Sideband Transmitter**: Indicates sideband repair result for the Transmitter Result[3:0]
23:20	RO	**Repair Address for Sideband Receiver**: Indicates sideband repair result for the Transmitter Result[3:0]
31:24	RsvdP	Reserved

7.5.4 Test/Compliance Register Block

These will be defined in the next rev of the spec.

7.5.5 Implementation Specific Register Blocks

These are left to be vendor defined. But these should carry the same header as defined in Table 79, at offset 0x0 of the register block. And the VendorID should be set to the specific vendor's ID and the 'VendorID register block' field set to 2h its a vendor specific register block. The other fields in that header are set by the vendor to track their revision number and the block length. Max length cannot exceed 1MB in size and length is always in multiples of 4KB. Implementations are highly encouraged to pack registers and reduce length of the region as much as possible.

7.6 MSI and MSI-X Capability in Hosts/Switches for UCIe interrupt

Follow the base spec for details, but MSI/MSI-X capability implemented in host and switch must request 2 vectors for UCIe usage - 1 for Link status events and 1 for Link error events. Note that in MSI scenario, OS might not always allot both the requested vectors and in that case both the Link Status and Link error events use the same MSI vector number. The MSI designs must also support the Pending and Mask bits.

8.0 Interface Definitions

This chapter will cover the details of interface operation and signal definitions for the Raw Die-to-Die Interface (RDI), as well as the Flit-Aware Die-to-Die Interface (FDI). Common rules across RDI and FDI are covered as a separate section. The convention used in this chapter is that "assertion" of a signal is for 0b to 1b transition, and "deassertion" of a signal is for 1b to 0b transition.

8.1 Raw Die-to-Die Interface (RDI)

This section defines the signal descriptions and functionality associated with a single instance of Raw Die-to-Die Interface (RDI). A single instance could be used for a configuration associated with a single Die-to-Die module (i.e. one Die-to-Die Adapter for one module), or a single instance is also applicable for configurations where multiple modules are grouped together for a single logical Die-to-Die Link (i.e. one Die-to-Die Adapter for multiple modules). Figure 108 shows example configurations using RDI.

Figure 108. Example configurations using RDI

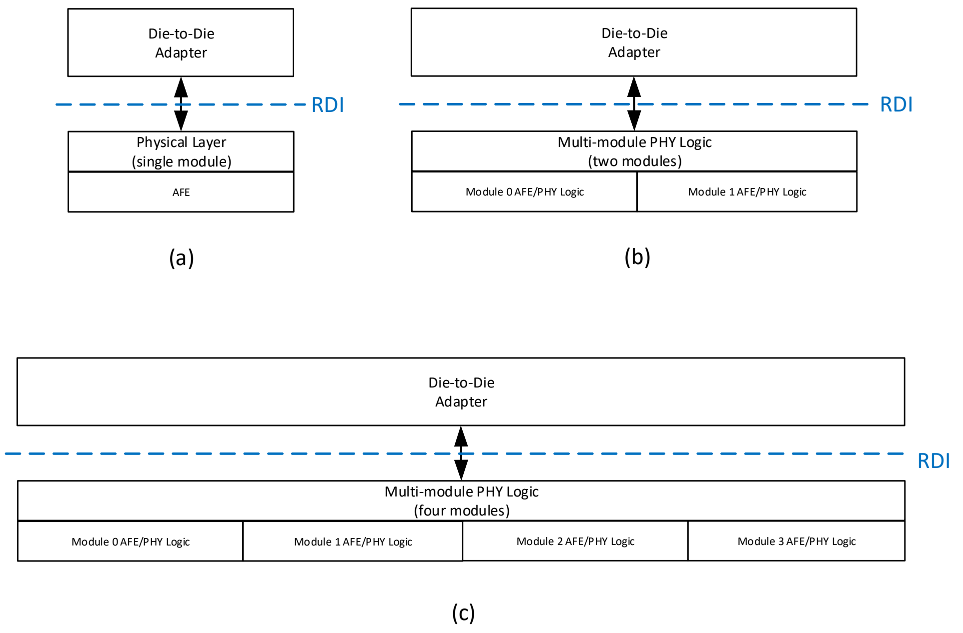


Table 114 lists the RDI signals and their descriptions. All signals are synchronous with **`lclk`**.

In Table 114:

- pl_* indicates that the signal is driven away from the Physical Layer to the Die-to-Die Adapter.
- lp_* indicates that the signal is driven away from the Die-to-Die Adapter to the Physical Layer.

Table 114. RDI signal list

Signal Name	Signal Description
lclk	The clock at which RDI operates. SOC must provide **lclk** to both the Physical Layer and the Die-to-Die Adapter.
lp_irdy	Adapter to Physical Layer signal indication that the Adapter has data to send. This must be asserted if lp_valid is asserted and the Protocol Layer wants the Adapter to sample the data. **lp_irdy** must not be presented by the Adapter when **pl_state_sts** is Reset except when the status transitions from LinkError to Reset. On a LinkError to Reset transition, it is permitted for **lp_irdy** to be asserted for a few clocks but it must be de-asserted eventually. Physical Layer must ignore **lp_irdy** when status is Reset.
lp_valid	Adapter to Physical Layer indication that data is valid on the corresponding **lp_data** bytes.
lp_data[NBYTES-1:0][7:0]	Adapter to Physical Layer data, where 'NBYTES' equals number of bytes determined by the data width for the RDI instance.
lp_retimer_crd	When asserted at a rising clock edge, it indicates a single credit return from the Adapter to the Physical Layer for the Retimer Receiver buffers. Each credit corresponds to 256B of mainband data. This signal must NOT assert for dies that are not UCIe Retimers.
pl_trdy	The Physical Layer is ready to accept data. Data is accepted by the Physical Layer when **pl_trdy**, **lp_valid**, and **lp_irdy** are asserted at the rising edge of **lclk**.
pl_valid	Physical Layer to Adapter indication that data is valid on **pl_data**.
pl_data[NBYTES-1:0][7:0]	Physical Layer to Adapter data, where NBYTES equals the number of bytes determined by the data width for the RDI instance.
pl_retimer_crd	When asserted at a rising clock edge, it indicates a single credit return from the Retimer to the Adapter. Each credit corresponds to 256B of mainband data. This signal must NOT assert if the remote Link partner is not a Retimer.
lp_state_req[3:0]	Adapter request to Physical Layer to request state change. Encodings as follows: 0000b : NOP 0001b : Active 0100b : L1 1000b : L2 1001b : LinkReset 1011b : Retrain 1100b : Disabled All other encodings are reserved.
lp_linkerror	Adapter to Physical Layer indication that an error has occurred which requires the Link to go down. Physical Layer must move to LinkError state and stay there as long as **lp_linkerror**=1. The reason for having this be an indication decoupled from regular state transitions is to allow immediate action on part of the Adapter and Physical Layer in order to provide the quickest path for error containment when applicable (for example, a viral error escalation must map to the LinkError state). The Adapter must OR internal error conditions with **lp_linkerror** received from Protocol Layer on FDI.

Signal Name	Signal Description
pl_state_sts[3:0]	Physical Layer to Adapter Status indication of the Interface. Encodings as follows: 0000b : Reset 0001b : Active 0011b : Active.PMNAK 0100b : L1 1000b : L2 1001b : LinkReset 1010b : LinkError 1011b : Retrain 1100b : Disabled All other encodings are reserved. The status signal is permitted to transition from Physical Layer autonomously when applicable. For example the Physical Layer asserts the Retrain status when it decides to enter retraining either autonomously or when requested by remote agent.
pl_inband_pres	Physical Layer to the Adapter indication that the Die-to-Die Link has finished training and is ready for RDI transition to Active and Stage 3 of bring up. Once it transitions to 1b, this must stay 1b until Physical Layer determines the Link is down (i.e. the Link Training State Machine transitions to TrainError or Reset).
pl_error	Physical Layer to the Adapter indication that it has detected a framing related error which is recoverable through Link Retrain. It is pipeline matched with the receive data path. Physical Layer is expected to go through Retrain flow after this signal has been asserted and it must not send valid data to Adapter until the Link has retrained. If **pl_error**=1 and **pl_valid**=1 in the same clock cycle, the Adapter must discard the corresponding Flit (even if it is only partially received when **pl_error** asserted) and trigger a retry (if retry is enabled).
pl_cerror	Physical Layer to the Adapter indication that a correctable error was detected that does not affect the data path and will not cause Retrain on the Link. The Adapter must OR the **pl_error** and **pl_cerror** signals for Correctable Error Logging.
pl_nferror	Physical Layer to the Adapter indication that a non-fatal error was detected.
pl_trainerror	Indicates a fatal error from the Physical Layer. Physical Layer must transition **pl_state_sts** to LinkError if not already in LinkError state. This must be escalated to upper Protocol Layers. Implementations are permitted to map any fatal error to this signal that require upper layer escalation (or interrupt generation) depending on system level requirements.
pl_phyinrecenter	Physical Layer indication to Adapter that the Physical Layer is training or retraining. If this is asserted during a state where clock gating is permitted, the **pl_clk_req/lp_clk_ack** handshake must be performed with the upper layer. The upper layers are permitted to use this to update the "Link Training/Retraining" bit in the UCIe Link Status register.
pl_stallreq	Physical Layer request to Adapter to align Transmitter at Flit boundary and not send any new Flits to prepare for state transition. Refer to section 8.3.1.

Signal Name	Signal Description
lp_stallack	Adapter to Physical Layer indication that the Flits are aligned and stalled (if **pl_stallreq** was asserted). It is strongly recommended that this response logic be on a global free running clock, so the Adapter can respond to **pl_stallreq** with **lp_stallack** even if other significant portions of the Adapter are clock gated. Refer to section 8.3.1.
pl_speedmode[2:0]	Current Link speed. The following encodings are used: 000b: 4GT/s 001b: 8GT/s 010b: 12GT/s 011b: 16GT/s 100b: 24GT/s 101b: 32GT/s other encodings are reserved. The Adapter must only consider this signal to be relevant when the RDI state is Active or Retrain. For multi-module configurations, all modules must operate at the same speed.
pl_lnk_cfg[2:0]	Current Link Configuration. Indicates the current operating width of a module. 001b: x8 010b: x16 011b: x32 100b: x64 101b: x128 110b: x256 other encodings are reserved. The Adapter must only consider this signal to be relevant when the RDI state is Active or Retrain. This signal indicates the total width across all Active Modules corresponding to the RDI instance.
pl_clk_req	Request from the Physical Layer to remove clock gating from the internal logic of the Adapter. This is an asynchronous signal relative to **lclk** from the Adapter's perspective since it is not tied to **lclk** being available in the Adapter. Together with **lp_clk_ack**, it forms a four-way handshake to enable dynamic clock gating in the Adapter. When dynamic clock gating is supported, the Adapter must use this signal to exit clock gating before responding with **lp_clk_ack**. If dynamic clock gating is not supported, it is permitted for the Physical Layer to tie this signal to 1b.
lp_clk_ack	Response from the Adapter to the Physical Layer acknowledging that its clocks have been ungated in response to **pl_clk_req**. This signal is only asserted when **pl_clk_req** is asserted, and de-asserted after **pl_clk_req** has de-asserted. When dynamic clock gating is not supported by the Adapter, it must stage **pl_clk_req** internally for one or more clock cycles and turn it around as **lp_clk_ack**. This way it will still participate in the handshake even though it does not support dynamic clock gating.

Signal Name	Signal Description
lp_wake_req	Request from the Adapter to remove clock gating from the internal logic of the Physical Layer. This is an asynchronous signal from the Physical Layer's perspective since it is not tied to **lclk** being available in the Physical Layer. Together with **pl_wake_ack**, it forms a four-way handshake to enable dynamic clock gating in the Physical Layer. When dynamic clock gating is supported, the Physical Layer must use this signal to exit clock gating before responding with **pl_wake_ack**. If dynamic clock gating is not supported, it is permitted for the Adapter to tie this signal to 1b.
pl_wake_ack	Response from the Physical Layer to the Adapter acknowledging that its clocks have been ungated in response to **lp_wake_req**. This signal is only asserted after **lp_wake_req** has asserted, and is de-asserted after **lp_wake_req** has de-asserted. When dynamic clock gating is not supported by the Physical Layer, it must stage **lp_wake_req** internally for one or more clock cycles and turn it around as **pl_wake_ack**. This way it will still participate in the handshake even though it does not support dynamic clock gating.
pl_cfg[NC-1:0]	This is the sideband interface from the Physical Layer to the Adapter. See Chapter 6.0 for packet format details. NC is the width of the interface. Supported values are 8, 16, and 32.
pl_cfg_vld	When asserted, indicates that **pl_cfg** has valid information that should be consumed by the Adapter.
pl_cfg_crd	Credit return for sideband packets from the Physical Layer to the Adapter for sideband packets. Each credit corresponds to 64-bit of header and 64-bit of data. Even transactions that don't carry data or carry 32-bit of data consume the same credit and the Physical Layer returns the credit once the corresponding transaction has been processed or de-allocated from its internal buffers. Refer to section 6.1.3.1 for additional flow control rules.
lp_cfg[NC-1:0]	This is the sideband interface from Adapter to the Physical Layer. See Chapter 6.0 for details. NC is the width of the interface. Supported values are 8, 16, and 32.
lp_cfg_vld	When asserted, indicates that **lp_cfg** has valid information that should be consumed by the Physical Layer.
lp_cfg_crd	Credit return for sideband packets from the Adapter to the Physical Layer for sideband packets. Each credit corresponds to 64-bit of header and 64-bit of data. Even transactions that don't carry data or carry 32-bit of data consume the same credit and the Adapter returns the credit once the corresponding transaction has been processed or de-allocated from its internal buffers. Refer to section 6.1.3.1 for additional flow control rules.

8.1.1 Interface reset requirements

RDI does not define a separate interface signal for reset; however, it is required that the logic entities on both sides of RDI are in the same reset domain and the reset for each side is derived from the same source.

8.1.2 Interface clocking requirements

RDI requires both sides of the interface to be on the same clock domain. Moreover, the clock domain for sideband interface (`*cfg*`) is the same as the mainband signals.

Each side is permitted to instantiate clock crossing FIFOs internally if needed, as long as it does not violate the requirements at the interface itself.

It is important to note that there is no back pressure possible from the Adapter to the Physical Layer on the main data path. So any clock crossing related logic internal to the Adapter must take this into consideration.

For example, for a 64 Lane module with a max speed of 16GT/s, RDI could be 64B wide running at 2GHz to be exactly bandwidth matched.

8.1.3 Dynamic clock gating

Dynamic coarse clock gating is permitted in the Adapter and Physical Layer when `pl_state_sts` is Reset, LinkReset, Disabled, or PM. This section defines the rules around entry and exit of clock gating. Note that clock gating is not permitted in LinkError state; it is expected that for UCIe usages, error handlers will be enabled to make sure the Link is not stuck in LinkError state if the intent is save power for Links in error state.

8.1.3.1 Rules and description for lp_wake_req/pl_wake_ack handshake

Adapter can request removal of clock gating of the Physical Layer by asserting `lp_wake_req` (asynchronous to `lclk` availability in the Physical Layer). All Physical Layer implementations must respond with a `pl_wake_ack` (synchronous to `lclk`). The extent of internal clock ungating when `pl_wake_ack` is asserted is implementation-specific, but lclk must be available by this time to enable RDI interface transitions from the Adapters. The Wake Req/Ack is a full handshake and it must be used for state transition requests (on `lp_state_req` or `lp_linkerror`) when moving away from Reset or PM states. It must also be used for sending packets on the sideband interface.

Rules for this handshake:

1. Adapter asserts `lp_wake_req` to request ungating of clocks by the Physical Layer.
2. The Physical Layer asserts `pl_wake_ack` to indicate that clock gating has been removed. There must be at least one clock cycle bubble between `lp_wake_req` assertion and `pl_wake_ack` assertion.
3. `lp_wake_req` must de-assert before `pl_wake_ack` de-asserts. It is the responsibility of the Adapter to control the specific scenario of de-assertion. As an example, when performing the handshake for a state request, it is permitted to keep `lp_wake_req` asserted until it observes the desired state status. Adapter is also permitted to keep `lp_wake_req` asserted through states where clock gating is not allowed in the Physical Layer (i.e. Active, LinkError or Retrain).
4. `lp_wake_req` should not be the only consideration for Physical Layer to perform clock gating, it must take into account `pl_state_sts` and other internal or Link requirements before performing global and/or local clock gating.
5. When performing `lp_wake_req`/`pl_wake_ack` handshake for `lp_state_req` transitions or `lp_linkerror` transition, the Adapter is permitted to not wait for `pl_wake_ack` before changing `lp_state_req or lp_linkerror`.
6. When performing `lp_wake_req`/`pl_wake_ack` handshake for `lp_cfg` transitions, Adapter must wait for `pl_wake_ack` before changing `lp_cfg` or `lp_cfg_vld`.

Because `lp_cfg` can have multiple transitions for a single packet transfer, it is necessary to make sure that the Physical Layer clocks are up before transfer begins.

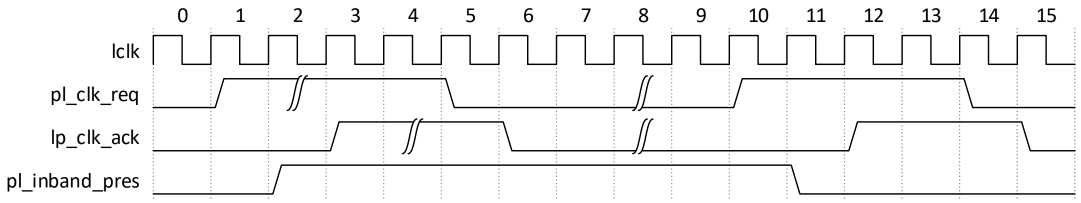
8.1.3.2 Rules and description for pl_clk_req/lp_clk_ack handshake

Physical Layer is allowed to initiate `pl_clk_req/lp_clk_ack` handshake at any time and the Adapter must respond.

Rules for this handshake:

1. Physical Layer asserts `pl_clk_req` to request removal of clock gating by the Adapter. This can be done anytime, and independent of current RDI state.
2. The Adapter asserts `lp_clk_ack` to indicate that clock gating has been removed. There must be at least one clock cycle bubble between `pl_clk_req` assertion and `lp_clk_ack` assertion.
3. `pl_clk_req` must de-assert before `lp_clk_ack`. It is the responsibility of the Physical Layer to control the specific scenario of de-assertion, after the required actions for this handshake are completed.
4. `pl_clk_req` should not be the only consideration for the Adapter to perform clock gating, it must take into account `pl_state_sts` and other protocol-specific requirements before performing trunk and/or local clock gating.
5. The Physical Layer must use this handshake to ensure transitions of `pl_inband_pres` have been observed by the Adapter. Since `pl_inband_pres` is a level oriented signal (once asserted it stays asserted during the lifetime of Link operation), the Physical Layer is permitted to let the signal transition without waiting for `lp_clk_ack`. When this is done during initial Link bring up, it is strongly recommended for the Physical Layer to keep `pl_clk_req` asserted until the state status transitions away from Reset to a state where clock gating is not permitted.

Figure 109. Example Waveform Showing Handling of Level Transition



6. The Physical Layer must also perform this handshake before transition to LinkError state from Reset or PM state (when the LinkError transition occurs by the Physical Layer without being directed by the Adapter). It is permitted to assert `pl_clk_req` before the state change, in which case it must stay asserted until the state status transitions. It is also permitted to assert `pl_clk_req` after the state status transition, but in this case Physical Layer must wait for `lp_clk_ack` before performing another state transition.
7. The Physical Layer must also perform this handshake when the status is PM and remote Link partner is requesting PM exit. For exit from Reset or PM states to a state that is not LinkError, it is required to assert pl_clk_req before the status change, and in this case it must stay asserted until the state status transitions away from Reset or PM.
8. When clock-gated in RESET states, Adapters that rely on dynamic clock gating to save power must wait in clock gated state for `pl_inband_pres`=1. The Physical Layer will request clock gating exit when it transitions `pl_inband_pres`, and the Adapter must wait for `pl_inband_pres` assertion before requesting `lp_state_req` = ACTIVE. If `pl_inband_pres` de-asserts while `pl_state_sts` = RESET, then the

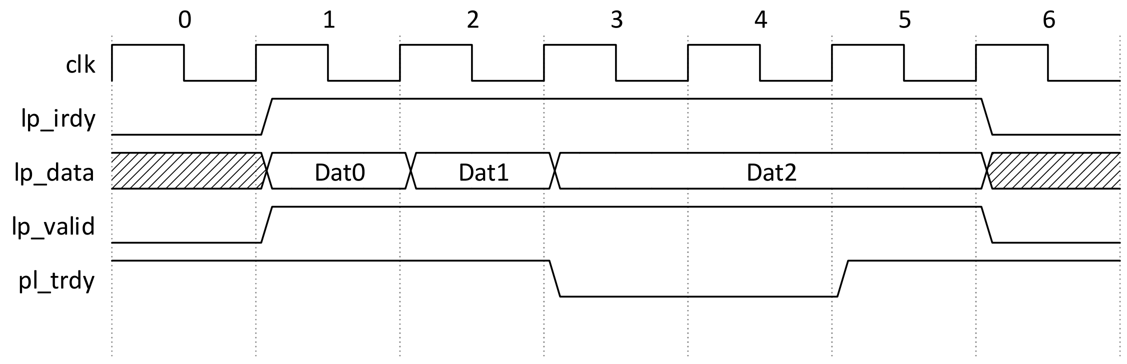
Adapter is permitted to return to clock-gated state after moving **`lp_state_req`** to NOP.

9. Physical Layer must also perform this handshake for sideband traffic to Adapter. When performing the handshake for **`pl_cfg`** transitions, Physical Layer must wait for **`lp_clk_ack`** before changing **`pl_cfg`** or **`pl_cfg_vld`**. Because **`pl_cfg`** can have multiple transitions for a single packet transfer, it is necessary to make sure that the Adapter clocks are up before transfer begins.

8.1.4 Data Transfer

As indicated in the signal list descriptions, when Adapter is sending data to the Physical Layer, data is transferred when **`lp_irdy`**, **`pl_trdy`** and **`lp_valid`** are asserted. Figure 110 shows an example waveform for data transfer from the Adapter to the Physical Layer. Data is transmitted on clock cycles 1, 2, and 5. No assumption should be made by Adapter about when **`pl_trdy`** can de-assert or for how many cycles it remains de-asserted before it is asserted again, unless explicitly guaranteed by the Physical Layer. If a Flit transfer takes multiple clock cycles, the Adapter is not permitted to insert bubbles in the middle of a Flit transfer (i.e. **`lp_valid`** and **`lp_irdy`** must be asserted continuously until the Flit transfer is complete. Of course, data transfer can stall because of **`pl_trdy`** de-assertion).

Figure 110. Data Transfer from Adapter to Physical Layer

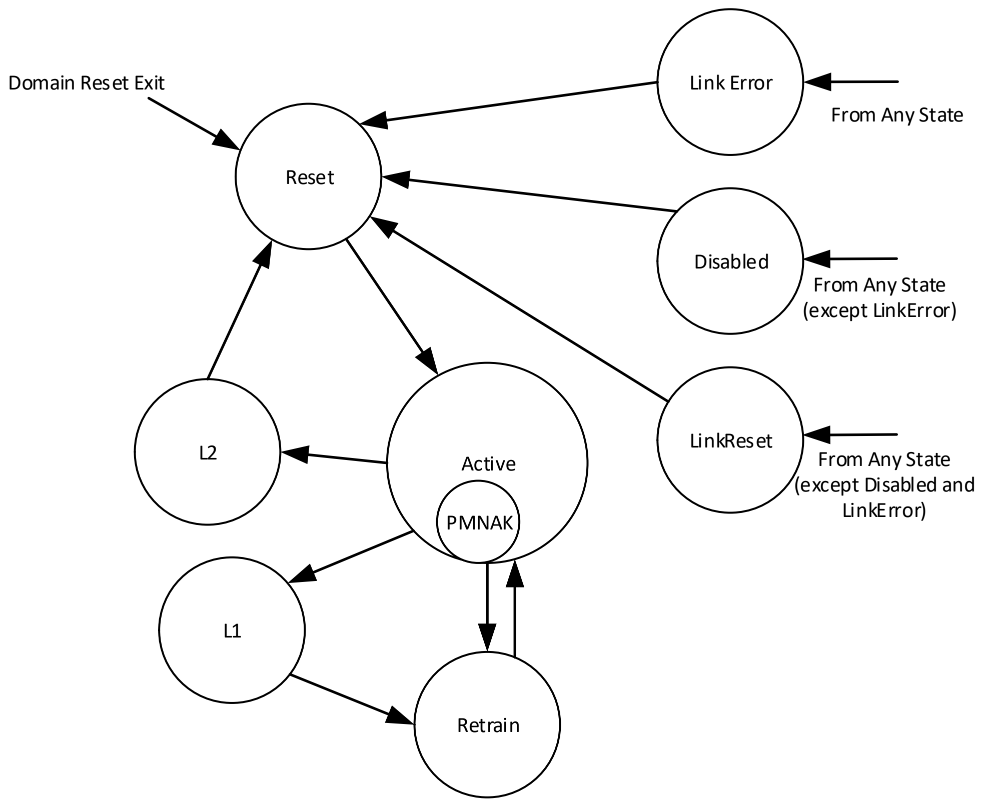


As indicated in the signal list descriptions, when Physical Layer is sending data to the Adapter, there is no backpressure mechanism, and data is transferred whenever **`pl_valid`** is asserted. The Physical Layer is permitted to insert bubbles in the middle of a Flit transfer and the Adapter must be able to handle that.

8.1.5 RDI State Status Machine

Figure 111 shows the RDI interface state machine.

Figure 111. RDI Interface State Machine



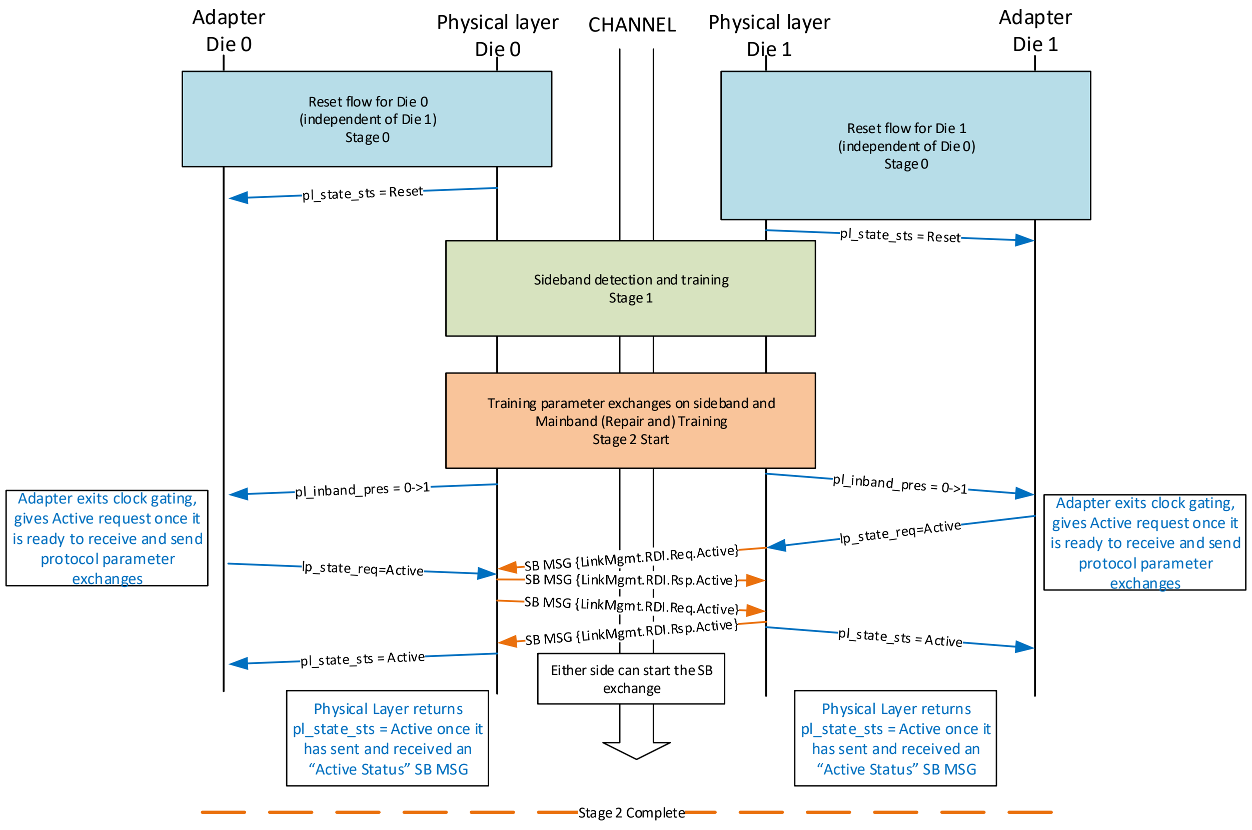
8.1.6 RDI bring up flow

Figure 112 shows an example flow for Stage 2 of the Link bring up highlighting the transitions on RDI. This stage requires sequencing on RDI that co-ordinates the state transition from Reset to Active.

1. Once Physical Layer has completed Link training, it must do the **`pl_clk_req`** handshake with the Adapter and reflect **`pl_inband_pres`**=1 on RDI. Note that the **`pl_clk_req`** handshake is not shown in the example flow in Figure 112
2. This is the trigger for Adapter to request Active state. It must perform the **`lp_wake_req`** handshake as described in section 8.1.3. Note that the **`lp_wake_req`** handshake is not shown in the example flow in Figure 112.
3. Only after sampling **`lp_state_req`** = Active, the Physical Layer must send the {LinkMgmt.RDI.Req.Active} packet on sideband to remote Link partner's Physical Layer.
4. The Physical Layer must respond to the {LinkMgmt.RDI.Rsp.Active} sideband message with an {LinkMgmt.RDI.Rsp.Active} message on sideband. The {LinkMgmt.RDI.Rsp.Active} must only be sent after the Physical Layer has sampled **`lp_state_req`** = Active from its local RDI interface.
5. Once the Physical Layer has sent and received the {LinkMgmt.RDI.Rsp.Active} sideband message, it must transition **`pl_state_sts`** to Active.
6. This opens up the Adapter to transition to Stage 3 of the bring up flow.

Steps 3 to 5 are referred to as the "Active Entry handshake" and must be done before transition to Active from Reset or Retrain states.

Figure 112. Example flow of Link bring up on RDI


Adapter Die 0
Physical layer Die 0
CHANNEL
Physical layer Die 1
Adapter Die 1
Reset flow for Die 0 (independent of Die 1) Stage 0
Reset flow for Die 1 (independent of Die 0) Stage 0
pl_state_sts = Reset
pl_state_sts = Reset
Sideband detection and training Stage 1
Training parameter exchanges on sideband and Mainband (Repair and) Training Stage 2 Start
pl_inband_pres = 0->1
pl_inband_pres = 0->1
Adapter exits clock gating, gives Active request once it is ready to receive and send protocol parameter exchanges
Adapter exits clock gating, gives Active request once it is ready to receive and send protocol parameter exchanges
lp_state_req=Active
lp_state_req=Active
SB MSG {LinkMgmt.RDI.Req.Active}
SB MSG {LinkMgmt.RDI.Rsp.Active}
SB MSG {LinkMgmt.RDI.Req.Active}
SB MSG {LinkMgmt.RDI.Rsp.Active}
pl_state_sts = Active
pl_state_sts = Active
Either side can start the SB exchange
Physical Layer returns pl_state_sts = Active once it has sent and received an "Active Status" SB MSG
Physical Layer returns pl_state_sts = Active once it has sent and received an "Active Status" SB MSG
Stage 2 Complete

8.1.7 RDI PM flow

This section defines the rules for PM entry, exit and abort flows as they apply to handshakes on the RDI interface. The rules for L1 and L2 are the same, except that exit from L2 is to Reset state, whereas exit from L1 is to Retrain state. This section uses PM to denote L1 or L2. A "PM request" sideband message is {LinkMgmt.RDI.Req.L1} or {LinkMgmt.RDI.Req.L2}. A "PM response" sideband message is {LinkMgmt.RDI.Rsp.L1} or {LinkMgmt.RDI.Rsp.L2}.

- Regardless of protocol, the PM entry or exit flow is symmetric on RDI. Both Physical Layer must issue PM entry request through a sideband message once the conditions of PM entry have been satisfied. PM entry is considered successful and complete once both sides have received a valid "PM Response" sideband message. Figure 113 shows an example flow for L1. Once the RDI status is PM, the Physical Layer can transition itself to a power savings state (turning off the PLL for example). Note that the sideband logic and corresponding PLL needs to stay on even during L1 state.
- All the Adapter state machines (Adapter LSMs) in the Adapter must have moved to the corresponding PM state before the Adapter requests PM entry from remote Link partner. Adapter LSM in PM implies the retry buffer of the Adapter must be empty, and it must not have any new Flits (or Ack/Nak) pending to be scheduled.

Essentially there should be no traffic on mainband when PM entry is requested by the Adapter to the Physical Layer. The Adapter is permitted to clock gate its sideband logic once RDI status is PM and there are no outstanding transactions or responses on sideband. Physical Layer must do **`pl_clk_req`** handshake (if **`pl_clk_req`** is not already asserted or status is not Active) before forwarding sideband requests from the Link to the Adapter.

- Adapter requests PM entry by transitioning **`lp_state_req`** to the corresponding PM encoding. Once requested, the Adapter cannot change this request until it observes PM, Active.PMNAK or LinkError state on **`pl_state_sts`**. While requesting PM state, if the Adapter receives Active request from the Protocol Layer, or a PM exit request for the Adapter LSM on sideband, it must sink the message but delay processing it until **`pl_state_sts`** has resolved. Once the RDI state is resolved, the Adapter must first bring it back to Active before processing the other requests.
  - If the resolution is PM (upon successful PM entry) and the Protocol Layer needs to exit PM (or there is a pending Protocol Layer Active request from remote Link partner) then the Adapter must initiate PM exit flow on RDI by requesting **`lp_state_req`** = Active. All PM entry related handshakes must have finished prior to this (This is when both sides Physical Layer have received a valid "PM Response" sideband message).
  - If the resolution is Active.PMNAK, the Adapter must initiate a request of Active on RDI. Once the status moves to Active, it must wait for at least 2 us before re-requesting PM entry (if all conditions of PM entry are still met). Figure 114 shows an example of PM abort flow. The PM request could have been from either side.
  - If the resolution is LinkError, then the Adapter must propagate this to Protocol Layers. This also resets any outstanding PM handshakes.
- Physical Layer initiates a sideband "PM request" once it samples the corresponding PM encoding on **`lp_state_req`**.
- Once a Physical Layer receives a sideband "PM request", it must respond to it within 2 us:
  - if its local Adapter is requesting the corresponding PM state , it must respond with the corresponding "PM response" sideband message. If the current status is not PM, it must transition **`pl_state_sts`** to PM after responding to the sideband message.
  - If the current **`pl_state_sts`** = PM, it must respond with "PM Response" sideband message.
  - If **`pl_state_sts`** = Active and **`lp_state_req`** = Active and it stays this way for 1us after receiving the "PM Request" message, it must respond with {LinkMgmt.RDI.Rsp.PMNAK} sideband message.
- If a Physical Layer receives a sideband "PM response" message in response to a "PM Request" message, it must transition **`pl_state_sts`** on its local RDI to PM (if it is not currently in a PM state).
- If a Physical Layer receives a sideband {LinkMgmt.RDI.Rsp.PMNAK} message in response to a "PM Request" message, it must transition **`pl_state_sts`** on its local RDI to Active.PMNAK state. It is permitted to retry PM entry handshake (if all conditions of PM entry are satisfied) at least 2us after receiving the {LinkMgmt.RDI.Rsp.PMNAK} message.
- PM exit is initiated by the Adapter requesting Active on RDI. This triggers the Physical Layer to initiate PM exit by sending an {LinkMgmt.RDI.Req.Active} message on sideband. Physical Layer must make sure it has finished any Link retraining steps before it responds with the {LinkMgmt.RDI.Rsp.Active} sideband message. Figure 115 shows an example flow of PM exit on RDI.
  - PM exit handshake completion requires both Physical Layers to send as well as receive a {LinkMgmt.RDI.Rsp.Active} message on sideband. Once this has

completed, the Physical Layer is permitted to transition **`pl_state_sts`** to Active on RDI.

— If **`pl_state_sts`** = PM and an {LinkMgmt.RDI.Req.Active} message is received, the Physical Layer must initiate **`pl_clk_req`** handshake with the Adapter, and transition **`pl_state_sts`** to Retrain (For L2 exit, the transition is to Reset). This must trigger the Adapter to request Active on **`lp_state_req`** (if not already doing so), and this in turn triggers the Physical Layer to send {LinkMgmt.RDI.Req.Active} sideband message to the remote Link partner.

Note that the figures below are examples for L1, and they do not show the **`lp_wake_req`**, **`pl_clk_req`** handshakes. Implementations must follow the rules outlined for these handshakes in previous sections.

Figure 113. Successful PM entry flow

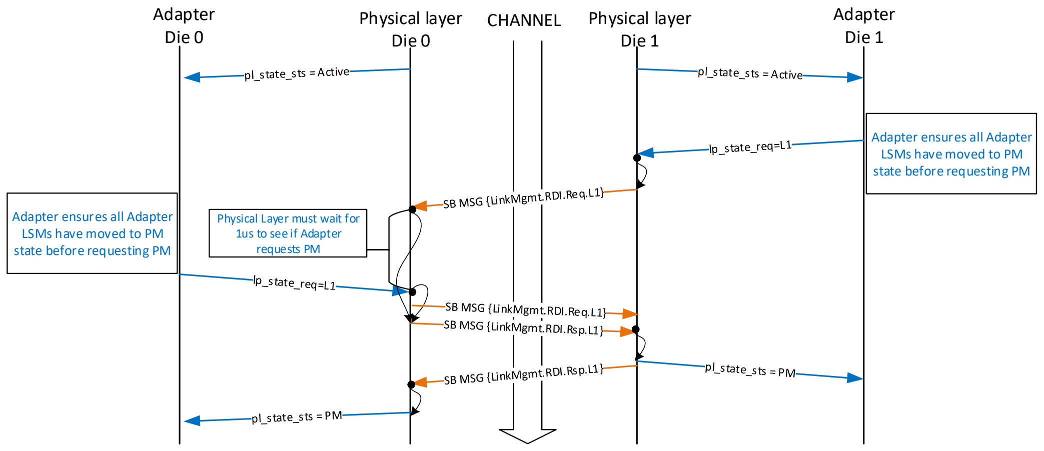

Adapter Die 0
Physical layer Die 0
CHANNEL
Physical layer Die 1
Adapter Die 1
pl_state_sts = Active
pl_state_sts = Active
Adapter ensures all Adapter LSMs have moved to PM state before requesting PM
lp_state_req=L1
SB MSG {LinkMgmt.RDI.Req.L1}
Adapter ensures all Adapter LSMs have moved to PM state before requesting PM
Physical Layer must wait for 1us to see if Adapter requests PM
lp_state_req=L1
SB MSG {LinkMgmt.RDI.Req.L1}
SB MSG {LinkMgmt.RDI.Rsp.L1}
pl_state_sts = PM
SB MSG {LinkMgmt.RDI.Rsp.L1}
pl_state_sts = PM

Figure 114. PM Abort flow

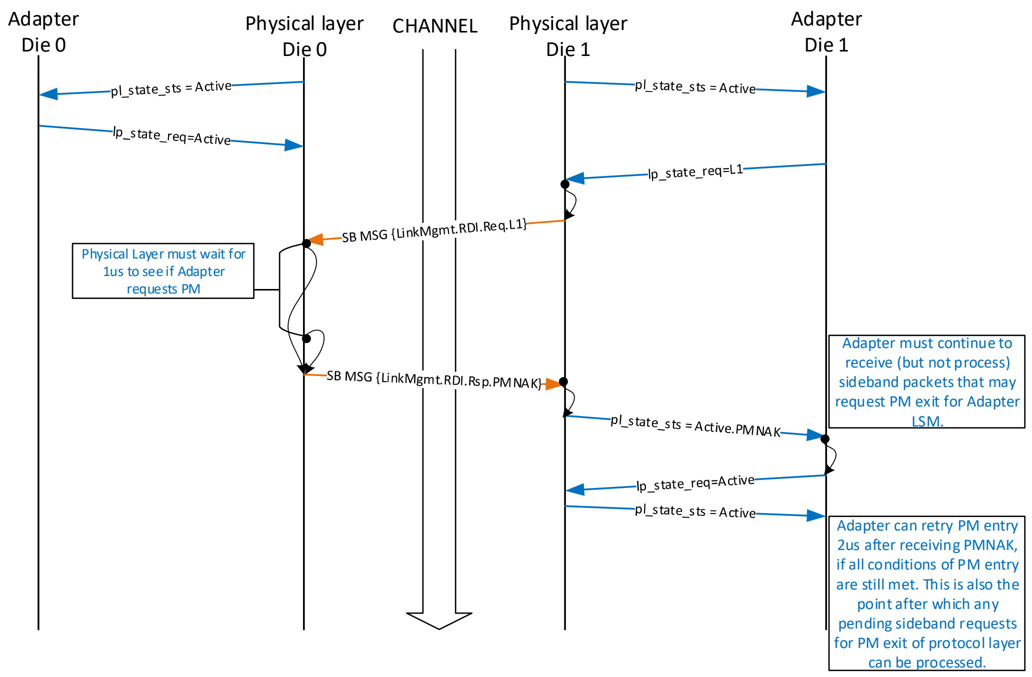
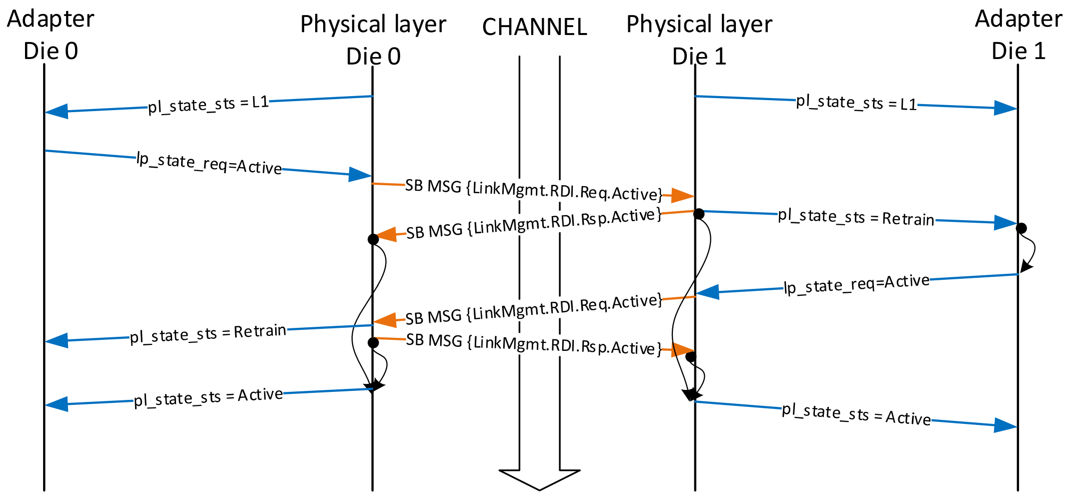

Adapter
Die 0
Physical layer
Die 0
CHANNEL
Physical layer
Die 1
Adapter
Die 1
pl_state_sts = Active
pl_state_sts = Active
lp_state_req=Active
lp_state_req=L1
SB MSG {LinkMgmt.RDI.Req.L1}
Physical Layer must wait for 1us to see if Adapter requests PM
Adapter must continue to receive (but not process) sideband packets that may request PM exit for Adapter LSM.
SB MSG {LinkMgmt.RDI.Rsp.PMNAK}
pl_state_sts = Active.PMNAK
lp_state_req=Active
pl_state_sts = Active
Adapter can retry PM entry 2us after receiving PMNAK, if all conditions of PM entry are still met. This is also the point after which any pending sideband requests for PM exit of protocol layer can be processed.

Figure 115. PM Exit flow



8.2 Flit-Aware Die-to-Die Interface (FDI)

This section defines the signal descriptions and functionality associated with a single instance of Flit-Aware Die-to-Die Interface (FDI). A single instance is used for a Protocol Layer to Adapter connection. However, a single Adapter can host multiple protocol stacks using multiple instances of FDI. Figure 116 shows example configurations using multiple instances of FDI.

Figure 116. Example configurations using FDI

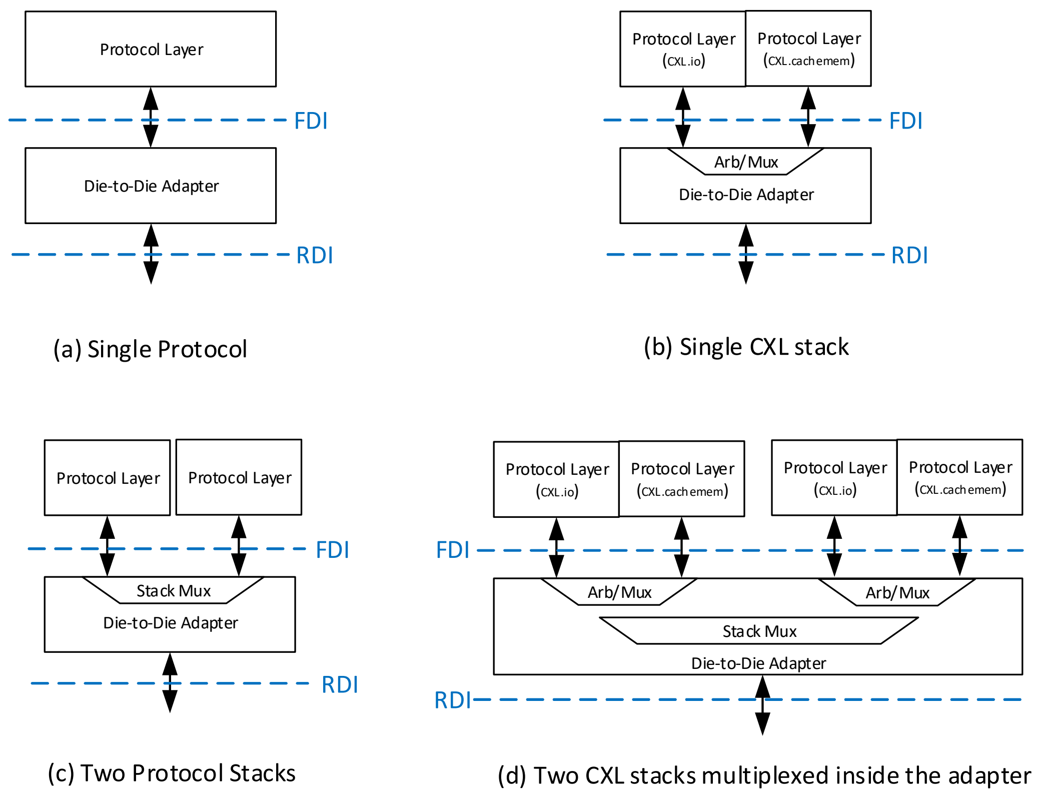


Table 115 lists the FDI signals and their descriptions. All signals are synchronous with **`lclk`**.

In Table 115:

- pl_* indicates that the signal is driven away from the Die-to-Die Adapter to the Protocol Layer.
- lp_* indicates that the signal is driven away from the Protocol Layer to the Die-to-Die Adapter.

Note: The same signal naming convention as RDI is used to highlight that RDI signal list is a proper subset of FDI signal list.

Table 115. FDI signal list

Signal Name	Signal Description
lclk	The clock at which FDI operates. SOC must provide **lclk** to both the Protocol Layer and the Die-to-Die Adapter.
lp_irdy	Signal indicating that the Protocol Layer potentially has data to send. This must be asserted if lp_valid is asserted and the Protocol Layer wants the Adapter to sample the data. **lp_irdy** must not be presented by the Protocol Layer when **pl_state_sts** is Reset except when the status transitions from LinkError to Reset. On a LinkError to Reset transition, it is permitted for **lp_irdy** to be asserted for a few clocks but it must be de-asserted eventually. Physical Layer must ignore **lp_irdy** when status is Reset.
lp_valid	Protocol Layer to Adapter indication that data is valid on the corresponding **lp_data** bytes.
lp_data[NBYTES-1:0][7:0]	Protocol Layer to Adapter data, where 'NBYTES' equals number of bytes determined by the data width for the FDI instance.
lp_nop_flit	Protocol Layer to Adapter indication that the Flit is NOP and is permitted to bypass the TX Retry buffer. This signal must only be asserted for the last chunk transfer of a Flit, and is only applicable for 256B Flit formats for PCIe or CXL protocols. The Protocol Layer must insert the appropriate Protocol Identifier in the Flit Header bytes.
lp_retimer_crd	When asserted at a rising clock edge, it indicates a single credit return for the Retimer Receiver buffer. Each credit corresponds to 256B of mainband data (including Flit header and CRC etc.). This signal must NOT assert if a Retimer is not present. On FDI, this is an optional signal. It is permitted to have the Receiver buffers in the Protocol Layer for Raw Mode only. If this is not exposed to Protocol Layer, Adapter must track credit at 256B granularity even for Raw Mode and return credits to Physical Layer on RDI. When this is exposed on FDI, the Adapter must have the initial credits knowledge through other implementation specific means in order to advertise this to the remote Link partner during parameter exchanges.
lp_corrupt_crc	This signal is only applicable for CXL.cachemem in UCIe Flit Mode (i.e. the Adapter doing Retry) for CXL 256B Flit Mode. It is meant as a latency optimization that enables detection and containment for viral or poison using the Adapter to corrupt CRC of outgoing Flit. For Standard 256B Flits, Protocol Layer asserts this along with **lp_valid** for the last chunk of the Flit that needs containment. Adapter corrupts CRC for both of the 128B halves of the Flit which had this set. It also must make sure to overwrite this flit (with the next flit sent by the Protocol Layer) in the Tx Retry buffer. For Latency Optimized 256B Flits, Protocol Layer asserts this along with **lp_valid** for the last chunk of the 128B Flit half that needs containment. If **lp_corrupt_crc** is asserted on the first 128B half of the Flit, Protocol Layer must assert it on the second 128B half of the Flit as well. The very next Flit from the Protocol Layer after this signal has been asserted must carry the information relevant for viral, as defined in the CXL specification. If this was asserted on the second 128B half of the Flit only, it is the responsibility of the Protocol Layer to send the first 128B half exactly as before, and insert the viral information in the second half of the Flit. Adapter corrupts CRC for the 128B half of the Flit which had this set. It also must make sure to overwrite this flit (with the next flit sent by the Protocol Layer) in the Tx Retry buffer.

Signal Name	Signal Description
lp_dllp[NDLLP-1:0]	Protocol Layer to Adapter transfer of DLLP bytes. This is not used for 68B Flit Mode, CXL.cachemem or Streaming Protocols. For a 64B data path on lp_data, it is recommended to assign NDLLP >= 8, so that 1 DLLP per Flit can be transferred from the Protocol Layer to the Adapter on average. The Adapter is responsible for inserting DLLP into DLP bytes 2:5 if the Flit packing rules permit it. Refer to section 8.2.4.1 for additional rules.
lp_dllp_valid	Indicates valid DLLP transfer on **lp_dllp**. DLLP transfers are not subject to backpressure by **pl_trdy** (the Adapter must have storage for different types of DLLP and this can be overwritten so that the latest DLLPs are sent to remote Link partner). DLLP transfers are subject to backpressure by **pl_stallreq** - Protocol Layer must stop DLLP transfers at DLLP Flit aligned boundary before giving **lp_stallack** or requesting PM.
lp_dllp_ofc	Indicates that the corresponding DLLP bytes on **lp_dllp** follow the Optimized_Update_FC format. It must stay asserted for the entire duration of the DLLP transfer on **lp_dllp**.
lp_stream[7:0]	Protocol Layer to Adapter signal that indicates the stream ID to use with data. Each stream ID maps to a unique protocol and stack. 00h : Reserved 01h : Stack 0 : PCIe 02h : Stack 0 : CXL.io 03h : Stack 0 : CXL.cachemem 04h : Stack 0 : Streaming protocol 11h : Stack 1 : PCIe 12h : Stack 1 : CXL.io 13h : Stack 1 : CXL.cachemem 14h : Stack 1 : Streaming protocol Other encodings are Reserved.
pl_trdy	The Adapter is ready to accept data. Data is accepted by the Adapter when **pl_trdy**, **lp_valid**, and **lp_irdy** are asserted at the rising edge of **lclk**.
pl_valid	Adapter to Protocol Layer indication that data is valid on **pl_data**.
pl_data[NBYTES-1:0][7:0]	Adapter to Protocol Layer data, where NBYTES equals the number of bytes determined by the data width for the FDI instance.
pl_retimer_crd	When asserted at a rising clock edge, it indicates a single credit return from the Retimer. Each credit corresponds to 256B of mainband data (including Flit header and CRC etc.). This signal must NOT assert if a Retimer is not present. On FDI, this is an optional signal. It is permitted to expose these credits to Protocol Layer for Raw Mode only. If this is not exposed to Protocol Layer, Adapter must track credit at 256B granularity even for Raw Mode and back-pressure the Protocol Layer using **pl_trdy**. When this is exposed on FDI, the Adapter converts the initial credits received from the Retimer over sideband to credit returns to the Protocol Layer on this bit after Adapter LSM has moved to Active state.

Signal Name	Signal Description
pl_dllp[NDLLP-1:0]	Adapter to Protocol Layer transfer of DLLP bytes. This is not used for 68B Flit Mode, CXL.cachemem or Streaming Protocols. For a 64B data path on **pl_data**, it is recommended to assign NDLLP >= 8, so that 1 DLLP per Flit can be transferred from the Adapter to the Protocol Layer, on average. The Adapter is responsible for extracting DLLP from DLP bytes 2:5 if a Flit Marker is not present. The Adapter is also responsible for indicating Optimized_Update_FC format by setting **pl_dllp_ofc** = 1 for the corresponding transfer on FDI.
pl_dllp_valid	Indicates valid DLLP transfer on **pl_dllp**. DLLPs can be transferred to the Protocol Layer whenever valid Flits can be transferred on **pl_data**. There is no backpressure and the Protocol Layer must always sink DLLPs.
pl_dllp_ofc	Indicates that the corresponding DLLP bytes on **pl_dllp** follow the Optimized_Update_FC format. It must stay asserted for the entire duration of the DLLP transfer on **pl_dllp**.
pl_stream[7:0]	Adapter to Protocol Layer signal that indicates the stream ID to use with data. Each stream ID maps to a unique protocol. 00h : Reserved 01h : Stack 0 : PCIe 02h : Stack 0 : CXL.io 03h : Stack 0 : CXL.cachemem 04h : Stack 0 : Streaming protocol 11h : Stack 1 : PCIe 12h : Stack 1 : CXL.io 13h : Stack 1 : CXL.cachemem 14h : Stack 1 : Streaming protocol Other encodings are Reserved.
pl_flit_cancel	Adapter to Protocol Layer indication to dump a Flit. This enables latency optimizations on the Receiver data path when CRC checking is enabled in the Adapter. It is not applicable for Raw Mode. For Standard 256B Flit, it is required to have a fixed number of clock cycle delay between the last chunk of a Flit transfer and the assertion of **pl_flit_cancel**. This delay is fixed to be 1 cycle; i.e., the cycle after the last chunk transfer of a Flit. For Latency Optimized 256B Flits, it is required to have a fixed number of clock cycle delay between the last chunk of a 128B half Flit transfer and the assertion of **pl_flit_cancel**. This delay is fixed to be 1 cycle; i.e., the cycle after the last transfer of the corresponding 128B chunk. When this signal is asserted, Protocol Layer must not consume the associated Flit. When this mode is supported, Protocol Layer must support it for all Flit formats associated with the corresponding protocol. Adapter must guarantee this to be a single cycle pulse when dumping a Flit. It is the responsibility of the Adapter to make sure that the canceled Flits or Flit halves are eventually replayed on the interface in the correct order.

Signal Name	Signal Description
lp_state_req[3:0]	Protocol Layer request to Adapter to request state change. Encodings as follows: 0000b : NOP 0001b : Active 0100b : L1 1000b : L2 1001b : LinkReset 1011b : Retrain 1100b : Disabled All other encodings are reserved.
lp_linkerror	Protocol Layer to Adapter indication that an error has occurred which requires the Link to go down. Adapter must propagate this request to RDI, and move the Adapter LSMs (and CXL vLSMs if applicable) to LinkError state once RDI is in LinkError state. It must stay there as long as **lp_linkerror**=1.The reason for having this be an indication decoupled from regular state transitions is to allow immediate action on part of the Protocol Layer and Adapter in order to provide the quickest path for error containment when applicable (for example, a viral error escalation could map to the LinkError state)
pl_state_sts[3:0]	Adapter to Protocol Layer Status indication of the Interface. Encodings as follows: 0000b : Reset 0001b : Active 0011b : Active.PMNAK 0100b : L1 1000b : L2 1001b : LinkReset 1010b : LinkError 1011b : Retrain 1100b : Disabled All other encodings are reserved. The status signal is permitted to transition from Adapter autonomously when applicable. For example the Adapter asserts the Retrain status when it decides to enter retraining either autonomously or when requested by remote agent. For PCIe/Streaming protocols, the Adapter LSM is exposed as pl_state_sts to the Protocol Layer. For CXL protocol, the ARB/MUX vLSM is exposed as pl_state_status to the Protocol Layer. The Link Status is considered to be Up from Protocol Layer perspective when FDI status is Active, Active.PMNAK, Retrain, L1 or L2. The Link Status is considered Down for other states of FDI.
pl_inband_pres	Adapter to the Protocol Layer indication that the Die-to-Die Link has finished negotiation of parameters with remote Link partner and is ready for transitioning the FDI Link State Machine (LSM) to Active. Once it transitions to 1b, this must stay 1b until FDI moves to Active or LinkError. It stays asserted while FDI is in Retrain, Active.PMNAK, L1 or L2. It must deassert during LinkReset, Disabled or LinkError states.

Signal Name	Signal Description
pl_error	Adapter to the Protocol Layer indication that it has detected a framing related error. It is pipeline matched with the receive data path. It must also assert if **pl_error** was asserted on RDI by the Physical Layer for a Flit which the Adapter is forwarding to the Protocol Layer. It is permitted for Protocol Layer to use **pl_error** indication to log correctable errors when Retry is enabled from the Adapter. The Adapter must finish any partial Flits sent to the Protocol Layer and assert **pl_flit_cancel** in order to prevent consumption of that Flit by the Protocol Layer. Adapter must initiate Link Retrain on RDI following this, if it was a framing error detected by the Adapter.
pl_cerror	Adapter to the Protocol Layer indication that a correctable error was detected that does not affect the data path and will not cause Retrain on the Link. The Protocol Layer must OR the **pl_error** and **pl_cerror** signals for Correctable Error Logging. The Adapter must OR any internally detected errors with the **pl_cerror** indication on RDI and forward it to the Protocol Layer.
pl_nferror	Adapter to the Protocol Layer indication that a non-fatal error was detected. This is used by Protocol Layer for error logging and corresponding escalation to software. The Adapter must OR any internally detected errors with **pl_nferror** on RDI and forward the result on FDI.
pl_trainerror	Indicates a fatal error from the Adapter. Adapter must transition **pl_state_sts** to LinkError if not already in LinkError state. Implementations are permitted to map any fatal error to this signal that require upper layer escalation (or interrupt generation) depending on system level requirements.
pl_rx_active_req	Adapter asserts this signal to request the Protocol Layer to open its Receiver's data path and get ready for receiving protocol data or Flits. The rising edge of this signal must be when **pl_state_sts** is Reset, Retrain or Active. Together with **lp_rx_active_sts**, it forms a four way handshake. See section 8.2.6 for rules related to this handshake.
lp_rx_active_sts	Protocol Layer responds to **pl_rx_active_req** after it is ready to receive and parse protocol data or Flits. Together with **pl_rx_active_req**, it forms a four way handshake. See section 8.2.6 for rules related to this handshake.
pl_protocol[2:0]	Adapter indication to Protocol Layer the protocol that was negotiated during training. It has the following encodings: 000b - PCIe 011b - CXL.1 [Single protocol, i.e. CXL.io] 100b - CXL.2 [Multi-protocol, Type 1 device] 101b - CXL.3 [Multi-protocol, Type 2 device] 110b - CXL.4 [Multi-protocol, Type 3 device] 111b - Streaming Protocol Other encodings are Reserved

Signal Name	Signal Description
pl_protocol_flitfmt[3:0]	This indicates the negotiated Format. Refer to Chapter 3.0 for the definitions of these formats. 0001b - Format 1 : Raw Mode 0010b - Format 2 : CXL 2.0 68B Flit Mode 0011b - Format 3 : PCIe 256B Flit Mode 0100b - Format 4 : CXL Standard 256B Flit Mode 0101b - Format 5 : CXL Latency Optimized Mode, no optional bytes 0110b - Format 6 : CXL Latency Optimized Mode with optional bytes 0111b - Format 7 : CXL 68B Enhanced Flit Mode Other encodings are Reserved
pl_protocol_vld	Indication that **pl_protocol**, and **pl_protocol_flitfmt** have valid information. This is a level signal, asserted when the Adapter has determined the appropriate protocol, but must only de-assert again after subsequent transitions to LinkError or Reset state depending on the Link state machine transitions. Protocol Layer must sample and store **pl_protocol** and **pl_protocol_flitfmt** when **pl_protocol_vld** = 1 and **pl_state_sts** = Reset and **pl_inband_pres** = 1. It must treat this saved value as the negotiated protocol until **pl_state_sts** = Reset and **pl_inband_pres** = 0. The Adapter must ensure that if **pl_inband_pres** = 1, **pl_protocol_vld** = 1 and **pl_state_sts** = Reset, then **pl_protocol** and **pl_protocol_flitfmt** are the correct values that can be sampled by the Protocol Layer.
pl_stallreq	Adapter request to Protocol Layer to flush all Flits for state transition and not prepare any new Flits. See section 8.2.5 for details.
lp_stallack	Protocol Layer to Adapter indication that the Flits are aligned and stalled (if **pl_stallreq** was asserted). It is strongly recommended that this response logic be on a global free running clock, so the Protocol Layer can respond to **pl_stallreq** with **lp_stallack** even if other significant portions of the Protocol Layer are clock gated.
pl_phyinrecenter	Adapter indication to Protocol Layer that the Link is doing training or retraining(i.e. RDI has **pl_phyinrecenter** asserted or the Adapter LSM has not moved to Active yet). If this is asserted during a state where clock gating is permitted, the **pl_clk_req/lp_clk_ack** handshake must be performed with the upper layer. The upper layers are permitted to use this to update the "Link Training/Retraining" bit in the UCIe Link Status register.
pl_phyinl1	Adapter indication to Protocol Layer that the Physical Layer is in L1 power management state (i.e. RDI is in L1 state).
pl_phyinl2	Adapter indication to Protocol Layer that the Physical Layer is in L2 power management state (i.e. RDI is in L2 state).

Signal Name	Signal Description
`pl_speedmode[2:0]`	Current Link speed. The following encodings are used: 000b: 4GT/s 001b: 8GT/s 010b: 12GT/s 011b: 16GT/s 100b: 24GT/s 101b: 32GT/s other encodings are reserved. The Protocol Layer must only consider this signal to be relevant when the RDI state is Active or Retrain. For multi-module configurations, all modules must operate at the same speed.
`pl_lnk_cfg[2:0]`	Current Link Configuration. Indicates the current operating width of a module. 001b: x8 010b: x16 011b: x32 100b: x64 101b: x128 110b: x256 other encodings are reserved. The Protocol Layer must only consider this signal to be relevant when the FDI state is Active or Retrain. This is the total width across all Active modules for the corresponding FDI instance.
`pl_clk_req`	Request from the Adapter to remove clock gating from the internal logic of the Protocol Layer. This is an asynchronous signal from the Protocol Layer's perspective since it is not tied to **`lclk`** being available in the Protocol Layer. Together with **`lp_clk_ack`**, it forms a four-way handshake to enable dynamic clock gating in the Protocol Layer. When dynamic clock gating is supported, the Protocol Layer must use this signal to exit clock gating before responding with **`lp_clk_ack`**. If dynamic clock gating is not supported, it is permitted for the Adapter to tie this signal to 1b.
`lp_clk_ack`	Response from the Protocol Layer to the Adapter acknowledging that its clocks have been ungated in response to **`pl_clk_req`**. This signal is only asserted when **`pl_clk_req`** is asserted, and de-asserted after **`pl_clk_req`** has de-asserted. When dynamic clock gating is not supported by the Protocol Layer, it must stage **`pl_clk_req`** internally for one or more clock cycles and turn it around as **`lp_clk_ack`**. This way it will still participate in the handshake even though it does not support dynamic clock gating.
`lp_wake_req`	Request from the Protocol Layer to remove clock gating from the internal logic of the Adapter. This is an asynchronous signal relative to **`lclk`** from the Adapter's perspective since it is not tied to **`lclk`** being available in the Adapter. Together with **`pl_wake_ack`**, it forms a four-way handshake to enable dynamic clock gating in the Adapter. When dynamic clock gating is supported, the Adapter must use this signal to exit clock gating before responding with **`pl_wake_ack`**. If dynamic clock gating is not supported, it is permitted for the Protocol Layer to tie this signal to 1b.

Signal Name	Signal Description
pl_wake_ack	Response from the Adapter to the Protocol Layer acknowledging that its clocks have been ungated in response to **lp_wake_req**. This signal is only asserted after **lp_wake_req** has asserted, and is de-asserted after **lp_wake_req** has de-asserted. When dynamic clock gating is not supported by the Adapter, it must stage **lp_wake_req** internally for one or more clock cycles and turn it around as **pl_wake_ack**. This way it will still participate in the handshake even though it does not support dynamic clock gating.
pl_cfg[NC-1:0]	This is the sideband interface from the Adapter to the Protocol Layer. See Chapter 6.0 for details. NC is the width of the interface. Supported values are 8, 16, and 32.
pl_cfg_vld	When asserted, indicates that **pl_cfg** has valid information that should be consumed by the Protocol Layer.
pl_cfg_crd	Credit return for sideband packets from the Adapter to the Protocol Layer for sideband packets. Each credit corresponds to 64-bit of header and 64-bit of data. Even transactions that don't carry data or carry 32-bit of data consume the same credit and the Receiver returns the credit once the corresponding transaction has been processed or de-allocated from its internal buffers. Refer to section 6.1.3.1 for additional flow control rules.
lp_cfg[NC-1:0]	This is the sideband interface from Protocol Layer to the Adapter. See Chapter 6.0 for details. NC is the width of the interface. Supported values are 8, 16, and 32.
lp_cfg_vld	When asserted, indicates that **lp_cfg** has valid information that should be consumed by the Adapter.
lp_cfg_crd	Credit return for sideband packets from the Protocol Layer to the Adapter for sideband packets. Each credit corresponds to 64-bit of header and 64-bit of data. Even transactions that don't carry data or carry 32-bit of data consume the same credit and the Receiver returns the credit once the corresponding transaction has been processed or de-allocated from its internal buffers. Refer to section 6.1.3.1 for additional flow control rules.

8.2.1 Interface reset requirements

FDI does not define a separate interface signal for reset; however, it is required that the logic entities on both sides of FDI are in the same reset domain and the reset for each side is derived from the same source.

8.2.2 Interface clocking requirements

FDI requires both sides of the interface to be on the same clock domain. Moreover, the clock domain for sideband interface (***cfg***) is the same as the mainband signals.

Each side is permitted to instantiate clock crossing FIFOs internally if needed, as long as it does not violate the requirements at the interface itself.

It is important to note that there is no back pressure possible from the Protocol Layer to the Adapter on the main data path. So any clock crossing related logic internal to the Protocol Layer must take this into consideration.

8.2.3 Dynamic clock gating

Dynamic coarse clock gating is permitted in the Adapter and Protocol Layer when `pl_state_sts` is Reset, LinkReset, Disabled or PM states. This section defines the rules around entry and exit of clock gating. Note that clock gating is not permitted in LinkError states - it is expected that the UCIe usages will enable error handlers to make sure the Link is not stuck in a LinkError state, if the intent is to save power when a Link is in an error state.

8.2.3.1 Rules and description for lp_wake_req/pl_wake_ack handshake

Protocol Layer can request removal of clock gating of the Adapter by asserting `lp_wake_req` (asynchronous to `lclk` availability in the Adapter). All Adapter implementations must respond with a `pl_wake_ack` (synchronous to `lclk`). The extent of internal clock ungating when `pl_wake_ack` is asserted is implementation-specific, but lclk must be available by this time to enable FDI interface transitions from the Protocol Layers. The Wake Req/Ack is a full handshake and it must be used for state transition requests (on `lp_state_req` or `lp_linkerror`) when moving away from Reset or PM states. It must also be used for sending packets on the sideband interface.

Rules for this handshake:

1. Protocol Layer asserts `lp_wake_req` to request ungating of clocks by the Adapter.
2. The Adapter asserts `pl_wake_ack` to indicate that clock gating has been removed. There must be at least one clock cycle bubble between `lp_wake_req` assertion and `pl_wake_ack` assertion.
3. `lp_wake_req` must de-assert before `pl_wake_ack` de-asserts. It is the responsibility of the Protocol Layer to control the specific scenario of de-assertion. As an example, when performing the handshake for a state request, it is permitted to keep `lp_wake_req` asserted until it observes the desired state status. Protocol Layer is also permitted to keep `lp_wake_req` asserted through states where clock gating is not allowed in the Adapter (i.e. Active, LinkError or Retrain).
4. `lp_wake_req` should not be the only consideration for Adapter to perform clock gating, it must take into account `pl_state_sts` and other internal or Link requirements before performing global and/or local clock gating.
5. When performing `lp_wake_req/pl_wake_ack` handshake for `lp_state_req` transitions or `lp_linkerror` transition, the Protocol Layer is permitted to not wait for `pl_wake_ack` before changing `lp_state_req or lp_linkerror`.
6. When performing `lp_wake_req/pl_wake_ack` handshake for `lp_cfg` transitions, Protocol Layer must wait for `pl_wake_ack` before changing `lp_cfg` or `lp_cfg_vld`. Because `lp_cfg` can have multiple transitions for a single packet transfer, it is necessary to make sure that the Adapter clocks are up before transfer begins.

8.2.3.2 Rules and description for pl_clk_req/lp_clk_ack handshake

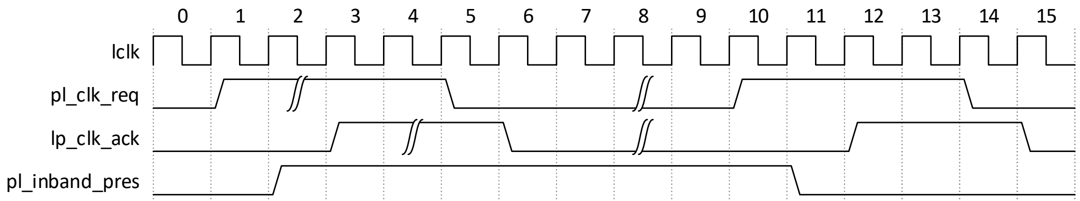
Adapter is allowed to initiate `pl_clk_req/lp_clk_ack` handshake at any time and the Protocol Layer must respond.

Rules for this handshake:

1. Adapter asserts `pl_clk_req` to request removal of clock gating by the Protocol Layer. This can be done anytime, and independent of current FDI state.
2. The Protocol Layer asserts `lp_clk_ack` to indicate that clock gating has been removed. There must be at least one clock cycle bubble between `pl_clk_req` assertion and `lp_clk_ack` assertion.

3. **pl_clk_req** must de-assert before **lp_clk_ack**. It is the responsibility of the Adapter to control the specific scenario of de-assertion, after the required actions for this handshake are completed.
4. **pl_clk_req** should not be the only consideration for the Protocol Layer to perform clock gating, it must take into account **pl_state_sts** and other protocol-specific requirements before performing trunk and/or local clock gating.
5. The Adapter must use this handshake to ensure transitions of **pl_inband_pres, pl_phyinl1, pl_phyinl2, pl_phyinrecenter,** and **pl_rx_active_req** have been observed by the Protocol Layer. Since these are level oriented signals, the Adapter is permitted to let the signal transition without waiting for **lp_clk_ack**. When this is done during initial Link bring up, it is strongly recommended for the Adapter to keep **pl_clk_req** asserted until the state status transitions away from Reset to a state where clock gating is not permitted or until the state status is Reset and **pl_inband_pres** de-asserts.

Figure 117. Example Waveform Showing Handling of Level Transition



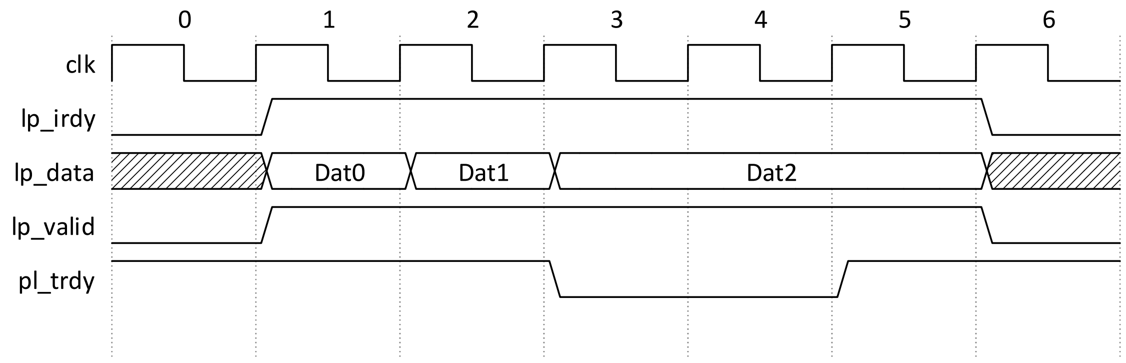
6. The Adapter must also perform this handshake before transition to LinkError state from Reset, LinkReset, Disabled or PM state (especially when the LinkError transition occurs by the Adapter without being directed by the Protocol Layer). It is permitted to assert **pl_clk_req** before the state change, in which case it must stay asserted until the state status transitions. It is also permitted to assert **pl_clk_req** after the state status transition, but in this case Adapter must wait for **lp_clk_ack** before performing another state transition.
7. The Adapter must also perform this handshake when the status is PM and remote Link partner is requesting PM exit. For exit from Reset, LinkReset, Disabled or PM states to a state that is not LinkError, it is required to assert **pl_clk_req** before the status change, and in this case it must stay asserted until the state status transitions away from Reset or PM.
8. The Adapter must also perform this handshake for sideband transfers from the Adapter to the Protocol Layer. When performing the handshake for **pl_cfg** transitions, Adapter must wait for **lp_clk_ack** before changing **pl_cfg** or **pl_cfg_vld**. Because **pl_cfg** can have multiple transitions for a single packet transfer, it is necessary to make sure that the Protocol Layer clocks are up before transfer begins.

When clock-gated in Reset states, Protocol Layers that rely on dynamic clock gating to save power must wait in clock gated state for **pl_inband_pres**=1. The Adapter will request clock gating exit when it transitions **pl_inband_pres**, and the Protocol Layer must wait for **pl_inband_pres** assertion before requesting **lp_state_req** = ACTIVE. If **pl_inband_pres** de-asserts while **pl_state_sts** = Reset, then the Protocol Layer is permitted to return to clock-gated state after moving **lp_state_req** to NOP.

8.2.4 Data Transfer

As indicated in the signal list descriptions, when Protocol Layer is sending data to the Adapter, data is transferred when **lp_irdy**, **pl_trdy** and **lp_valid** are asserted. Figure 118 shows an example waveform for data transfer from the Protocol Layer to the Adapter. Data is transmitted on clock cycles 1, 2, and 5. No assumption should be made by Protocol Layer about when **pl_trdy** can de-assert or for how many cycles it remains de-asserted before it is asserted again, unless explicitly guaranteed by the Adapter. If a Flit transfer takes multiple clock cycles, the Protocol Layer is not permitted to insert bubbles in the middle of a Flit transfer (i.e. **`lp_valid`** and **`lp_irdy`** must be asserted continuously until the Flit transfer is complete. Of course, data transfer can stall because of **`pl_trdy`** de-assertion).

Figure 118. Data Transfer from Protocol Layer to Adapter



As indicated in the signal list descriptions, when Adapter is sending data to the Protocol layer, there is no back-pressure mechanism, and data is transferred whenever **`pl_valid`** is asserted. The Adapter is permitted to insert bubbles in the middle of a Flit transfer and the Protocol Layer must be able to handle that.

8.2.4.1 DLLP transfer rules for 256B Flit Mode

For PCIe and CXL.io 256B Flits (both Standard and Latency Optimized), FDI provides a separate signal for DLLP transfers from the Protocol Layer to the Adapter and vice-versa. Since the DLLPs have to bypass the Retry buffer, the separate signal enables the Adapter to insert DLLPs into the Flits from the Protocol Layer or the Retry buffer, if it is permitted to do so per the Flit packing rules of the corresponding Flit format. Rules relevant for FDI operation (per FDI instance) are outlined below:

For the Transmitting side:

- Protocol Layer is responsible for sending the relevant DLLPs at the rate defined by the underlying Protocol to prevent timeouts of DLLP exchanges. If the Protocol Layer has no TLPs to send, it must insert NOP Flits to ensure that the Adapter gets an opportunity to insert the DLLP bytes.
- When transferring DLLP over multiple chunks across FDI, Byte 0 is transferred on the first chunk LSB, Byte 1 following it and so on.
- The Adapter must have storage for at least 1 DLLP of every unique DLLP encoding (including Optimized_Update_FC) per supported VC that is possible for transfer to remote Link partner. The Adapter tracks pending DLLPs and schedules them on the next available opportunity for the relevant Flits. Credit update DLLPs must not be reordered for a VC by the Adapter. It is however permitted to discard a pending credit DLLP if the Protocol Layer presented a new credit DLLP of the same FC and VC. This extends to Optimized_Update_FC packets; i.e., it is permitted to discard any pending NP or P Update FC DLLPs, if the Protocol Layer transferred an Optimized_Update_FC for the corresponding VC.

On the Receiving side:

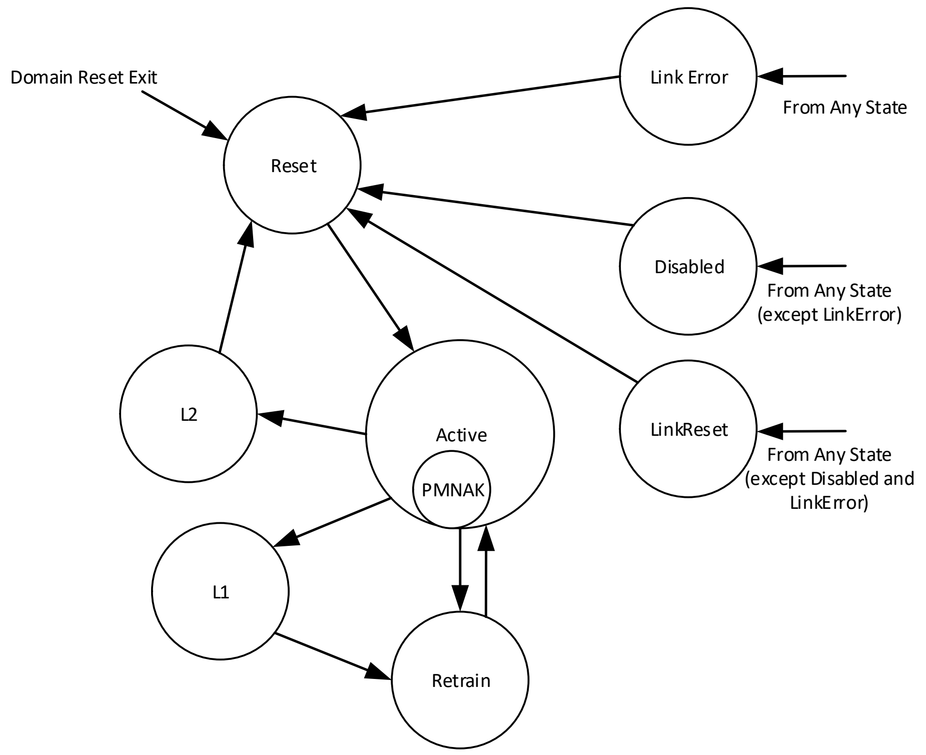
- The Adapter must extract DLLPs from received Flits of the corresponding protocol and forward them to the Protocol Layer. The FDI signal width of **`pl_dllp`** must be wide enough to keep up with the maximum rate of DLLPs that could be received from the Link.

- When transferring DLLP over multiple chunks across FDI, Byte 0 is transferred on the first chunk LSB, Byte 1 following it and so on.
- The Protocol Identifier corresponding to D2D Adapter in the Flit Header overlaps with the Flit usage of NOP Flits defined in PCIe and CXL specifications. The Adapter must check for available DLLPs in these Flits as well. All 0 bits in the DLLP byte positions indicate the a NOP DLLP, and must not be forwarded to the Protocol Layer.

8.2.5 FDI State Status Machine

Figure 119 shows the RDI interface state machine.

Figure 119. FDI Interface State Machine



8.2.6 Rx_active_req/Sts Handshake

The Adapter negotiates Active state transitions on FDI using sideband messages when the Adapter LSM is exposed to the Protocol Layer. Since the sideband Link is running slower than the main-band Link, the Adapter needs to make sure that the Protocol Layer's Receiver is already in Active state (even though `pl_state_sts` might not have moved to Active yet) before responding to the Active request sideband message from remote Link partner. Rx_active_req/Sts handshake facilitates this.

When CXL is sent over UCIe, ARB/MUX functionality is performed by the Adapter and CXL vLSMs are exposed on FDI. Even though ALMPs are transmitted over main-band, the interface to the Protocol Layer is FDI and it follows the rules of Rx_active_req/Sts Handshake as well.

Rules for this handshake:

1. The Adapter (or ARB/MUX) asserts **pl_rx_active_req** in order to trigger the Protocol Layer to open its Receiver's data path for receiving protocol data or Flits. This signal does not affect the Transmitter data path (it must wait for **pl_state_sts** to move to Active and follow the rules of **pl_trdy**). **pl_rx_active_req** should have a rising edge only when **lp_rx_active_sts** = 0 and **pl_state_sts** is Reset, Retrain or Active.
2. The Protocol Layer asserts **lp_rx_active_sts** after **pl_rx_active_req** has asserted and when it is ready to receive protocol data or Flits. There must be at least one clock cycle delay between **lp_rx_active_sts** assertion and **lp_rx_active_sts** assertion to prevent a combinatorial loop.
3. When **pl_rx_active_req** = 1 and **lp_rx_active_sts** = 1, the Receiver is in Active state if **pl_state_sts** is Reset, Retrain or Active.
4. **pl_rx_active_req** should have a falling edge only when **lp_rx_active_sts** = 1. This must trigger Protocol Layer to deassert **lp_rx_active_sts**, and this completes the transition of the Receiver away from Active state.
5. For graceful exit from Active state (i.e. a transition to PM, Retrain,LinkReset or Disabled states), both **pl_rx_active_req** and **lp_rx_active_sts** must de-assert before **pl_state_sts** transitions away from Active.
6. If **pl_rx_active_req** = 0 while **pl_state_sts** = Active, the Adapter must guarantee no Flits would be sent to the Protocol Layer (for example, this can happen if the Adapter LSM or RDI is in Retrain, but the vLSM exposed to Protocol Layer is still in Active). Thus, it is permitted to perform this handshake even when the state status on FDI remains Active throughout.
7. For Active to LinkError transition, it is permitted for **pl_state_sts** to transition to LinkError before **pl_rx_active_req** de-asserts, but both **pl_rx_active_req** and **lp_rx_active_sts** must de-assert before **pl_state_sts** transitions away from LinkError.

8.2.7 FDI Bring up flow

Figure 120 shows an example flow for Stage 3 of the Link bring up highlighting the transitions on FDI. This stage requires sequencing on FDI that co-ordinates the state transition from Reset to Active. If multiple stacks of protocol or ARB/MUX is present, the same sequence happens independently for each Protocol Layer stack. The flows on FDI are illustrated for Adapter 0 LSM in the sideband message encodings, however Adapter 1 LSM must send the sideband message encodings corresponding to Adapter 1 to its remote Link partner.

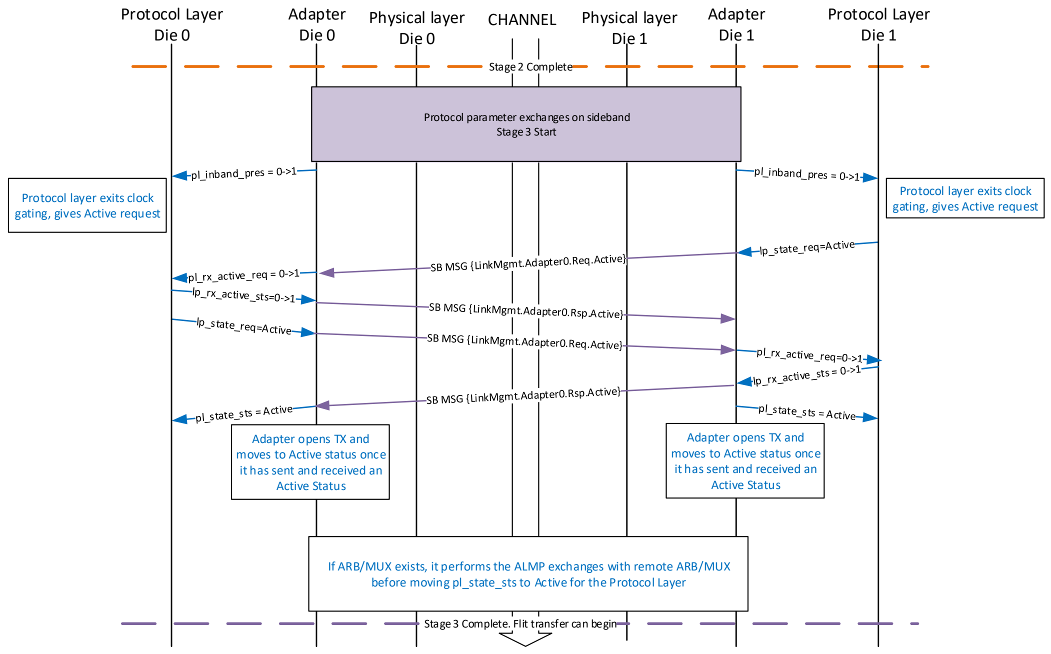
1. Once Adapter has completed transition to Active on RDI and successful parameter negotiation with the remote Link partner, it must do the **pl_clk_req** handshake with the Protocol Layer and reflect **pl_inband_pres**=1 on FDI. Note that the **pl_clk_req** handshake is not shown in the example flow in Figure 120
2. This is the trigger for Protocol Layer to request Active state. It is permitted for the Protocol Layer to wait unlit **pl_protocol_vld** = 1 before requesting Active. It must perform the **lp_wake_req** handshake as described in section 8.2.3.1. Note that the **lp_wake_req** handshake is not shown in the example flow in Figure 120.
3. On sampling **lp_state_req** = Active, the Adapter must send the {LinkMgmt.Adapter0.Req.Active} message on sideband to remote Link partner.
4. The Adapter must respond to the {LinkMgmt.Adapter.Req.Active} sideband message with a {LinkMgmt.Adapter.Rsp.Active} message on sideband after making sure that the Protocol Layer's Receiver is ready. The {LinkMgmt.Adapter0.Rsp.Active} must only be sent after the Adapter has sampled

pl_rx_active_req = **lp_rx_active_sts** = 1. As mentioned previously, the **pl_clk_req** handshake applies to **pl_rx_active_req** as well; it is permitted for the Adapter to keep **pl_clk_req** asserted continuously (once it has been asserted for **pl_inband_pres**) while doing the bring up flow.

5. If no ARB/MUX is present, once the Adapter has sent and received the {LinkMgmt.Adapter0.Rsp.Active} sideband message, it must transition **pl_state_sts** to Active for the Protocol Layer, and Flit transfer can begin.
6. If ARB/MUX is present, the sending and receipt of {LinkMgmt.Adapter0.Rsp.Active} sideband message opens up the ARB/MUX to perform ALMP exchanges over main-band and eventually transition the vLSMs to Active state.

Steps 3 to 6 constitute the "Active Entry Handshake" on FDI and must be performed for every entry to Active state.

Figure 120. FDI Bring up flow


Protocol Layer Die 0
Adapter Die 0
Physical layer Die 0
CHANNEL
Physical layer Die 1
Adapter Die 1
Protocol Layer Die 1
Stage 2 Complete
Protocol parameter exchanges on sideband
Stage 3 Start
pl_inband_pres = 0->1
pl_inband_pres = 0->1
Protocol layer exits clock gating, gives Active request
Protocol layer exits clock gating, gives Active request
lp_state_req=Active
SB MSG {LinkMgmt.Adapter0.Req.Active}
pl_rx_active_req = 0->1
lp_rx_active_sts=0->1
SB MSG {LinkMgmt.Adapter0.Rsp.Active}
lp_state_req=Active
SB MSG {LinkMgmt.Adapter0.Req.Active}
pl_rx_active_req=0->1
lp_rx_active_sts = 0->1
SB MSG {LinkMgmt.Adapter0.Rsp.Active}
pl_state_sts = Active
pl_state_sts = Active
Adapter opens TX and moves to Active status once it has sent and received an Active Status
Adapter opens TX and moves to Active status once it has sent and received an Active Status
If ARB/MUX exists, it performs the ALMP exchanges with remote ARB/MUX before moving pl_state_sts to Active for the Protocol Layer
Stage 3 Complete. Flit transfer can begin

8.2.8 FDI PM Flow

This section describes the sequencing and rules for PM entry and exit on FDI. The rules are the same for L1 or L2 entry. L1 exit transitions the state machine through Retrain to Active, whereas L2 exit transitions the state machine through Reset to Active. The flow illustrations in the section use L1 as an example. A "PM request" sideband message is {LinkMgmt.Adapter*.Req.L1} or {LinkMgmt.Adapter*.Req.L2}. A "PM response" sideband message is {LinkMgmt.Adapter*.Rsp.L1} or {LinkMgmt.Adapter*.Rsp.L2}. The flows on FDI are illustrated for Adapter 0 LSM in the sideband message encodings, however Adapter 1 LSM must send the sideband message encodings corresponding to Adapter 1 to its remote Link partner.

- The Protocol Layer requests PM entry on FDI after idle time criteria has been met. The criteria for idle time is implementation specific and could be dependent on the protocol. For PCIe and CXL.io protocols, PM DLLPs are **<u>not</u>** used to negotiate PM entry/exit when using D2D Adapter's Retry buffer (i.e. UCIe Flit Mode).

- If operating in UCIe Flit Mode, ARB/MUX is present within the D2D Adapter, and it follows the rules of CXL specification (for 256B Flit Mode) to take the vLSMs to the corresponding PM state. Note that even for CXL 1.1, CXL 2.0, 68B-Enhanced Flit Mode, the same ALMP rules as 256B Flit Mode are used. This is a simplification on UCIe, since ALMPs always go through the retry buffer. Once vLSMs are in the PM state, ARB/MUX requests the Adapter Link State Machine to enter the corresponding PM state. The Adapter Link State Machine transition to PM follows the same rules as outlined for Protocol Layer and Adapter below.
- If CXL or PCIe protocol has been negotiated, only the upstream port (UP) can initiate PM entry. This is done using a sideband message from the UP Adapter to the downstream port (DP) Adapter. DP Adapter must not initiate entry into PM. PM support is required for CXL and PCIe protocols. PM entry is considered successful and complete once UP receives a valid "PM Response" sideband message.Figure 121 shows an example flow for CXL or PCIe protocol PM Entry on FDI and Adapter. Once the FDI status is PM for all Protocol Layers, the Adapter can request PM transition on RDI.
- If Streaming protocol has been negotiated or UCIe is in Raw Mode, then both side Adapters must issue PM entry request through a sideband message once the conditions of PM entry have been satisfied. PM entry is considered successful and complete once both sides have received a valid "PM Response" sideband message. Figure 122 shows an example flow for symmetric protocols. Once the FDI status is PM for all Protocol Layers, the Adapter can request PM transition on RDI.
- Protocol Layer requests PM entry once it has blocked transmission of any new Protocol Layer Flits, by transitioning **`lp_state_req`** to L1 or L2 encoding. Once requested, the Protocol Layer cannot change this request until it observes the corresponding PM state, Retrain, Active.PMNAK or LinkError state on **`pl_state_sts`**; unless it is a DP Protocol Layer for PCIe or CXL. Once the FDI state is resolved, the Adapter must first bring it back to Active before processing any new PM requests from the Protocol Layer.
    - If the resolution is PM (upon successful PM entry) and the Protocol Layer needs to exit PM (or there is a pending Protocol Layer Active request from remote Link partner) then the Protocol Layer must initiate PM exit flow on FDI by requesting **`lp_state_req`** = Active. All PM entry related handshakes must have finished prior to this (for CXL/PCIe protocols, this is when UP has received a valid "PM Response" sideband message. For symmetric protocols, this is when both sides Adapter have received a valid "PM Response" sideband message).
    - If the resolution is Active.PMNAK, the Protocol Layer must initiate a request of Active on FDI. Once the status moves to Active, it must wait for at least 2 us before re-requesting PM entry (if all conditions of PM entry are still met). Figure 123 shows an example of PM abort flow. The PM request could have been from either side depending on the configuration. Protocol Layer must continue receiving protocol data or Flits while the status is Active or Active.PMNAK.
    - DP Protocol Layer for PCIe or CXL is permitted to change request from PM to Active without waiting for PM or Active.PMNAK - however, it is still possible for the Adapter to initiate a stallreq/ack and complete PM entry if it was in the process of committing to PM entry when the Protocol Layer changed its request. In this scenario, the Protocol Layer will see **`pl_state_sts`** transition to PM and it is permitted to continue asking for the new state request.
    - If the resolution is LinkError, then the Link is down and it resets any outstanding PM handshakes.
- Adapter (UP port only if CXL or PCIe protocol), initiates a sideband "PM request" once it samples a PM request on **`lp_state_req`** and has completed the StallReq/

Ack handshake with its Protocol Layer and its Retry buffer is empty (all pending Acks have been received).

- If the Adapter LSM moves to Retrain while waiting for a "PM Response" message, it must wait for the response. Once the response is received, it must transition back to Active before requesting a new PM entry. Note that the transition to Active requires Active Entry handshake with the remote Link partner, and that will cause the remote partner to exit PM. If the Adapter LSM receives a "PM Request" after it has transitioned to Retrain, it must respond with {LinkMgmt.Adapter0.Rsp.PMNAK} immediately.
- Once the Adapter receives a sideband "PM request", it must respond to it within 2 us (The time is only counted during the Adapter LSM being in Active state.):
  - — if its local Protocol Layer is requesting PM, it must respond with the corresponding "PM Response" sideband message after finishing the StallReq/ Ack handshake with its Protocol Layer and its Retry buffer being empty. If the current status is not PM, it must transition **`pl_state_sts`** to PM after responding to the sideband message.
  - — If the current **`pl_state_sts`** = PM, it must respond with "PM Response" sideband message.
  - — If **`pl_state_sts`** = Active and **`lp_state_req`** = Active and it stays this way for 1us after receiving the "PM Request" message, it must respond with {LinkMgmt.Adapter0.Rsp.PMNAK} sideband message. The time is only counted during all the relevant state machines being in Active state.
- If the Adapter receives a sideband "PM Response" message in response to a "PM Request" message, it must transition **`pl_state_sts`** on its local FDI to PM (if it is not currently in a PM state).
- If the Adapter receives a sideband {LinkMgmt.Adapter0.Rsp.PMNAK} message in response to a "PM Request" message, it must transition **`pl_state_sts`** on its local FDI to Active.PMNAK state. It is permitted to retry PM entry handshake (if all conditions of PM entry are satisfied) at least 2us after receiving the {LinkMgmt.Adapter0.Rsp.PMNAK} message.
- PM exit is initiated by the Protocol Layer requesting Active on FDI. This triggers the Adapter to initiate PM exit by performing the Active Entry handshakes on sideband. Figure 124 shows an example flow of PM exit on FDI when Adapter LSM is exposed.
  - — PM exit handshake completion requires both Adapters to send as well as receive an {LinkMgmt.Adapter0.Rsp.Active} message on sideband. Once this has completed, the Adapter is permitted to transition Adapter LSM to Active.
  - — If **`pl_state_sts`** = PM and an {LinkMgmt.Adapter0.Req.Active} message is received, the Adapter must initiate **`pl_clk_req`** handshake with the Protocol Layer, and transition Adapter LSM to Retrain (For L2 exit, the transition is to Reset). This must trigger the Protocol Layer to request Active on **`lp_state_req`** (if not already doing so), and this in turn triggers the Adapter to send {LinkMgmt.Adapter0.Req.Active} sideband message to the remote Link partner.

Note that the figures below are examples and they do not show the **`lp_wake_req`**, **`pl_clk_req`** handshakes. Implementations must follow the rules outlined for these handshakes in previous sections.

Figure 121. PM Entry example for CXL or PCIe protocols

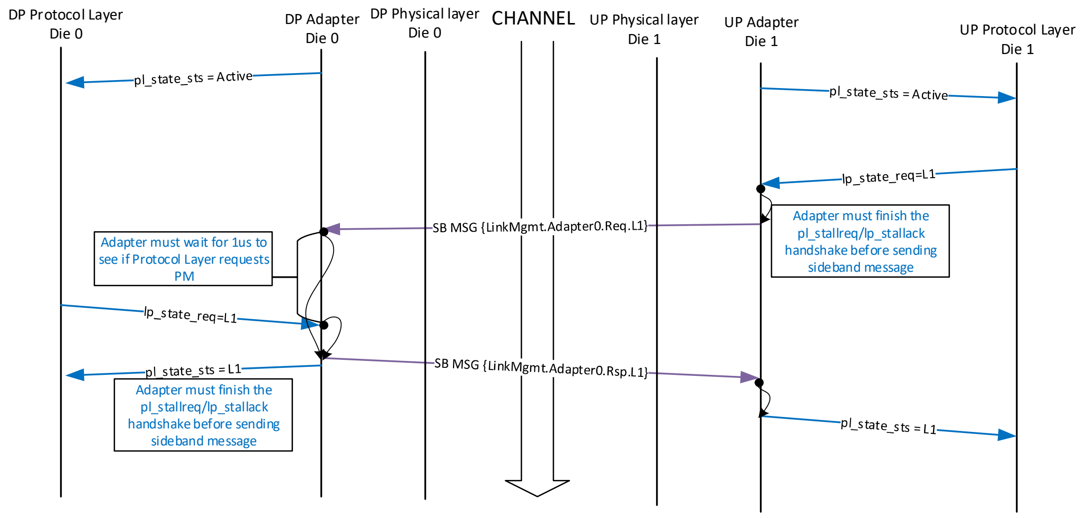

DP Protocol Layer Die 0
DP Adapter Die 0
DP Physical layer Die 0
CHANNEL
UP Physical layer Die 1
UP Adapter Die 1
UP Protocol Layer Die 1
pl_state_sts = Active
pl_state_sts = Active
lp_state_req=L1
SB MSG {LinkMgmt.Adapter0.Req.L1}
Adapter must finish the pl_stallreq/lp_stallack handshake before sending sideband message
Adapter must wait for 1us to see if Protocol Layer requests PM
lp_state_req=L1
SB MSG {LinkMgmt.Adapter0.Rsp.L1}
pl_state_sts = L1
Adapter must finish the pl_stallreq/lp_stallack handshake before sending sideband message
pl_state_sts = L1

Figure 122. PM Entry example for symmetric protocol

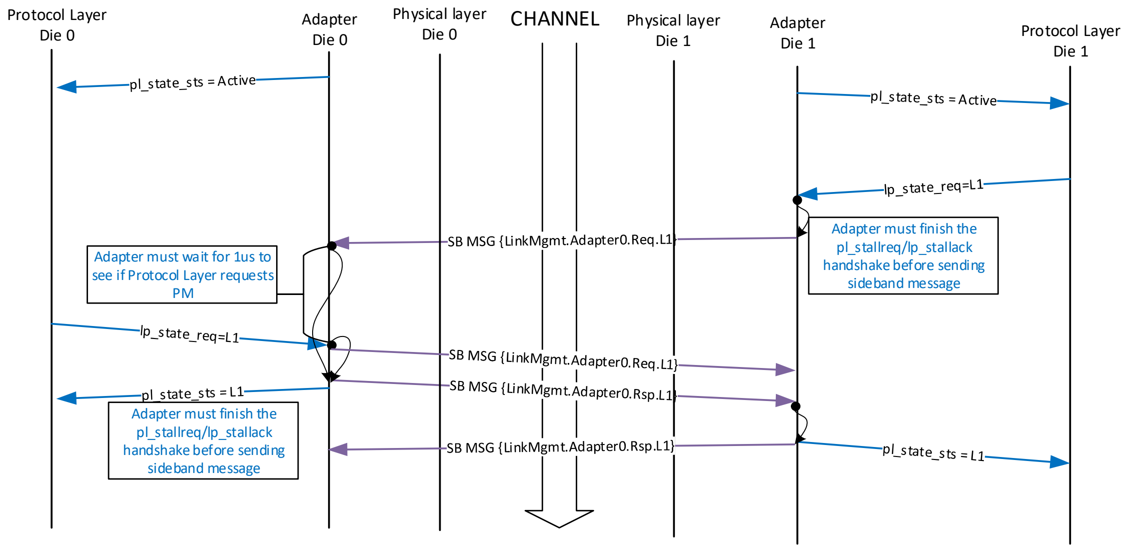

Protocol Layer Die 0
Adapter Die 0
Physical layer Die 0
CHANNEL
Physical layer Die 1
Adapter Die 1
Protocol Layer Die 1
pl_state_sts = Active
pl_state_sts = Active
lp_state_req=L1
SB MSG {LinkMgmt.Adapter0.Req.L1}
Adapter must finish the pl_stallreq/lp_stallack handshake before sending sideband message
Adapter must wait for 1us to see if Protocol Layer requests PM
lp_state_req=L1
SB MSG {LinkMgmt.Adapter0.Req.L1}
SB MSG {LinkMgmt.Adapter0.Rsp.L1}
pl_state_sts = L1
Adapter must finish the pl_stallreq/lp_stallack handshake before sending sideband message
SB MSG {LinkMgmt.Adapter0.Rsp.L1}
pl_state_sts = L1

Figure 123. PM Abort Example

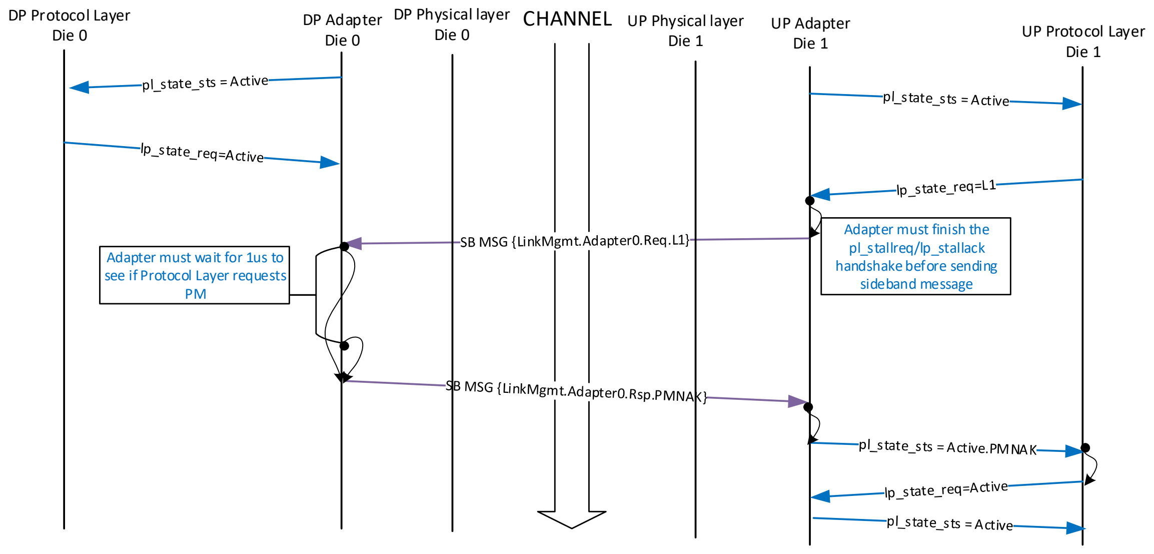
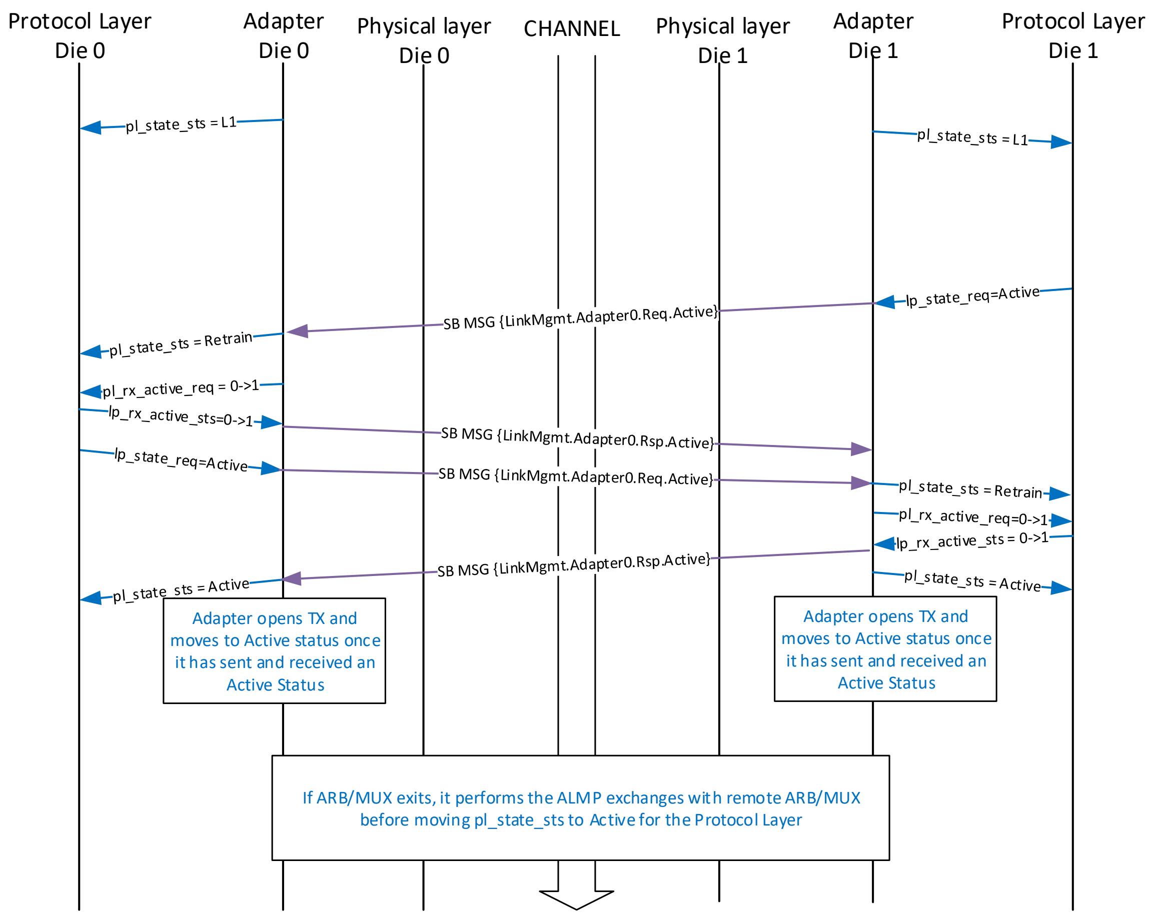

DP Protocol Layer Die 0
DP Adapter Die 0
DP Physical layer Die 0
CHANNEL
UP Physical layer Die 1
UP Adapter Die 1
UP Protocol Layer Die 1
pl_state_sts = Active
pl_state_sts = Active
lp_state_req=Active
lp_state_req=L1
SB MSG {LinkMgmt.Adapter0.Req.L1}
Adapter must finish the pl_stallreq/lp_stallack handshake before sending sideband message
Adapter must wait for 1us to see if Protocol Layer requests PM
SB MSG {LinkMgmt.Adapter0.Rsp.PMNAK}
pl_state_sts = Active.PMNAK
lp_state_req=Active
pl_state_sts = Active

Figure 124. PM Exit Example


Protocol Layer Die 0
Adapter Die 0
Physical layer Die 0
CHANNEL
Physical layer Die 1
Adapter Die 1
Protocol Layer Die 1
pl_state_sts = L1
pl_state_sts = L1
lp_state_req=Active
SB MSG {LinkMgmt.Adapter0.Req.Active}
pl_state_sts = Retrain
pl_rx_active_req = 0->1
lp_rx_active_sts=0->1
SB MSG {LinkMgmt.Adapter0.Rsp.Active}
lp_state_req=Active
SB MSG {LinkMgmt.Adapter0.Req.Active}
pl_state_sts = Retrain
pl_rx_active_req=0->1
lp_rx_active_sts = 0->1
SB MSG {LinkMgmt.Adapter0.Rsp.Active}
pl_state_sts = Active
pl_state_sts = Active
Adapter opens TX and moves to Active status once it has sent and received an Active Status
Adapter opens TX and moves to Active status once it has sent and received an Active Status
If ARB/MUX exits, it performs the ALMP exchanges with remote ARB/MUX before moving pl_state_sts to Active for the Protocol Layer

8.3 Common rules for FDI and RDI

This section covers common set of rules applicable to both FDI and RDI interfaces and cross-interactions between them. Any applicable differences are called out as well. In order to have common terminology for the common set of rules, Upper Layer is used to refer to Adapter for RDI, and Protocol Layer for FDI. Lower Layer is used to refer to Physical Layer for RDI and Adapter for FDI. In the following sections, "UCIe Flit Mode" refers to scenarios where the Retry buffer in the Adapter is being utilized and "UCIe Raw Mode" refers to scenarios where the Retry buffer in the Adapter is not being utilized.

Since Active.PMNAK is a sub-state of Active, all rules that apply for Active are also applicable for Active.PMNAK.

8.3.1 Stallreq/Ack Mechanism

The Stallreq/Ack mechanism is used by the Lower Layer to interrupt the Flit transfers by the Upper Layer at a Flit boundary. On RDI, the Stallreq/Ack mechanism must be used when exiting Active state to Retrain, PM, LinkReset or Disabled states. On FDI, for UCIe Raw Mode, the Stallreq/Ack mechanism must be used when exiting Active state to

Retrain, PM, LinkReset or Disabled states. On FDI, for UCIe Flit Mode, the Stallreq/Ack mechanism must only be used when exiting Active State to a PM state. For other scenarios that exit Active state for UCIe Flit Mode, the Adapter must simply de-assert **pl_trdy** at a Flit boundary before state transition.

The Stallreq/Ack mechanism is mandatory for all FDI and RDI implementations. **lp_stallack** assertion implies that Upper Layer has stalled its pipeline at a Flit aligned boundary.

The **lp_stallreq/pl_stallack** handshake is a four-phase sequence that follows the rules below:

1. The **pl_stallreq** and **lp_stallack** must be de-asserted before domain reset exit.
2. A rising edge on **pl_stallreq** must only occur when **lp_stallack** is de-asserted.
3. A falling edge on **pl_stallreq** must only occur when **lp_stallack** is asserted or when the domain is in reset.
4. A rising edge on **lp_stallack** must only occur when **pl_stallreq** is asserted.
5. A falling edge on **lp_stallack** must only occur when **pl_stallreq** is de-asserted or when domain is in reset.
6. When **lp_stallack** is asserted **lp_valid** and **lp_irdy** must both be de-asserted.
7. While **pl_stallreq** is asserted, any data presented on the interface must be accepted by the physical layer until the rising edge of **lp_stallack**. **pl_trdy** is not required to be asserted consecutively.
8. The logic path between **pl_stallreq** and **lp_stallack** must contain at least one flop to prevent a combinatorial loop.
9. A complete stallreq/stallack handshake is defined as the completion of all four phases: Rising edge on **pl_stallreq**, rising edge on **lp_stallack**, falling edge on **pl_stallreq**, falling edge on **lp_stallack**.
10. It is strongly recommended that Upper Layer implements providing **lp_stallack** on a global free running clock so that it can finish the handshake even if the rest of its logic is clock gated.

To avoid performance penalties, it is recommended that this handshake be completed as quickly as possible while satisfying the above rules.

8.3.2 State Request and Status

Table 116 describes the Requests considered by the Lower layer in each of the interface states. The Upper layer must take into account the interface state status and make the necessary request modifications.

The requests are listed on the Row and the state status is listed in the Column.

The entries in Table 116 denote the following:

- Yes: Indicates that the request is considered for next state transition by the physical layer.
- N/A: Not Applicable
- Ignore: Indicates that the request is ignored and has no effect on the next state transition.

Table 116. Requests Considered in Each LPIF State by Physical Layer

Request (Row) Versus Status (Column) on LPIF	RESET	ACTIVE	L1	LinkReset	RETRAIN	DISABLE	L2	LinkError
NOP	Yes	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore
ACTIVE	Yes[1]	Ignore[2]	Yes	Yes	Yes	Yes	Yes	Yes
DAPM	Ignore	Yes	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore
L1	Ignore	Yes	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore
LinkReset	Yes[1]	Yes	Yes	Ignore	Yes	Ignore	Yes	Ignore
RETRAIN	Ignore	Yes	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore
DISABLE	Yes[1]	Yes	Yes	Yes	Yes	Ignore	Yes	Ignore
L2	Ignore	Yes	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore
LinkError (sideband wire)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

1. Requires request transition from NOP
2. If the Status is Active.PMNAK, then the Lower Layer transitions to Active upon sampling the Active Request.

8.3.2.1 Reset State rules

The Reset State can be entered on de-assertion of interface reset signal or from LinkReset/Disable/LinkError states. In Reset state, the physical layer is permitted to begin its initialization/training process.

The **`pl_state_sts`** is not permitted to exit Reset state until requested by the upper layer. The exit from Reset state is requested by the upper layer by changing the **`lp_state_req`** signal from NOP encoding value to the permitted next state encoding value.

The rules for Reset state transition are as follows:

1. Reset→Active: The lower layer triggers transitions to the Active state upon observing lp_state_req == Reset (NOP) for at least one clock while pl_state_sts is indicating Reset followed by observing lp_state_req == Active. The transition to Active is only completed once the corresponding Active Entry handshakes have completed on the Link. For RDI, it is when the Physical Layer has sent and received an Active Response sideband message to and from the remote Physical Layer respectively. For the Adapter LSM, it is when the Adapter has sent and received an Active Status sideband message to and from the remote Adapter respectively. For the ARB/MUX vLSM, it is when the ARB/MUX has sent and received an Active Status ALMP to and from the remote ARB/MUX respectively.
2. Reset→LinkReset: The lower layer transitions to the LinkReset state upon observing lp_state_req == Reset (NOP) for at least one clock while pl_state_sts is indicating Reset followed by observing lp_state_req == LinkReset OR when requested by remote Link partner through the relevant sideband message. The lower layer is permitted to transition through Active State, and when it does, Active state exit conditions apply.

3. Reset→Disabled: The lower layer transitions to the Disabled state upon observing lp_state_req == Reset (NOP) for at least one clock while pl_state_sts is indicating Reset followed by observing lp_state_req == Disabled OR when requested by the remote Link partner through the relevant sideband message. The lower layer is permitted to transition through Active State, and when it does, Active state exit conditions apply.
4. Reset→LinkError: The lower layer transitions to LinkError based on observing an internal request to move to the LinkError or `lp_linkerror` assertion. For RDI, this transition is permitted if requested by the remote Link partner through the relevant sideband message.

8.3.2.2 Active State rules

The Active state to next state transitions are described below.

The rules for Active State transition are as follows:

1. Active→Retrain: The Lower layer transitions to the Retrain state upon observing lp_state_req == Retrain or due to an internal request to retrain the Link while pl_state_sts == Active. This arc is not applicable for CXL vLSMs exposed on FDI (CXL Flit Mode with Retry in the Adapter).
2. Active→L1: The physical layer transitions to L1 based on observing lp_state_req == L1 while in the Active state, if other conditions of PM entry have also been satisfied.
3. Active→L2: The physical layer transitions to L2 based on observing lp_state_req == L2 while in the Active state, if other conditions of PM entry have also been satisfied.

All of the above transitions are applicable from the Active.PMNAK sub state as well.

section 8.3.2.8 describes the transition from Active to LinkReset, Disable, or LinkError states.

8.3.2.3 PM Entry/Exit rules

Pleaser refer to the PM entry and exit sequences in the RDI and FDI sections.

8.3.2.4 Retrain State rules

Adapter requests Retrain on RDI if any of the following events occur:

1. Software writes to the Retrain bit and the Link is in Active state
2. Number of CRC or parity errors detected crosses a threshold. The specific algorithm for determining this is implementation specific.
3. Protocol Layer requests Retrain (only applicable for Raw Mode)
4. any other implementation specific condition (if applicable)

Physical Layer triggers a Retrain transition on RDI if:

1. Valid framing errors are observed.
2. Remote Physical Layer requests Retrain entry.
3. Adapter requests Retrain

Protocol Layer must not request Retrain on FDI, unless UCIe is operating in Raw mode.

A Retrain transition on RDI must always be propagated to Adapter LSM. Retrain transitions of the UCIe Link are not propagated to CXL vLSMs. Upon Retrain entry, the credit counter for UCIe Retimer (if present) must be reset to the value advertised during initial Link bring up (the value is given by the "Retimer_Credits" Parameter in the {AdvCap.Adapter} sideband message during initial Link bring up). The Retimer must drain or dump any Flits in flight or its internal transport buffers upon entry to Retrain, and it must trigger Retrain of the remote UCIe Link (across the Off-Package Interconnect) as well.

Entry into Retrain state resets power management state for the corresponding state machine, and power management entry if required must be re-initiated after the interface enters Active state. If there was an outstanding PM request that returns PM Response, the corresponding state machine must perform Active Entry handshakes to bring that state machine back to Active.

The rules for Retrain state transition are as follows:

1. Retrain→Active: If Retrain was entered from L1, the lower layer begins Active Entry handshakes upon observing **`lp_state_req`** == Active while the **`pl_state_sts`** == Retrain. If Retrain was entered from Active, the lower layer begins Active Entry handshakes only after observing a NOP-> Active transition on **`lp_state_req`**. Lower layer transitions to Active once the corresponding Active Entry handshakes have completed. Exit from Retrain on RDI requires the Active Entry handshakes to have completed between Physical Layers. Exit from Retrain on FDI must ensure that RDI has moved back to Active, and Active Entry handshakes have successfully completed between Adapters (for the Adapter LSM).
2. Transitional state: The lower layer is permitted to transition to the Active state upon observing **`lp_state_req`** == LinkReset or Disabled while the **`pl_state_sts`** == Retrain. Following the entry into Active the lower layer is permitted to make a transition to the requested state.

section 8.3.2.8 describes Retrain exit to LinkReset, Disable, or LinkError states.

Note: The requirement to wait for NOP->Active transition ensures that the Upper Layer has a way to delay Active transition in case it is waiting for any relevant sideband handshakes to complete (for example the Parity Feature handshake).

8.3.2.5 LinkReset State rules

LinkReset is used for reset flows (HotReset equivalent in PCIe, Protocol Layer must use this to propagate SBR to the device as well) to convey device and/or Link Reset across the UCIe Link.

Adapter triggers LinkReset transition upon observing a LinkReset request from the Protocol Layer, on receiving a sideband message requesting LinkReset entry from the remote Link partner or an implementation specific internal condition (if applicable). Implementations must make best efforts to gracefully drain the Retry buffers when transitioning to LinkReset, however, entry to LinkReset must not timeout on waiting for the Retry buffer to drain. The Protocol Layer and Adapter must drain/flush their pipelines and retry buffer of the Flits for the corresponding Protocol Stack once the FDI state machines have entered LinkReset.

If all the FDI state machines and Adapter LSMs are in LinkReset, the Adapter triggers RDI to enter LinkReset as well.

The rules for LinkReset State transitions are as follows:

1. LinkReset→Reset: The lower layer transitions to the Reset state due to an internal request to move to Reset (example reset pin trigger) or **`lp_state_req`** == Active while **`pl_state_sts`** == LinkReset and all necessary actions with respect to LinkReset have been completed.

2. LinkReset→Disabled The lower layer transitions to Disabled based on observing **`lp_state_req`** == Disabled or due to an internal request to move to Disabled while pl_state_sts == LinkReset.
3. Transitional State: The PHY is permitted to transition through Reset State, and when it does, Reset state exit conditions apply.
4. LinkReset→LinkError: The lower layer transitions to LinkError due to an internal request to move to LinkError or **`lp_linkerror`** assertion while **`pl_state_sts`** == LinkReset.

8.3.2.6 Disabled State rules

Adapter triggers Disabled entry when any of the following events occur:

- Protocol Layer requests entry to Disabled state
- Software writes to the Link Disable bit corresponding to UCIe
- Remote Link partner requests entry to Disabled state through the relevant sideband message
- An implementation specific internal condition (if applicable)

Implementations must make best efforts to gracefully drain the Retry buffers when transitioning to Disabled, however, entry to Disabled must not timeout on waiting for the Retry buffer to drain. The Protocol Layer and Adapter must drain/flush their pipelines and retry buffer of the Flits for the corresponding Protocol Stack once the FDI state machines have entered Disabled.

If all the FDI state machines and Adapter LSMs are in Disabled, the Adapter triggers RDI to enter Disabled as well.

The rules for Disabled State are as follows:

1. Disabled→Reset: The physical layer transitions to the Reset state due to an internal request to move to Reset (example reset pin trigger) or **`lp_state_req`** == Active while **`pl_state_sts`** == Disabled and all necessary actions with respect to Disabled transition have completed.
2. Disabled→LinkError: The physical layer transitions to LinkError due to an internal request to move to LinkError or **`lp_linkerror`** assertion while **`pl_state_sts`** == Disabled.

8.3.2.7 LinkError State rules

Lower layer enters LinkError state when directed via **`lp_linkerror`** signal or due to Internal LinkError conditions. For RDI, the entry is also triggered if the remote Link partner requested LinkError entry through the relevant sideband message. It is not required to complete the stallreq/ack handshake before entering this state. However, for implementations where LinkError state is not a terminal state (terminal implies SoC needs to go through reset flow after reaching LinkError state.), it is expected that software can come and retrain the Link after clearing error status registers, etc.), and the following rules should be followed:

1. If the lower layer decides to perform a **`pl_stallreq/lp_stallack`** handshake, it must make sure to provide **`pl_trdy`** to the upper layer to drain the packets. In cases where there is an uncorrectable internal error in the lower layer these packets could be dropped and not transmitted on the Link.
2. It is required for upper layer to internally clean up the data path even if **`pl_trdy`** is not asserted and it has sampled LinkError on **`pl_state_sts`** for at least one clock cycle.

Lower layer may enter LinkError state due to Internal LinkError requests such as when

1. Encountering uncorrectable errors due to hardware failure or directed by Upper Layer.
2. Remote Link partner requests entry into LinkError. (RDI only)

The rules for LinkError state are as follows:

LinkError→Reset: The physical layer transitions to Reset due to an internal request to move to Reset (example: reset pin assertion, or software clearing the error status bits that triggered the error) or **`lp_state_req`** == Active and **`lp_linkerror`** = 0, while **`pl_state_sts`** == LinkError and minimum residency requirements are met. Physical Layer must implement a minimum residency time in LinkError of 8ms to ensure that the remote Link partner will be forced to enter LinkError due to timeouts (to cover for cases where the LinkError transition happened and sideband was not functional).

8.3.2.8 Common State rules

This section covers some of the common conditions for exit from Active, Retrain, L1, and L2 to LinkReset, Disable and LinkError states. For RDI, PM encoding and rules correspond to L1 in the text below.

The rules are as follows:

1. [Active, Retrain, L1, L2]→LinkReset: The lower layer transitions to LinkReset based on observing **`lp_state_req`** ==LinkReset or due to an internal request to move to LinkReset or the remote Link partner requesting LinkReset over sideband.
2. [Active, Retrain, L1, L2]→Disabled: The physical layer transitions to Disabled based on observing **`lp_state_req`** ==Disabled or due to an internal request to move to Disabled while **`pl_state_sts`** == Active, or the remote Link partner requesting Disabled over sideband.
3. [Active, Retrain, L1, L2]→LinkError: The physical layer transitions RDI to LinkError based on observing an internal request to move to the LinkError or **`lp_linkerror`** assertion, or the remote Link partner requesting LinkError over sideband. RDI must move to LinkError before propagating LinkError to all Adapter LSMs.

From a state machine hierarchy perspective, it is required for Adapter LSM to move to LinkReset, Disabled or LinkError before propagating this to CXL vLSMs. This ensures CXL rules are followed where these states are "non-virtual" from the perspective of CXL vLSMs.

Adapter LSM can transition to LinkReset or Disabled without RDI transitioning to these states. In the case of multi-protocol stacks over the same Physical Link/Adapter, each Protocol can independently enter these states without affecting the other protocol stack on the RDI.

If all the Adapter LSMs have moved to a common state of LinkReset/Disabled or Linkerror, then RDI is taken to the corresponding state. If however, the Adapter LSMs are in different state combinations of LinkError, Disabled or LinkReset, the RDI is moved to the highest priority state. The priority order from highest to lowest is LinkError, Disabled, LinkReset. For a LinkError/LinkReset/Disabed transition on RDI, Physical Layer must initiate the corresponding sideband handshake to transition remote Link partner to the required state. If no response is received from remote Link partner for this message after 8ms, RDI transitions to LinkError.

If RDI moves to a state that is of a higher priority order than the current Adapter LSM, it is required for the Adapter to propagate that to the Adapter LSM using sideband handshakes to ensure the transition with the remote Link partner.

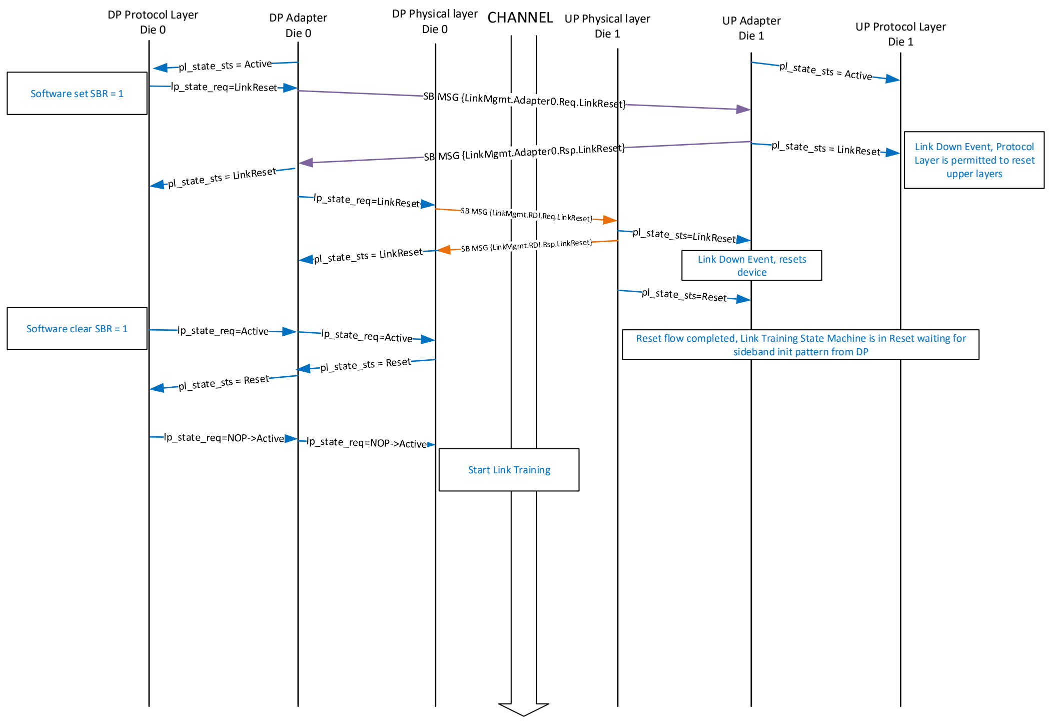
After transition from LinkError/LinkReset/Disable to Reset on RDI, the Physical Layer must not begin training unless it observes a NOP->Active transition on **`lp_state_req`** from the Adapter or Software writes 1b to the Start Link Training bit. The Adapter should not trigger NOP->Active unless it receives this transition from the Protocol Layer or has internally decided to bring the Link Up. The Adapter must trigger this on RDI if the Protocol Layer has triggered this even if **`pl_inband_pres`** = 0. Thus, if the Protocol Layer is waiting for software intervention and wants to hold back the Link from training, it can delay the NOP->Active trigger on FDI. Upper Layers are permitted to transition **`lp_state_req`** back to NOP after giving the NOP->Active trigger in order to clock gate while waiting for **`pl_inband_pres`** to assert.

8.3.3 Example flow diagrams

8.3.3.1 LinkReset Entry and Exit

Figure 125 shows an example flow for LinkReset entry and exit. In the multi-protocol stack scenario, it is permitted for each protocol stack to independently transition to LinkReset. RDI is only transitioned to LinkReset if all the corresponding Adapter LSMs are in LinkReset.

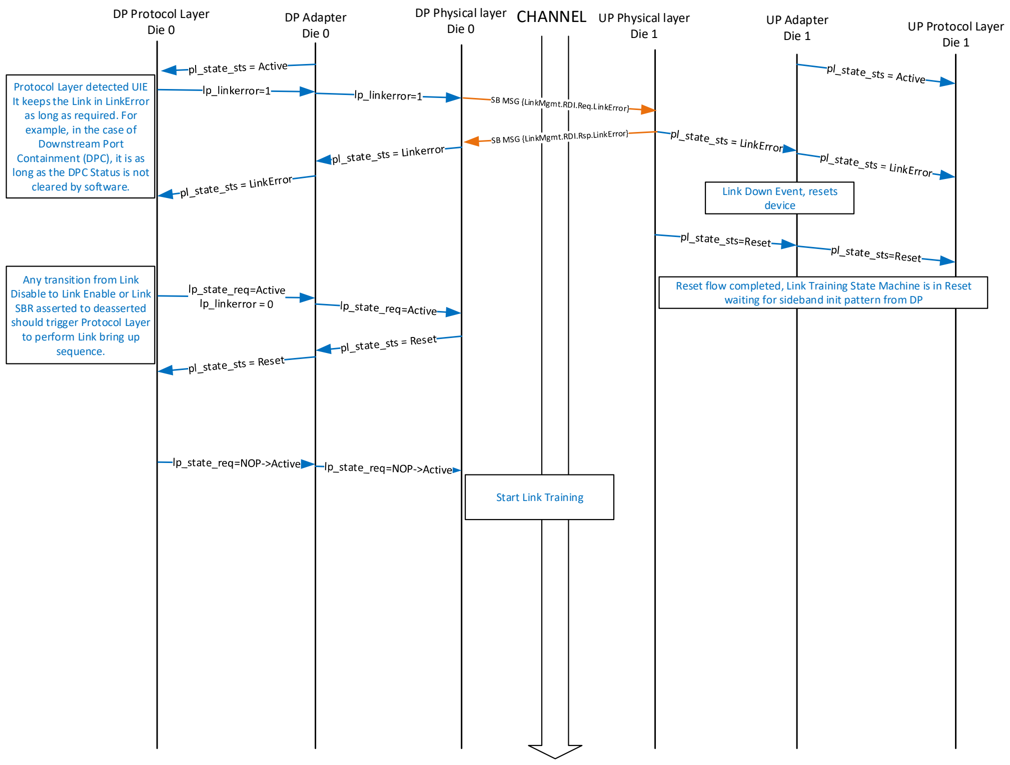
Figure 125. LinkReset Example



8.3.3.2 LinkError

Figure 126 shows an example of LinkError entry and exit when the Protocol Layer detected an uncorrectable internal error.

Figure 126. LinkError example


DP Protocol Layer Die 0
DP Adapter Die 0
DP Physical layer Die 0
CHANNEL
UP Physical layer Die 1
UP Adapter Die 1
UP Protocol Layer Die 1
pl_state_sts = Active
pl_state_sts = Active
Protocol Layer detected UIE It keeps the Link in LinkError as long as required. For example, in the case of Downstream Port Containment (DPC), it is as long as the DPC Status is not cleared by software.
lp_linkerror=1
lp_linkerror=1
SB MSG {LinkMgmt.RDI.Req.LinkError}
SB MSG {LinkMgmt.RDI.Rsp.LinkError}
pl_state_sts = LinkError
pl_state_sts = LinkError
pl_state_sts = Linkerror
pl_state_sts = LinkError
Link Down Event, resets device
pl_state_sts=Reset
pl_state_sts=Reset
Any transition from Link Disable to Link Enable or Link SBR asserted to deasserted should trigger Protocol Layer to perform Link bring up sequence.
lp_state_req=Active
lp_linkerror = 0
lp_state_req=Active
Reset flow completed, Link Training State Machine is in Reset waiting for sideband init pattern from DP
pl_state_sts = Reset
pl_state_sts = Reset
lp_state_req=NOP->Active
lp_state_req=NOP->Active
Start Link Training

9.0 Compliance and Debug

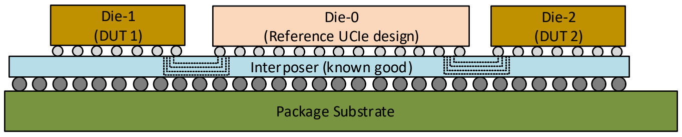
UCIe provisions for a system setup to perform interoperability testing and debug of the different UCIe components. The goal of interoperability testing is to validate the supported features of a Device Under Test (DUT) against a known good reference UCIe implementation. Eventually, the different components of UCIe (Physical Layer, Adapter, Protocol Layer) can have their own test suite for interoperability testing.

The system setup for interoperability testing or debug has the following components:

- Reference UCIe design: known good UCIe implementation
- Device Under Test (DUT): One or more DUTs that will be tested with the reference design. It is required that these have cleared the testing requirements of die sort and class before they are brought for interoperability testing.
- In the case of Advanced Package configuration, a known good silicon bridge or interposer to connect the different die

The above components are integrated together in a test package (Figure 127) which is then used for running interoperability testing or debug tests. It is expected that multiple DUTs will be part of a given test package to amortize the packaging costs.

Figure 127. Example of a test package for Advanced Package configuration



UCIe formalized a sideband Link that is separate from main-band data transfers. This Link works at relatively lower speeds and is required to be functional for any UCIe testing/enabling. The sideband plays a critical role for enabling interoperability testing and debug modes by allowing test or debug software to access registers from different UCIe components (Physical Layer or Adapter). Some capabilities and tests that will be enabled through accesses via sideband:

- Training progression and monitoring: Reference UCIe design would be able to monitor the state progressions and parameters of DUT to ensure specification compliance
- Adapter Parameter negotiation: Reference UCIe design could enable exercising different arcs of the Flit format negotiation tree to ensure the mandatory Flit formats are supported for a given protocol
- For CXL devices, they would be enabled and strongly recommended to leverage the coherency compliance algorithms specified in the CXL specification
- An optional capability could be specified for implementing on-die Flit tracing
- On the electrical side, the following capabilities are being considered for debug/ interoperability testing:
  - — Voltage and Timing Eye Margin testing
  - — Low frequency Receiver ViH/ViL voltage sensitivity testing
  - — Pattern generation and pattern checking
  - — Loopback

- — View pins (could be shared with the rest of the SoC)
- — I/O scan, BIST etc.

Note: The relevant registers and test setup will be specified in Rev 1.1

Appendices

A CXL/PCIe Register applicability to UCIe

A.1 CXL Registers applicability to UCIe

All CXL specifications defined DVSEC apply in the context of UCIe fully when operating in Raw mode. When operating in non-raw mode, a few register definitions need to be reinterpreted in the context of UCIe. See below for details. Note that regardless of the raw or non-raw mode, device/port configurations with UCIe 1.1 compliance is not permitted as was discussed in the registers chapter.

Table 117. CXL Registers for UCIe devices

Register Block	Register	**Bits**	**Comments**
DVSEC Capability	DVSEC Flex Bus Port Control	Bits 3,4	See next row for how these bits are handled
	DVSEC Flex Bus Port Status	Bits 3, 4	This bit just mirrors bit 3, 4 in Flex bus port control register, to mimic legacy behavior.
		Bits 11, 12	Hardwire to 0
	From 0x14-1Fh		N/A

A.2 PCIe Register applicability to UCIe

All PCIe specifications defined DVSEC apply in the context of UCIe as well. There are a few Link and PHY layer registers/bits though that are N/A or need to be reinterpreted in the context of UCIe. They are listed below.

Figure 128. PCIe Registers for UCIe devices

Register Block	Register	Bits	Comments
PCIe capability	PCI Express Capabilities Register	8	Slot implemented – set to 0. And hence follow rules for implementing other slot related registers/bits at various locations in the PCIe capability register set.
	Device capabilities Register	8:6	N/A and can be set to any value
	Link Capabilities Register	3:0	Max Link Speed: Set to 0011 indicating 8GT/s
		9:4	Max Link Width: 01 0000b, indicating x16
		11:10	ASPM support: 01/11b encodings disallowed
		14:12	N/A
		17:15	L1 Exit Latency: Devices/Ports must set this bit based on whether they are connected to a retimer or not, and also the retimer based exit latency might not be known at design time as well. To assist with this, these bits need to be made Hwinit from a device/port perspective so system FW can set this at boot time based on the specific retimer based latencies.
		18	N/A and hardwire to 0
	Link Control Register	6	HW ignores whats written here but follow any base spec rules for bit attributes.
		7	HW ignores whats written here but follow any base spec rules for bit attributes.
		8	Set to RO 0
		9	Set to RO 0
		10, 11, 12	HW ignores whats written in these bits but follows any base spec rules for bit attributes.
	Link Status Register	3:0	Current Link speed: Set to 0011 indicating 8GT/s
		9:4	Negotiated Link width: x16
		15	Hardwire to 0
	Link Capabilities 2 Register	7:1	Set to 0000111
		15:9	Set to 0x0

PCIe Capability	Link Capabilities 2 Register	22:16	Set to 0x0
		24:23	Set to 0x0 just to appear compliant
	Link Control 2 Register	3:0	Target Link speed: Writes to this register are ignored by UCIe hardware, but HW follows the base spec rules for bit attributes
		4	HW ignores whats written in this bit but follows any base spec rules for bit attributes.
		5	HW autonomous speed disable – Set to RO 0
		15:6	N/A for UCIe. HW should follow base spec rules for register bit attributes.
	Link Status 2 Register	9:0	Set to RO 0
PCIe Extended Capability	Secondary PCI Express Extended Capability	All	Implement per the base spec, but HW ignores all commands from SW and also sets all equalization control registry entries to 0.

A.3 PCIe/CXL registers that need to be part of D2D

- PCIe Link Control Register
  - Bits 1:0, 5, 4, 13 are relevant for D2D operation
- CXL DVSEC Flex Bus Port Received Modified TS Data Phase1 Register
- CXL DVSEC Flex Bus Port Control
- CXL DVSEC Flex Bus Port Status
- CXL ARB/MUX registers

B CRC Code Generator

```
module crc_gen (

    input logic [1023:0] data_in,

    output logic [15:0] crc_out

);

always_comb begin

        crc_out[15] =
data_in[0]^data_in[1]^data_in[3]^data_in[4]^data_in[17]^data_in[18]^data_
in[19]^data_in[21]^data_in[23]^data_in[25]^data_in[27]^data_in[29]^data_i
n[31]^data_in[32]^data_in[33]^data_in[34]^data_in[37]^data_in[38]^data_in
[41]^data_in[42]^data_in[45]^data_in[46]^data_in[47]^data_in[48]^data_in[
49]^data_in[51]^data_in[52]^data_in[53]^data_in[55]^data_in[56]^data_in[5
7]^data_in[59]^data_in[60]^data_in[61]^data_in[62]^data_in[63]^data_in[64
]^data_in[69]^data_in[70]^data_in[71]^data_in[72]^data_in[80]^data_in[82]
^data_in[88]^data_in[90]^data_in[92]^data_in[94]^data_in[95]^data_in[98]^
data_in[100]^data_in[102]^data_in[103]^data_in[106]^data_in[107]^data_in[
111]^data_in[114]^data_in[115]^data_in[119]^data_in[123]^data_in[125]^dat
a_in[126]^data_in[128]^data_in[129]^data_in[130]^data_in[132]^data_in[135
]^data_in[137]^data_in[138]^data_in[142]^data_in[143]^data_in[144]^data_i
n[145]^data_in[148]^data_in[151]^data_in[152]^data_in[153]^data_in[155]^d
ata_in[161]^data_in[164]^data_in[169]^data_in[170]^data_in[172]^data_in[1
74]^data_in[177]^data_in[180]^data_in[182]^data_in[186]^data_in[187]^data
_in[190]^data_in[193]^data_in[196]^data_in[197]^data_in[199]^data_in[203]
^data_in[206]^data_in[209]^data_in[213]^data_in[214]^data_in[216]^data_in
[219]^data_in[222]^data_in[225]^data_in[227]^data_in[228]^data_in[229]^da
ta_in[232]^data_in[235]^data_in[238]^data_in[241]^data_in[242]^data_in[24
3]^data_in[244]^data_in[246]^data_in[247]^data_in[249]^data_in[250]^data_
in[252]^data_in[253]^data_in[255]^data_in[256]^data_in[257]^data_in[258]^
data_in[259]^data_in[263]^data_in[264]^data_in[265]^data_in[269]^data_in[
270]^data_in[271]^data_in[272]^data_in[273]^data_in[274]^data_in[276]^dat
a_in[281]^data_in[283]^data_in[284]^data_in[285]^data_in[286]^data_in[287
]^data_in[288]^data_in[289]^data_in[292]^data_in[294]^data_in[297]^data_i
n[298]^data_in[299]^data_in[300]^data_in[301]^data_in[302]^data_in[303]^d
ata_in[304]^data_in[306]^data_in[307]^data_in[310]^data_in[320]^data_in[3
21]^data_in[322]^data_in[324]^data_in[325]^data_in[327]^data_in[329]^data
_in[331]^data_in[333]^data_in[338]^data_in[339]^data_in[340]^data_in[343]
^data_in[344]^data_in[347]^data_in[348]^data_in[350]^data_in[352]^data_in
[353]^data_in[354]^data_in[355]^data_in[357]^data_in[358]^data_in[359]^da
ta_in[361]^data_in[362]^data_in[363]^data_in[366]^data_in[367]^data_in[36
8]^data_in[369]^data_in[370]^data_in[375]^data_in[376]^data_in[377]^data_
in[378]^data_in[380]^data_in[381]^data_in[382]^data_in[383]^data_in[384]^
data_in[385]^data_in[387]^data_in[389]^data_in[390]^data_in[391]^data_in[
392]^data_in[393]^data_in[401]^data_in[402]^data_in[409]^data_in[411]^dat
a_in[413]^data_in[415]^data_in[416]^data_in[417]^data_in[419]^data_in[421
]^data_in[423]^data_in[424]^data_in[427]^data_in[428]^data_in[433]^data_i
n[434]^data_in[437]^data_in[438]^data_in[439]^data_in[441]^data_in[442]^d
ata_in[443]^data_in[445]^data_in[447]^data_in[448]^data_in[449]^data_in[4
51]^data_in[452]^data_in[453]^data_in[454]^data_in[459]^data_in[460]^data
_in[465]^data_in[466]^data_in[467]^data_in[468]^data_in[469]^data_in[471]
^data_in[473]^data_in[474]^data_in[475]^data_in[477]^data_in[479]^data_in
[480]^data_in[481]^data_in[482]^data_in[483]^data_in[484]^data_in[487]^da
ta_in[488]^data_in[489]^data_in[490]^data_in[493]^data_in[494]^data_in[49
5]^data_in[496]^data_in[497]^data_in[498]^data_in[499]^data_in[501]^data_
```

```
in[502]^data_in[503]^data_in[504]^data_in[505]^data_in[507]^data_in[508]^
data_in[509]^data_in[510]^data_in[511]^data_in[512]^data_in[513]^data_in[
514]^data_in[521]^data_in[522]^data_in[523]^data_in[524]^data_in[525]^dat
a_in[526]^data_in[527]^data_in[528]^data_in[529]^data_in[531]^data_in[533
]^data_in[535]^data_in[536]^data_in[537]^data_in[538]^data_in[539]^data_i
n[540]^data_in[541]^data_in[542]^data_in[543]^data_in[544]^data_in[547]^d
ata_in[548]^data_in[560]^data_in[564]^data_in[566]^data_in[568]^data_in[5
70]^data_in[572]^data_in[574]^data_in[575]^data_in[577]^data_in[580]^data
_in[581]^data_in[584]^data_in[585]^data_in[588]^data_in[589]^data_in[590]
^data_in[593]^data_in[597]^data_in[601]^data_in[606]^data_in[609]^data_in
[611]^data_in[612]^data_in[614]^data_in[617]^data_in[619]^data_in[622]^da
ta_in[625]^data_in[626]^data_in[627]^data_in[630]^data_in[633]^data_in[63
4]^data_in[636]^data_in[637]^data_in[639]^data_in[640]^data_in[641]^data_
in[642]^data_in[644]^data_in[645]^data_in[647]^data_in[648]^data_in[649]^
data_in[653]^data_in[654]^data_in[655]^data_in[656]^data_in[657]^data_in[
661]^data_in[662]^data_in[663]^data_in[664]^data_in[666]^data_in[673]^dat
a_in[675]^data_in[676]^data_in[677]^data_in[678]^data_in[679]^data_in[682
]^data_in[684]^data_in[686]^data_in[689]^data_in[690]^data_in[691]^data_i
n[692]^data_in[693]^data_in[694]^data_in[696]^data_in[697]^data_in[700]^d
ata_in[701]^data_in[703]^data_in[704]^data_in[705]^data_in[706]^data_in[7
07]^data_in[708]^data_in[709]^data_in[713]^data_in[717]^data_in[718]^data
_in[719]^data_in[720]^data_in[721]^data_in[722]^data_in[723]^data_in[724]
^data_in[726]^data_in[729]^data_in[731]^data_in[732]^data_in[733]^data_in
[734]^data_in[735]^data_in[736]^data_in[737]^data_in[738]^data_in[739]^da
ta_in[742]^data_in[745]^data_in[746]^data_in[747]^data_in[748]^data_in[74
9]^data_in[750]^data_in[751]^data_in[752]^data_in[753]^data_in[754]^data_
in[756]^data_in[757]^data_in[759]^data_in[760]^data_in[761]^data_in[762]^
data_in[763]^data_in[764]^data_in[765]^data_in[766]^data_in[767]^data_in[
768]^data_in[769]^data_in[773]^data_in[774]^data_in[775]^data_in[776]^dat
a_in[777]^data_in[778]^data_in[779]^data_in[780]^data_in[781]^data_in[782
]^data_in[783]^data_in[784]^data_in[786]^data_in[800]^data_in[801]^data_i
n[803]^data_in[805]^data_in[807]^data_in[809]^data_in[811]^data_in[813]^d
ata_in[817]^data_in[818]^data_in[821]^data_in[822]^data_in[825]^data_in[8
26]^data_in[829]^data_in[831]^data_in[832]^data_in[833]^data_in[835]^data
_in[836]^data_in[837]^data_in[839]^data_in[840]^data_in[841]^data_in[843]
^data_in[844]^data_in[849]^data_in[850]^data_in[851]^data_in[852]^data_in
[857]^data_in[858]^data_in[859]^data_in[861]^data_in[863]^data_in[864]^da
ta_in[865]^data_in[866]^data_in[867]^data_in[869]^data_in[871]^data_in[87
2]^data_in[873]^data_in[874]^data_in[877]^data_in[878]^data_in[879]^data_
in[880]^data_in[881]^data_in[882]^data_in[885]^data_in[886]^data_in[887]^
data_in[888]^data_in[889]^data_in[891]^data_in[892]^data_in[893]^data_in[
894]^data_in[895]^data_in[896]^data_in[897]^data_in[899]^data_in[900]^dat
a_in[901]^data_in[902]^data_in[903]^data_in[904]^data_in[913]^data_in[914
]^data_in[915]^data_in[916]^data_in[917]^data_in[918]^data_in[919]^data_i
n[921]^data_in[923]^data_in[925]^data_in[927]^data_in[928]^data_in[929]^d
ata_in[930]^data_in[931]^data_in[932]^data_in[933]^data_in[934]^data_in[9
37]^data_in[938]^data_in[941]^data_in[942]^data_in[943]^data_in[944]^data
_in[945]^data_in[946]^data_in[947]^data_in[948]^data_in[949]^data_in[951]
^data_in[952]^data_in[953]^data_in[955]^data_in[956]^data_in[957]^data_in
[958]^data_in[959]^data_in[960]^data_in[961]^data_in[962]^data_in[963]^da
ta_in[964]^data_in[969]^data_in[970]^data_in[971]^data_in[972]^data_in[97
3]^data_in[974]^data_in[975]^data_in[976]^data_in[977]^data_in[978]^data_
in[979]^data_in[981]^data_in[983]^data_in[984]^data_in[985]^data_in[986]^
data_in[987]^data_in[988]^data_in[989]^data_in[990]^data_in[991]^data_in[
992]^data_in[993]^data_in[994]^data_in[997]^data_in[998]^data_in[999]^dat
a_in[1000]^data_in[1001]^data_in[1002]^data_in[1003]^data_in[1004]^data_i
n[1005]^data_in[1006]^data_in[1007]^data_in[1008]^data_in[1009]^data_in[1
011]^data_in[1012]^data_in[1013]^data_in[1014]^data_in[1015]^data_in[1016
]^data_in[1017]^data_in[1018]^data_in[1019]^data_in[1020]^data_in[1021]^d
ata_in[1022]^data_in[1023];
```

```
    crc_out[14] =
data_in[2]^data_in[3]^data_in[5]^data_in[17]^data_in[20]^data_in[21]^data
_in[22]^data_in[23]^data_in[24]^data_in[25]^data_in[26]^data_in[27]^data_
in[28]^data_in[29]^data_in[30]^data_in[31]^data_in[35]^data_in[37]^data_i
n[39]^data_in[41]^data_in[43]^data_in[45]^data_in[50]^data_in[51]^data_in
[54]^data_in[55]^data_in[58]^data_in[59]^data_in[65]^data_in[69]^data_in[
73]^data_in[80]^data_in[81]^data_in[82]^data_in[83]^data_in[88]^data_in[8
9]^data_in[90]^data_in[91]^data_in[92]^data_in[93]^data_in[94]^data_in[96
]^data_in[98]^data_in[99]^data_in[100]^data_in[101]^data_in[102]^data_in[
104]^data_in[106]^data_in[108]^data_in[111]^data_in[112]^data_in[114]^dat
a_in[116]^data_in[119]^data_in[120]^data_in[123]^data_in[124]^data_in[125
]^data_in[127]^data_in[128]^data_in[131]^data_in[132]^data_in[133]^data_i
n[135]^data_in[136]^data_in[137]^data_in[139]^data_in[142]^data_in[146]^d
ata_in[148]^data_in[149]^data_in[151]^data_in[154]^data_in[155]^data_in[1
56]^data_in[161]^data_in[162]^data_in[164]^data_in[165]^data_in[169]^data
_in[171]^data_in[172]^data_in[173]^data_in[174]^data_in[175]^data_in[177]
^data_in[178]^data_in[180]^data_in[181]^data_in[182]^data_in[183]^data_in
[186]^data_in[188]^data_in[190]^data_in[191]^data_in[193]^data_in[194]^da
ta_in[196]^data_in[198]^data_in[199]^data_in[200]^data_in[203]^data_in[20
4]^data_in[206]^data_in[207]^data_in[209]^data_in[210]^data_in[213]^data_
in[215]^data_in[216]^data_in[217]^data_in[219]^data_in[220]^data_in[222]^
data_in[223]^data_in[225]^data_in[226]^data_in[227]^data_in[230]^data_in[
232]^data_in[233]^data_in[235]^data_in[236]^data_in[238]^data_in[239]^dat
a_in[241]^data_in[245]^data_in[246]^data_in[248]^data_in[249]^data_in[251
]^data_in[252]^data_in[254]^data_in[255]^data_in[260]^data_in[263]^data_i
n[266]^data_in[269]^data_in[275]^data_in[276]^data_in[277]^data_in[281]^d
ata_in[282]^data_in[283]^data_in[290]^data_in[292]^data_in[293]^data_in[2
94]^data_in[295]^data_in[297]^data_in[305]^data_in[306]^data_in[308]^data
_in[310]^data_in[311]^data_in[320]^data_in[323]^data_in[324]^data_in[326]
^data_in[327]^data_in[328]^data_in[329]^data_in[330]^data_in[331]^data_in
[332]^data_in[333]^data_in[334]^data_in[338]^data_in[341]^data_in[343]^da
ta_in[345]^data_in[347]^data_in[349]^data_in[350]^data_in[351]^data_in[35
2]^data_in[356]^data_in[357]^data_in[360]^data_in[361]^data_in[364]^data_
in[366]^data_in[371]^data_in[375]^data_in[379]^data_in[380]^data_in[386]^
data_in[387]^data_in[388]^data_in[389]^data_in[394]^data_in[401]^data_in[
403]^data_in[409]^data_in[410]^data_in[411]^data_in[412]^data_in[413]^dat
a_in[414]^data_in[415]^data_in[418]^data_in[419]^data_in[420]^data_in[421
]^data_in[422]^data_in[423]^data_in[425]^data_in[427]^data_in[429]^data_i
n[433]^data_in[435]^data_in[437]^data_in[440]^data_in[441]^data_in[444]^d
ata_in[445]^data_in[446]^data_in[447]^data_in[450]^data_in[451]^data_in[4
55]^data_in[459]^data_in[461]^data_in[465]^data_in[470]^data_in[471]^data
_in[472]^data_in[473]^data_in[476]^data_in[477]^data_in[478]^data_in[479]
^data_in[485]^data_in[487]^data_in[491]^data_in[493]^data_in[500]^data_in
[501]^data_in[506]^data_in[507]^data_in[515]^data_in[521]^data_in[530]^da
ta_in[531]^data_in[532]^data_in[533]^data_in[534]^data_in[535]^data_in[54
5]^data_in[547]^data_in[549]^data_in[560]^data_in[561]^data_in[564]^data_
in[565]^data_in[566]^data_in[567]^data_in[568]^data_in[569]^data_in[570]^
data_in[571]^data_in[572]^data_in[573]^data_in[574]^data_in[576]^data_in[
577]^data_in[578]^data_in[580]^data_in[582]^data_in[584]^data_in[586]^dat
a_in[588]^data_in[591]^data_in[593]^data_in[594]^data_in[597]^data_in[598
]^data_in[601]^data_in[602]^data_in[606]^data_in[607]^data_in[609]^data_i
n[610]^data_in[611]^data_in[613]^data_in[614]^data_in[615]^data_in[617]^d
ata_in[618]^data_in[619]^data_in[620]^data_in[622]^data_in[623]^data_in[6
25]^data_in[628]^data_in[630]^data_in[631]^data_in[633]^data_in[635]^data
_in[636]^data_in[638]^data_in[639]^data_in[643]^data_in[644]^data_in[646]
^data_in[647]^data_in[650]^data_in[653]^data_in[658]^data_in[661]^data_in
[665]^data_in[666]^data_in[667]^data_in[673]^data_in[674]^data_in[675]^da
ta_in[680]^data_in[682]^data_in[683]^data_in[684]^data_in[685]^data_in[68
```

```
6]^data_in[687]^data_in[689]^data_in[695]^data_in[696]^data_in[698]^data_
in[700]^data_in[702]^data_in[703]^data_in[710]^data_in[713]^data_in[714]^
data_in[717]^data_in[725]^data_in[726]^data_in[727]^data_in[729]^data_in[
730]^data_in[731]^data_in[740]^data_in[742]^data_in[743]^data_in[745]^dat
a_in[755]^data_in[756]^data_in[758]^data_in[759]^data_in[770]^data_in[773
]^data_in[785]^data_in[786]^data_in[787]^data_in[800]^data_in[802]^data_i
n[803]^data_in[804]^data_in[805]^data_in[806]^data_in[807]^data_in[808]^d
ata_in[809]^data_in[810]^data_in[811]^data_in[812]^data_in[813]^data_in[8
14]^data_in[817]^data_in[819]^data_in[821]^data_in[823]^data_in[825]^data
_in[827]^data_in[829]^data_in[830]^data_in[831]^data_in[834]^data_in[835]
^data_in[838]^data_in[839]^data_in[842]^data_in[843]^data_in[845]^data_in
[849]^data_in[853]^data_in[857]^data_in[860]^data_in[861]^data_in[862]^da
ta_in[863]^data_in[868]^data_in[869]^data_in[870]^data_in[871]^data_in[87
5]^data_in[877]^data_in[883]^data_in[885]^data_in[890]^data_in[891]^data_
in[898]^data_in[899]^data_in[905]^data_in[913]^data_in[920]^data_in[921]^
data_in[922]^data_in[923]^data_in[924]^data_in[925]^data_in[926]^data_in[
927]^data_in[935]^data_in[937]^data_in[939]^data_in[941]^data_in[950]^dat
a_in[951]^data_in[954]^data_in[955]^data_in[965]^data_in[969]^data_in[980
]^data_in[981]^data_in[982]^data_in[983]^data_in[995]^data_in[997]^data_i
n[1010]^data_in[1011];



    crc_out[13] =
data_in[3]^data_in[4]^data_in[6]^data_in[18]^data_in[21]^data_in[22]^data
_in[23]^data_in[24]^data_in[25]^data_in[26]^data_in[27]^data_in[28]^data_
in[29]^data_in[30]^data_in[31]^data_in[32]^data_in[36]^data_in[38]^data_i
n[40]^data_in[42]^data_in[44]^data_in[46]^data_in[51]^data_in[52]^data_in
[55]^data_in[56]^data_in[59]^data_in[60]^data_in[66]^data_in[70]^data_in[
74]^data_in[81]^data_in[82]^data_in[83]^data_in[84]^data_in[89]^data_in[9
0]^data_in[91]^data_in[92]^data_in[93]^data_in[94]^data_in[95]^data_in[97
]^data_in[99]^data_in[100]^data_in[101]^data_in[102]^data_in[103]^data_in
[105]^data_in[107]^data_in[109]^data_in[112]^data_in[113]^data_in[115]^da
ta_in[117]^data_in[120]^data_in[121]^data_in[124]^data_in[125]^data_in[12
6]^data_in[128]^data_in[129]^data_in[132]^data_in[133]^data_in[134]^data_
in[136]^data_in[137]^data_in[138]^data_in[140]^data_in[143]^data_in[147]^
data_in[149]^data_in[150]^data_in[152]^data_in[155]^data_in[156]^data_in[
157]^data_in[162]^data_in[163]^data_in[165]^data_in[166]^data_in[170]^dat
a_in[172]^data_in[173]^data_in[174]^data_in[175]^data_in[176]^data_in[178
]^data_in[179]^data_in[181]^data_in[182]^data_in[183]^data_in[184]^data_i
n[187]^data_in[189]^data_in[191]^data_in[192]^data_in[194]^data_in[195]^d
ata_in[197]^data_in[199]^data_in[200]^data_in[201]^data_in[204]^data_in[2
05]^data_in[207]^data_in[208]^data_in[210]^data_in[211]^data_in[214]^data
_in[216]^data_in[217]^data_in[218]^data_in[220]^data_in[221]^data_in[223]
^data_in[224]^data_in[226]^data_in[227]^data_in[228]^data_in[231]^data_in
[233]^data_in[234]^data_in[236]^data_in[237]^data_in[239]^data_in[240]^da
ta_in[242]^data_in[246]^data_in[247]^data_in[249]^data_in[250]^data_in[25
2]^data_in[253]^data_in[255]^data_in[256]^data_in[261]^data_in[264]^data_
in[267]^data_in[270]^data_in[276]^data_in[277]^data_in[278]^data_in[282]^
data_in[283]^data_in[284]^data_in[291]^data_in[293]^data_in[294]^data_in[
295]^data_in[296]^data_in[298]^data_in[306]^data_in[307]^data_in[309]^dat
a_in[311]^data_in[312]^data_in[321]^data_in[324]^data_in[325]^data_in[327
]^data_in[328]^data_in[329]^data_in[330]^data_in[331]^data_in[332]^data_i
n[333]^data_in[334]^data_in[335]^data_in[339]^data_in[342]^data_in[344]^d
ata_in[346]^data_in[348]^data_in[350]^data_in[351]^data_in[352]^data_in[3
53]^data_in[357]^data_in[358]^data_in[361]^data_in[362]^data_in[365]^data
_in[367]^data_in[372]^data_in[376]^data_in[380]^data_in[381]^data_in[387]
^data_in[388]^data_in[389]^data_in[390]^data_in[395]^data_in[402]^data_in
[404]^data_in[410]^data_in[411]^data_in[412]^data_in[413]^data_in[414]^da
ta_in[415]^data_in[416]^data_in[419]^data_in[420]^data_in[421]^data_in[42
```

```
2]^data_in[423]^data_in[424]^data_in[426]^data_in[428]^data_in[430]^data_
in[434]^data_in[436]^data_in[438]^data_in[441]^data_in[442]^data_in[445]^
data_in[446]^data_in[447]^data_in[448]^data_in[451]^data_in[452]^data_in[
456]^data_in[460]^data_in[462]^data_in[466]^data_in[471]^data_in[472]^dat
a_in[473]^data_in[474]^data_in[477]^data_in[478]^data_in[479]^data_in[480
]^data_in[486]^data_in[488]^data_in[492]^data_in[494]^data_in[501]^data_i
n[502]^data_in[507]^data_in[508]^data_in[516]^data_in[522]^data_in[531]^d
ata_in[532]^data_in[533]^data_in[534]^data_in[535]^data_in[536]^data_in[5
46]^data_in[548]^data_in[550]^data_in[561]^data_in[562]^data_in[565]^data
_in[566]^data_in[567]^data_in[568]^data_in[569]^data_in[570]^data_in[571]
^data_in[572]^data_in[573]^data_in[574]^data_in[575]^data_in[577]^data_in
[578]^data_in[579]^data_in[581]^data_in[583]^data_in[585]^data_in[587]^da
ta_in[589]^data_in[592]^data_in[594]^data_in[595]^data_in[598]^data_in[59
9]^data_in[602]^data_in[603]^data_in[607]^data_in[608]^data_in[610]^data_
in[611]^data_in[612]^data_in[614]^data_in[615]^data_in[616]^data_in[618]^
data_in[619]^data_in[620]^data_in[621]^data_in[623]^data_in[624]^data_in[
626]^data_in[629]^data_in[631]^data_in[632]^data_in[634]^data_in[636]^dat
a_in[637]^data_in[639]^data_in[640]^data_in[644]^data_in[645]^data_in[647
]^data_in[648]^data_in[651]^data_in[654]^data_in[659]^data_in[662]^data_i
n[666]^data_in[667]^data_in[668]^data_in[674]^data_in[675]^data_in[676]^d
ata_in[681]^data_in[683]^data_in[684]^data_in[685]^data_in[686]^data_in[6
87]^data_in[688]^data_in[690]^data_in[696]^data_in[697]^data_in[699]^data
_in[701]^data_in[703]^data_in[704]^data_in[711]^data_in[714]^data_in[715]
^data_in[718]^data_in[726]^data_in[727]^data_in[728]^data_in[730]^data_in
[731]^data_in[732]^data_in[741]^data_in[743]^data_in[744]^data_in[746]^da
ta_in[756]^data_in[757]^data_in[759]^data_in[760]^data_in[771]^data_in[77
4]^data_in[786]^data_in[787]^data_in[788]^data_in[801]^data_in[803]^data_
in[804]^data_in[805]^data_in[806]^data_in[807]^data_in[808]^data_in[809]^
data_in[810]^data_in[811]^data_in[812]^data_in[813]^data_in[814]^data_in[
815]^data_in[818]^data_in[820]^data_in[822]^data_in[824]^data_in[826]^dat
a_in[828]^data_in[830]^data_in[831]^data_in[832]^data_in[835]^data_in[836
]^data_in[839]^data_in[840]^data_in[843]^data_in[844]^data_in[846]^data_i
n[850]^data_in[854]^data_in[858]^data_in[861]^data_in[862]^data_in[863]^d
ata_in[864]^data_in[869]^data_in[870]^data_in[871]^data_in[872]^data_in[8
76]^data_in[878]^data_in[884]^data_in[886]^data_in[891]^data_in[892]^data
_in[899]^data_in[900]^data_in[906]^data_in[914]^data_in[921]^data_in[922]
^data_in[923]^data_in[924]^data_in[925]^data_in[926]^data_in[927]^data_in
[928]^data_in[936]^data_in[938]^data_in[940]^data_in[942]^data_in[951]^da
ta_in[952]^data_in[955]^data_in[956]^data_in[966]^data_in[970]^data_in[98
1]^data_in[982]^data_in[983]^data_in[984]^data_in[996]^data_in[998]^data_
in[1011]^data_in[1012];



    crc_out[12] =
data_in[4]^data_in[5]^data_in[7]^data_in[19]^data_in[22]^data_in[23]^data
_in[24]^data_in[25]^data_in[26]^data_in[27]^data_in[28]^data_in[29]^data_
in[30]^data_in[31]^data_in[32]^data_in[33]^data_in[37]^data_in[39]^data_i
n[41]^data_in[43]^data_in[45]^data_in[47]^data_in[52]^data_in[53]^data_in
[56]^data_in[57]^data_in[60]^data_in[61]^data_in[67]^data_in[71]^data_in[
75]^data_in[82]^data_in[83]^data_in[84]^data_in[85]^data_in[90]^data_in[9
1]^data_in[92]^data_in[93]^data_in[94]^data_in[95]^data_in[96]^data_in[98
]^data_in[100]^data_in[101]^data_in[102]^data_in[103]^data_in[104]^data_i
n[106]^data_in[108]^data_in[110]^data_in[113]^data_in[114]^data_in[116]^d
ata_in[118]^data_in[121]^data_in[122]^data_in[125]^data_in[126]^data_in[1
27]^data_in[129]^data_in[130]^data_in[133]^data_in[134]^data_in[135]^data
_in[137]^data_in[138]^data_in[139]^data_in[141]^data_in[144]^data_in[148]
^data_in[150]^data_in[151]^data_in[153]^data_in[156]^data_in[157]^data_in
[158]^data_in[163]^data_in[164]^data_in[166]^data_in[167]^data_in[171]^da
ta_in[173]^data_in[174]^data_in[175]^data_in[176]^data_in[177]^data_in[17
```

```
9]^data_in[180]^data_in[182]^data_in[183]^data_in[184]^data_in[185]^data_
in[188]^data_in[190]^data_in[192]^data_in[193]^data_in[195]^data_in[196]^
data_in[198]^data_in[200]^data_in[201]^data_in[202]^data_in[205]^data_in[
206]^data_in[208]^data_in[209]^data_in[211]^data_in[212]^data_in[215]^dat
a_in[217]^data_in[218]^data_in[219]^data_in[221]^data_in[222]^data_in[224
]^data_in[225]^data_in[227]^data_in[228]^data_in[229]^data_in[232]^data_i
n[234]^data_in[235]^data_in[237]^data_in[238]^data_in[240]^data_in[241]^d
ata_in[243]^data_in[247]^data_in[248]^data_in[250]^data_in[251]^data_in[2
53]^data_in[254]^data_in[256]^data_in[257]^data_in[262]^data_in[265]^data
_in[268]^data_in[271]^data_in[277]^data_in[278]^data_in[279]^data_in[283]
^data_in[284]^data_in[285]^data_in[292]^data_in[294]^data_in[295]^data_in
[296]^data_in[297]^data_in[299]^data_in[307]^data_in[308]^data_in[310]^da
ta_in[312]^data_in[313]^data_in[322]^data_in[325]^data_in[326]^data_in[32
8]^data_in[329]^data_in[330]^data_in[331]^data_in[332]^data_in[333]^data_
in[334]^data_in[335]^data_in[336]^data_in[340]^data_in[343]^data_in[345]^
data_in[347]^data_in[349]^data_in[351]^data_in[352]^data_in[353]^data_in[
354]^data_in[358]^data_in[359]^data_in[362]^data_in[363]^data_in[366]^dat
a_in[368]^data_in[373]^data_in[377]^data_in[381]^data_in[382]^data_in[388
]^data_in[389]^data_in[390]^data_in[391]^data_in[396]^data_in[403]^data_i
n[405]^data_in[411]^data_in[412]^data_in[413]^data_in[414]^data_in[415]^d
ata_in[416]^data_in[417]^data_in[420]^data_in[421]^data_in[422]^data_in[4
23]^data_in[424]^data_in[425]^data_in[427]^data_in[429]^data_in[431]^data
_in[435]^data_in[437]^data_in[439]^data_in[442]^data_in[443]^data_in[446]
^data_in[447]^data_in[448]^data_in[449]^data_in[452]^data_in[453]^data_in
[457]^data_in[461]^data_in[463]^data_in[467]^data_in[472]^data_in[473]^da
ta_in[474]^data_in[475]^data_in[478]^data_in[479]^data_in[480]^data_in[48
1]^data_in[487]^data_in[489]^data_in[493]^data_in[495]^data_in[502]^data_
in[503]^data_in[508]^data_in[509]^data_in[517]^data_in[523]^data_in[532]^
data_in[533]^data_in[534]^data_in[535]^data_in[536]^data_in[537]^data_in[
547]^data_in[549]^data_in[551]^data_in[562]^data_in[563]^data_in[566]^dat
a_in[567]^data_in[568]^data_in[569]^data_in[570]^data_in[571]^data_in[572
]^data_in[573]^data_in[574]^data_in[575]^data_in[576]^data_in[578]^data_i
n[579]^data_in[580]^data_in[582]^data_in[584]^data_in[586]^data_in[588]^d
ata_in[590]^data_in[593]^data_in[595]^data_in[596]^data_in[599]^data_in[6
00]^data_in[603]^data_in[604]^data_in[608]^data_in[609]^data_in[611]^data
_in[612]^data_in[613]^data_in[615]^data_in[616]^data_in[617]^data_in[619]
^data_in[620]^data_in[621]^data_in[622]^data_in[624]^data_in[625]^data_in
[627]^data_in[630]^data_in[632]^data_in[633]^data_in[635]^data_in[637]^da
ta_in[638]^data_in[640]^data_in[641]^data_in[645]^data_in[646]^data_in[64
8]^data_in[649]^data_in[652]^data_in[655]^data_in[660]^data_in[663]^data_
in[667]^data_in[668]^data_in[669]^data_in[675]^data_in[676]^data_in[677]^
data_in[682]^data_in[684]^data_in[685]^data_in[686]^data_in[687]^data_in[
688]^data_in[689]^data_in[691]^data_in[697]^data_in[698]^data_in[700]^dat
a_in[702]^data_in[704]^data_in[705]^data_in[712]^data_in[715]^data_in[716
]^data_in[719]^data_in[727]^data_in[728]^data_in[729]^data_in[731]^data_i
n[732]^data_in[733]^data_in[742]^data_in[744]^data_in[745]^data_in[747]^d
ata_in[757]^data_in[758]^data_in[760]^data_in[761]^data_in[772]^data_in[7
75]^data_in[787]^data_in[788]^data_in[789]^data_in[802]^data_in[804]^data
_in[805]^data_in[806]^data_in[807]^data_in[808]^data_in[809]^data_in[810]
^data_in[811]^data_in[812]^data_in[813]^data_in[814]^data_in[815]^data_in
[816]^data_in[819]^data_in[821]^data_in[823]^data_in[825]^data_in[827]^da
ta_in[829]^data_in[831]^data_in[832]^data_in[833]^data_in[836]^data_in[83
7]^data_in[840]^data_in[841]^data_in[844]^data_in[845]^data_in[847]^data_
in[851]^data_in[855]^data_in[859]^data_in[862]^data_in[863]^data_in[864]^
data_in[865]^data_in[870]^data_in[871]^data_in[872]^data_in[873]^data_in[
877]^data_in[879]^data_in[885]^data_in[887]^data_in[892]^data_in[893]^dat
a_in[900]^data_in[901]^data_in[907]^data_in[915]^data_in[922]^data_in[923
]^data_in[924]^data_in[925]^data_in[926]^data_in[927]^data_in[928]^data_i
n[929]^data_in[937]^data_in[939]^data_in[941]^data_in[943]^data_in[952]^d
ata_in[953]^data_in[956]^data_in[957]^data_in[967]^data_in[971]^data_in[9
```

```
82]^data_in[983]^data_in[984]^data_in[985]^data_in[997]^data_in[999]^data
_in[1012]^data_in[1013];



    crc_out[11] =
data_in[5]^data_in[6]^data_in[8]^data_in[20]^data_in[23]^data_in[24]^data
_in[25]^data_in[26]^data_in[27]^data_in[28]^data_in[29]^data_in[30]^data_
in[31]^data_in[32]^data_in[33]^data_in[34]^data_in[38]^data_in[40]^data_i
n[42]^data_in[44]^data_in[46]^data_in[48]^data_in[53]^data_in[54]^data_in
[57]^data_in[58]^data_in[61]^data_in[62]^data_in[68]^data_in[72]^data_in[
76]^data_in[83]^data_in[84]^data_in[85]^data_in[86]^data_in[91]^data_in[9
2]^data_in[93]^data_in[94]^data_in[95]^data_in[96]^data_in[97]^data_in[99
]^data_in[101]^data_in[102]^data_in[103]^data_in[104]^data_in[105]^data_i
n[107]^data_in[109]^data_in[111]^data_in[114]^data_in[115]^data_in[117]^d
ata_in[119]^data_in[122]^data_in[123]^data_in[126]^data_in[127]^data_in[1
28]^data_in[130]^data_in[131]^data_in[134]^data_in[135]^data_in[136]^data
_in[138]^data_in[139]^data_in[140]^data_in[142]^data_in[145]^data_in[149]
^data_in[151]^data_in[152]^data_in[154]^data_in[157]^data_in[158]^data_in
[159]^data_in[164]^data_in[165]^data_in[167]^data_in[168]^data_in[172]^da
ta_in[174]^data_in[175]^data_in[176]^data_in[177]^data_in[178]^data_in[18
0]^data_in[181]^data_in[183]^data_in[184]^data_in[185]^data_in[186]^data_
in[189]^data_in[191]^data_in[193]^data_in[194]^data_in[196]^data_in[197]^
data_in[199]^data_in[201]^data_in[202]^data_in[203]^data_in[206]^data_in[
207]^data_in[209]^data_in[210]^data_in[212]^data_in[213]^data_in[216]^dat
a_in[218]^data_in[219]^data_in[220]^data_in[222]^data_in[223]^data_in[225
]^data_in[226]^data_in[228]^data_in[229]^data_in[230]^data_in[233]^data_i
n[235]^data_in[236]^data_in[238]^data_in[239]^data_in[241]^data_in[242]^d
ata_in[244]^data_in[248]^data_in[249]^data_in[251]^data_in[252]^data_in[2
54]^data_in[255]^data_in[257]^data_in[258]^data_in[263]^data_in[266]^data
_in[269]^data_in[272]^data_in[278]^data_in[279]^data_in[280]^data_in[284]
^data_in[285]^data_in[286]^data_in[293]^data_in[295]^data_in[296]^data_in
[297]^data_in[298]^data_in[300]^data_in[308]^data_in[309]^data_in[311]^da
ta_in[313]^data_in[314]^data_in[323]^data_in[326]^data_in[327]^data_in[32
9]^data_in[330]^data_in[331]^data_in[332]^data_in[333]^data_in[334]^data_
in[335]^data_in[336]^data_in[337]^data_in[341]^data_in[344]^data_in[346]^
data_in[348]^data_in[350]^data_in[352]^data_in[353]^data_in[354]^data_in[
355]^data_in[359]^data_in[360]^data_in[363]^data_in[364]^data_in[367]^dat
a_in[369]^data_in[374]^data_in[378]^data_in[382]^data_in[383]^data_in[389
]^data_in[390]^data_in[391]^data_in[392]^data_in[397]^data_in[404]^data_i
n[406]^data_in[412]^data_in[413]^data_in[414]^data_in[415]^data_in[416]^d
ata_in[417]^data_in[418]^data_in[421]^data_in[422]^data_in[423]^data_in[4
24]^data_in[425]^data_in[426]^data_in[428]^data_in[430]^data_in[432]^data
_in[436]^data_in[438]^data_in[440]^data_in[443]^data_in[444]^data_in[447]
^data_in[448]^data_in[449]^data_in[450]^data_in[453]^data_in[454]^data_in
[458]^data_in[462]^data_in[464]^data_in[468]^data_in[473]^data_in[474]^da
ta_in[475]^data_in[476]^data_in[479]^data_in[480]^data_in[481]^data_in[48
2]^data_in[488]^data_in[490]^data_in[494]^data_in[496]^data_in[503]^data_
in[504]^data_in[509]^data_in[510]^data_in[518]^data_in[524]^data_in[533]^
data_in[534]^data_in[535]^data_in[536]^data_in[537]^data_in[538]^data_in[
548]^data_in[550]^data_in[552]^data_in[563]^data_in[564]^data_in[567]^dat
a_in[568]^data_in[569]^data_in[570]^data_in[571]^data_in[572]^data_in[573
]^data_in[574]^data_in[575]^data_in[576]^data_in[577]^data_in[579]^data_i
n[580]^data_in[581]^data_in[583]^data_in[585]^data_in[587]^data_in[589]^d
ata_in[591]^data_in[594]^data_in[596]^data_in[597]^data_in[600]^data_in[6
01]^data_in[604]^data_in[605]^data_in[609]^data_in[610]^data_in[612]^data
_in[613]^data_in[614]^data_in[616]^data_in[617]^data_in[618]^data_in[620]
^data_in[621]^data_in[622]^data_in[623]^data_in[625]^data_in[626]^data_in
[628]^data_in[631]^data_in[633]^data_in[634]^data_in[636]^data_in[638]^da
ta_in[639]^data_in[641]^data_in[642]^data_in[646]^data_in[647]^data_in[64
```

```
9]^data_in[650]^data_in[653]^data_in[656]^data_in[661]^data_in[664]^data_
in[668]^data_in[669]^data_in[670]^data_in[676]^data_in[677]^data_in[678]^
data_in[683]^data_in[685]^data_in[686]^data_in[687]^data_in[688]^data_in[
689]^data_in[690]^data_in[692]^data_in[698]^data_in[699]^data_in[701]^dat
a_in[703]^data_in[705]^data_in[706]^data_in[713]^data_in[716]^data_in[717
]^data_in[720]^data_in[728]^data_in[729]^data_in[730]^data_in[732]^data_i
n[733]^data_in[734]^data_in[743]^data_in[745]^data_in[746]^data_in[748]^d
ata_in[758]^data_in[759]^data_in[761]^data_in[762]^data_in[773]^data_in[7
76]^data_in[788]^data_in[789]^data_in[790]^data_in[803]^data_in[805]^data
_in[806]^data_in[807]^data_in[808]^data_in[809]^data_in[810]^data_in[811]
^data_in[812]^data_in[813]^data_in[814]^data_in[815]^data_in[816]^data_in
[817]^data_in[820]^data_in[822]^data_in[824]^data_in[826]^data_in[828]^da
ta_in[830]^data_in[832]^data_in[833]^data_in[834]^data_in[837]^data_in[83
8]^data_in[841]^data_in[842]^data_in[845]^data_in[846]^data_in[848]^data_
in[852]^data_in[856]^data_in[860]^data_in[863]^data_in[864]^data_in[865]^
data_in[866]^data_in[871]^data_in[872]^data_in[873]^data_in[874]^data_in[
878]^data_in[880]^data_in[886]^data_in[888]^data_in[893]^data_in[894]^dat
a_in[901]^data_in[902]^data_in[908]^data_in[916]^data_in[923]^data_in[924
]^data_in[925]^data_in[926]^data_in[927]^data_in[928]^data_in[929]^data_i
n[930]^data_in[938]^data_in[940]^data_in[942]^data_in[944]^data_in[953]^d
ata_in[954]^data_in[957]^data_in[958]^data_in[968]^data_in[972]^data_in[9
83]^data_in[984]^data_in[985]^data_in[986]^data_in[998]^data_in[1000]^dat
a_in[1013]^data_in[1014];


    crc_out[10] =
data_in[6]^data_in[7]^data_in[9]^data_in[21]^data_in[24]^data_in[25]^data
_in[26]^data_in[27]^data_in[28]^data_in[29]^data_in[30]^data_in[31]^data_
in[32]^data_in[33]^data_in[34]^data_in[35]^data_in[39]^data_in[41]^data_i
n[43]^data_in[45]^data_in[47]^data_in[49]^data_in[54]^data_in[55]^data_in
[58]^data_in[59]^data_in[62]^data_in[63]^data_in[69]^data_in[73]^data_in[
77]^data_in[84]^data_in[85]^data_in[86]^data_in[87]^data_in[92]^data_in[9
3]^data_in[94]^data_in[95]^data_in[96]^data_in[97]^data_in[98]^data_in[10
0]^data_in[102]^data_in[103]^data_in[104]^data_in[105]^data_in[106]^data_
in[108]^data_in[110]^data_in[112]^data_in[115]^data_in[116]^data_in[118]^
data_in[120]^data_in[123]^data_in[124]^data_in[127]^data_in[128]^data_in[
129]^data_in[131]^data_in[132]^data_in[135]^data_in[136]^data_in[137]^dat
a_in[139]^data_in[140]^data_in[141]^data_in[143]^data_in[146]^data_in[150
]^data_in[152]^data_in[153]^data_in[155]^data_in[158]^data_in[159]^data_i
n[160]^data_in[165]^data_in[166]^data_in[168]^data_in[169]^data_in[173]^d
ata_in[175]^data_in[176]^data_in[177]^data_in[178]^data_in[179]^data_in[1
81]^data_in[182]^data_in[184]^data_in[185]^data_in[186]^data_in[187]^data
_in[190]^data_in[192]^data_in[194]^data_in[195]^data_in[197]^data_in[198]
^data_in[200]^data_in[202]^data_in[203]^data_in[204]^data_in[207]^data_in
[208]^data_in[210]^data_in[211]^data_in[213]^data_in[214]^data_in[217]^da
ta_in[219]^data_in[220]^data_in[221]^data_in[223]^data_in[224]^data_in[22
6]^data_in[227]^data_in[229]^data_in[230]^data_in[231]^data_in[234]^data_
in[236]^data_in[237]^data_in[239]^data_in[240]^data_in[242]^data_in[243]^
data_in[245]^data_in[249]^data_in[250]^data_in[252]^data_in[253]^data_in[
255]^data_in[256]^data_in[258]^data_in[259]^data_in[264]^data_in[267]^dat
a_in[270]^data_in[273]^data_in[279]^data_in[280]^data_in[281]^data_in[285
]^data_in[286]^data_in[287]^data_in[294]^data_in[296]^data_in[297]^data_i
n[298]^data_in[299]^data_in[301]^data_in[309]^data_in[310]^data_in[312]^d
ata_in[314]^data_in[315]^data_in[324]^data_in[327]^data_in[328]^data_in[3
30]^data_in[331]^data_in[332]^data_in[333]^data_in[334]^data_in[335]^data
_in[336]^data_in[337]^data_in[338]^data_in[342]^data_in[345]^data_in[347]
^data_in[349]^data_in[351]^data_in[353]^data_in[354]^data_in[355]^data_in
[356]^data_in[360]^data_in[361]^data_in[364]^data_in[365]^data_in[368]^da
ta_in[370]^data_in[375]^data_in[379]^data_in[383]^data_in[384]^data_in[39
```

```
0]^data_in[391]^data_in[392]^data_in[393]^data_in[398]^data_in[405]^data_
in[407]^data_in[413]^data_in[414]^data_in[415]^data_in[416]^data_in[417]^
data_in[418]^data_in[419]^data_in[422]^data_in[423]^data_in[424]^data_in[
425]^data_in[426]^data_in[427]^data_in[429]^data_in[431]^data_in[433]^dat
a_in[437]^data_in[439]^data_in[441]^data_in[444]^data_in[445]^data_in[448
]^data_in[449]^data_in[450]^data_in[451]^data_in[454]^data_in[455]^data_i
n[459]^data_in[463]^data_in[465]^data_in[469]^data_in[474]^data_in[475]^d
ata_in[476]^data_in[477]^data_in[480]^data_in[481]^data_in[482]^data_in[4
83]^data_in[489]^data_in[491]^data_in[495]^data_in[497]^data_in[504]^data
_in[505]^data_in[510]^data_in[511]^data_in[519]^data_in[525]^data_in[534]
^data_in[535]^data_in[536]^data_in[537]^data_in[538]^data_in[539]^data_in
[549]^data_in[551]^data_in[553]^data_in[564]^data_in[565]^data_in[568]^da
ta_in[569]^data_in[570]^data_in[571]^data_in[572]^data_in[573]^data_in[57
4]^data_in[575]^data_in[576]^data_in[577]^data_in[578]^data_in[580]^data_
in[581]^data_in[582]^data_in[584]^data_in[586]^data_in[588]^data_in[590]^
data_in[592]^data_in[595]^data_in[597]^data_in[598]^data_in[601]^data_in[
602]^data_in[605]^data_in[606]^data_in[610]^data_in[611]^data_in[613]^dat
a_in[614]^data_in[615]^data_in[617]^data_in[618]^data_in[619]^data_in[621
]^data_in[622]^data_in[623]^data_in[624]^data_in[626]^data_in[627]^data_i
n[629]^data_in[632]^data_in[634]^data_in[635]^data_in[637]^data_in[639]^d
ata_in[640]^data_in[642]^data_in[643]^data_in[647]^data_in[648]^data_in[6
50]^data_in[651]^data_in[654]^data_in[657]^data_in[662]^data_in[665]^data
_in[669]^data_in[670]^data_in[671]^data_in[677]^data_in[678]^data_in[679]
^data_in[684]^data_in[686]^data_in[687]^data_in[688]^data_in[689]^data_in
[690]^data_in[691]^data_in[693]^data_in[699]^data_in[700]^data_in[702]^da
ta_in[704]^data_in[706]^data_in[707]^data_in[714]^data_in[717]^data_in[71
8]^data_in[721]^data_in[729]^data_in[730]^data_in[731]^data_in[733]^data_
in[734]^data_in[735]^data_in[744]^data_in[746]^data_in[747]^data_in[749]^
data_in[759]^data_in[760]^data_in[762]^data_in[763]^data_in[774]^data_in[
777]^data_in[789]^data_in[790]^data_in[791]^data_in[804]^data_in[806]^dat
a_in[807]^data_in[808]^data_in[809]^data_in[810]^data_in[811]^data_in[812
]^data_in[813]^data_in[814]^data_in[815]^data_in[816]^data_in[817]^data_i
n[818]^data_in[821]^data_in[823]^data_in[825]^data_in[827]^data_in[829]^d
ata_in[831]^data_in[833]^data_in[834]^data_in[835]^data_in[838]^data_in[8
39]^data_in[842]^data_in[843]^data_in[846]^data_in[847]^data_in[849]^data
_in[853]^data_in[857]^data_in[861]^data_in[864]^data_in[865]^data_in[866]
^data_in[867]^data_in[872]^data_in[873]^data_in[874]^data_in[875]^data_in
[879]^data_in[881]^data_in[887]^data_in[889]^data_in[894]^data_in[895]^da
ta_in[902]^data_in[903]^data_in[909]^data_in[917]^data_in[924]^data_in[92
5]^data_in[926]^data_in[927]^data_in[928]^data_in[929]^data_in[930]^data_
in[931]^data_in[939]^data_in[941]^data_in[943]^data_in[945]^data_in[954]^
data_in[955]^data_in[958]^data_in[959]^data_in[969]^data_in[973]^data_in[
984]^data_in[985]^data_in[986]^data_in[987]^data_in[999]^data_in[1001]^da
ta_in[1014]^data_in[1015];



    crc_out[9] =
data_in[7]^data_in[8]^data_in[10]^data_in[22]^data_in[25]^data_in[26]^dat
a_in[27]^data_in[28]^data_in[29]^data_in[30]^data_in[31]^data_in[32]^data
_in[33]^data_in[34]^data_in[35]^data_in[36]^data_in[40]^data_in[42]^data_
in[44]^data_in[46]^data_in[48]^data_in[50]^data_in[55]^data_in[56]^data_i
n[59]^data_in[60]^data_in[63]^data_in[64]^data_in[70]^data_in[74]^data_in
[78]^data_in[85]^data_in[86]^data_in[87]^data_in[88]^data_in[93]^data_in[
94]^data_in[95]^data_in[96]^data_in[97]^data_in[98]^data_in[99]^data_in[1
01]^data_in[103]^data_in[104]^data_in[105]^data_in[106]^data_in[107]^data
_in[109]^data_in[111]^data_in[113]^data_in[116]^data_in[117]^data_in[119]
^data_in[121]^data_in[124]^data_in[125]^data_in[128]^data_in[129]^data_in
[130]^data_in[132]^data_in[133]^data_in[136]^data_in[137]^data_in[138]^da
ta_in[140]^data_in[141]^data_in[142]^data_in[144]^data_in[147]^data_in[15
```

```
1]^data_in[153]^data_in[154]^data_in[156]^data_in[159]^data_in[160]^data_
in[161]^data_in[166]^data_in[167]^data_in[169]^data_in[170]^data_in[174]^
data_in[176]^data_in[177]^data_in[178]^data_in[179]^data_in[180]^data_in[
182]^data_in[183]^data_in[185]^data_in[186]^data_in[187]^data_in[188]^dat
a_in[191]^data_in[193]^data_in[195]^data_in[196]^data_in[198]^data_in[199
]^data_in[201]^data_in[203]^data_in[204]^data_in[205]^data_in[208]^data_i
n[209]^data_in[211]^data_in[212]^data_in[214]^data_in[215]^data_in[218]^d
ata_in[220]^data_in[221]^data_in[222]^data_in[224]^data_in[225]^data_in[2
27]^data_in[228]^data_in[230]^data_in[231]^data_in[232]^data_in[235]^data
_in[237]^data_in[238]^data_in[240]^data_in[241]^data_in[243]^data_in[244]
^data_in[246]^data_in[250]^data_in[251]^data_in[253]^data_in[254]^data_in
[256]^data_in[257]^data_in[259]^data_in[260]^data_in[265]^data_in[268]^da
ta_in[271]^data_in[274]^data_in[280]^data_in[281]^data_in[282]^data_in[28
6]^data_in[287]^data_in[288]^data_in[295]^data_in[297]^data_in[298]^data_
in[299]^data_in[300]^data_in[302]^data_in[310]^data_in[311]^data_in[313]^
data_in[315]^data_in[316]^data_in[325]^data_in[328]^data_in[329]^data_in[
331]^data_in[332]^data_in[333]^data_in[334]^data_in[335]^data_in[336]^dat
a_in[337]^data_in[338]^data_in[339]^data_in[343]^data_in[346]^data_in[348
]^data_in[350]^data_in[352]^data_in[354]^data_in[355]^data_in[356]^data_i
n[357]^data_in[361]^data_in[362]^data_in[365]^data_in[366]^data_in[369]^d
ata_in[371]^data_in[376]^data_in[380]^data_in[384]^data_in[385]^data_in[3
91]^data_in[392]^data_in[393]^data_in[394]^data_in[399]^data_in[406]^data
_in[408]^data_in[414]^data_in[415]^data_in[416]^data_in[417]^data_in[418]
^data_in[419]^data_in[420]^data_in[423]^data_in[424]^data_in[425]^data_in
[426]^data_in[427]^data_in[428]^data_in[430]^data_in[432]^data_in[434]^da
ta_in[438]^data_in[440]^data_in[442]^data_in[445]^data_in[446]^data_in[44
9]^data_in[450]^data_in[451]^data_in[452]^data_in[455]^data_in[456]^data_
in[460]^data_in[464]^data_in[466]^data_in[470]^data_in[475]^data_in[476]^
data_in[477]^data_in[478]^data_in[481]^data_in[482]^data_in[483]^data_in[
484]^data_in[490]^data_in[492]^data_in[496]^data_in[498]^data_in[505]^dat
a_in[506]^data_in[511]^data_in[512]^data_in[520]^data_in[526]^data_in[535
]^data_in[536]^data_in[537]^data_in[538]^data_in[539]^data_in[540]^data_i
n[550]^data_in[552]^data_in[554]^data_in[565]^data_in[566]^data_in[569]^d
ata_in[570]^data_in[571]^data_in[572]^data_in[573]^data_in[574]^data_in[5
75]^data_in[576]^data_in[577]^data_in[578]^data_in[579]^data_in[581]^data
_in[582]^data_in[583]^data_in[585]^data_in[587]^data_in[589]^data_in[591]
^data_in[593]^data_in[596]^data_in[598]^data_in[599]^data_in[602]^data_in
[603]^data_in[606]^data_in[607]^data_in[611]^data_in[612]^data_in[614]^da
ta_in[615]^data_in[616]^data_in[618]^data_in[619]^data_in[620]^data_in[62
2]^data_in[623]^data_in[624]^data_in[625]^data_in[627]^data_in[628]^data_
in[630]^data_in[633]^data_in[635]^data_in[636]^data_in[638]^data_in[640]^
data_in[641]^data_in[643]^data_in[644]^data_in[648]^data_in[649]^data_in[
651]^data_in[652]^data_in[655]^data_in[658]^data_in[663]^data_in[666]^dat
a_in[670]^data_in[671]^data_in[672]^data_in[678]^data_in[679]^data_in[680
]^data_in[685]^data_in[687]^data_in[688]^data_in[689]^data_in[690]^data_i
n[691]^data_in[692]^data_in[694]^data_in[700]^data_in[701]^data_in[703]^d
ata_in[705]^data_in[707]^data_in[708]^data_in[715]^data_in[718]^data_in[7
19]^data_in[722]^data_in[730]^data_in[731]^data_in[732]^data_in[734]^data
_in[735]^data_in[736]^data_in[745]^data_in[747]^data_in[748]^data_in[750]
^data_in[760]^data_in[761]^data_in[763]^data_in[764]^data_in[775]^data_in
[778]^data_in[790]^data_in[791]^data_in[792]^data_in[805]^data_in[807]^da
ta_in[808]^data_in[809]^data_in[810]^data_in[811]^data_in[812]^data_in[81
3]^data_in[814]^data_in[815]^data_in[816]^data_in[817]^data_in[818]^data_
in[819]^data_in[822]^data_in[824]^data_in[826]^data_in[828]^data_in[830]^
data_in[832]^data_in[834]^data_in[835]^data_in[836]^data_in[839]^data_in[
840]^data_in[843]^data_in[844]^data_in[847]^data_in[848]^data_in[850]^dat
a_in[854]^data_in[858]^data_in[862]^data_in[865]^data_in[866]^data_in[867
]^data_in[868]^data_in[873]^data_in[874]^data_in[875]^data_in[876]^data_i
n[880]^data_in[882]^data_in[888]^data_in[890]^data_in[895]^data_in[896]^d
ata_in[903]^data_in[904]^data_in[910]^data_in[918]^data_in[925]^data_in[9
```

```
26]^data_in[927]^data_in[928]^data_in[929]^data_in[930]^data_in[931]^data
_in[932]^data_in[940]^data_in[942]^data_in[944]^data_in[946]^data_in[955]
^data_in[956]^data_in[959]^data_in[960]^data_in[970]^data_in[974]^data_in
[985]^data_in[986]^data_in[987]^data_in[988]^data_in[1000]^data_in[1002]^
data_in[1015]^data_in[1016];



    crc_out[8] =
data_in[8]^data_in[9]^data_in[11]^data_in[23]^data_in[26]^data_in[27]^dat
a_in[28]^data_in[29]^data_in[30]^data_in[31]^data_in[32]^data_in[33]^data
_in[34]^data_in[35]^data_in[36]^data_in[37]^data_in[41]^data_in[43]^data_
in[45]^data_in[47]^data_in[49]^data_in[51]^data_in[56]^data_in[57]^data_i
n[60]^data_in[61]^data_in[64]^data_in[65]^data_in[71]^data_in[75]^data_in
[79]^data_in[86]^data_in[87]^data_in[88]^data_in[89]^data_in[94]^data_in[
95]^data_in[96]^data_in[97]^data_in[98]^data_in[99]^data_in[100]^data_in[
102]^data_in[104]^data_in[105]^data_in[106]^data_in[107]^data_in[108]^dat
a_in[110]^data_in[112]^data_in[114]^data_in[117]^data_in[118]^data_in[120
]^data_in[122]^data_in[125]^data_in[126]^data_in[129]^data_in[130]^data_i
n[131]^data_in[133]^data_in[134]^data_in[137]^data_in[138]^data_in[139]^d
ata_in[141]^data_in[142]^data_in[143]^data_in[145]^data_in[148]^data_in[1
52]^data_in[154]^data_in[155]^data_in[157]^data_in[160]^data_in[161]^data
_in[162]^data_in[167]^data_in[168]^data_in[170]^data_in[171]^data_in[175]
^data_in[177]^data_in[178]^data_in[179]^data_in[180]^data_in[181]^data_in
[183]^data_in[184]^data_in[186]^data_in[187]^data_in[188]^data_in[189]^da
ta_in[192]^data_in[194]^data_in[196]^data_in[197]^data_in[199]^data_in[20
0]^data_in[202]^data_in[204]^data_in[205]^data_in[206]^data_in[209]^data_
in[210]^data_in[212]^data_in[213]^data_in[215]^data_in[216]^data_in[219]^
data_in[221]^data_in[222]^data_in[223]^data_in[225]^data_in[226]^data_in[
228]^data_in[229]^data_in[231]^data_in[232]^data_in[233]^data_in[236]^dat
a_in[238]^data_in[239]^data_in[241]^data_in[242]^data_in[244]^data_in[245
]^data_in[247]^data_in[251]^data_in[252]^data_in[254]^data_in[255]^data_i
n[257]^data_in[258]^data_in[260]^data_in[261]^data_in[266]^data_in[269]^d
ata_in[272]^data_in[275]^data_in[281]^data_in[282]^data_in[283]^data_in[2
87]^data_in[288]^data_in[289]^data_in[296]^data_in[298]^data_in[299]^data
_in[300]^data_in[301]^data_in[303]^data_in[311]^data_in[312]^data_in[314]
^data_in[316]^data_in[317]^data_in[326]^data_in[329]^data_in[330]^data_in
[332]^data_in[333]^data_in[334]^data_in[335]^data_in[336]^data_in[337]^da
ta_in[338]^data_in[339]^data_in[340]^data_in[344]^data_in[347]^data_in[34
9]^data_in[351]^data_in[353]^data_in[355]^data_in[356]^data_in[357]^data_
in[358]^data_in[362]^data_in[363]^data_in[366]^data_in[367]^data_in[370]^
data_in[372]^data_in[377]^data_in[381]^data_in[385]^data_in[386]^data_in[
392]^data_in[393]^data_in[394]^data_in[395]^data_in[400]^data_in[407]^dat
a_in[409]^data_in[415]^data_in[416]^data_in[417]^data_in[418]^data_in[419
]^data_in[420]^data_in[421]^data_in[424]^data_in[425]^data_in[426]^data_i
n[427]^data_in[428]^data_in[429]^data_in[431]^data_in[433]^data_in[435]^d
ata_in[439]^data_in[441]^data_in[443]^data_in[446]^data_in[447]^data_in[4
50]^data_in[451]^data_in[452]^data_in[453]^data_in[456]^data_in[457]^data
_in[461]^data_in[465]^data_in[467]^data_in[471]^data_in[476]^data_in[477]
^data_in[478]^data_in[479]^data_in[482]^data_in[483]^data_in[484]^data_in
[485]^data_in[491]^data_in[493]^data_in[497]^data_in[499]^data_in[506]^da
ta_in[507]^data_in[512]^data_in[513]^data_in[521]^data_in[527]^data_in[53
6]^data_in[537]^data_in[538]^data_in[539]^data_in[540]^data_in[541]^data_
in[551]^data_in[553]^data_in[555]^data_in[566]^data_in[567]^data_in[570]^
data_in[571]^data_in[572]^data_in[573]^data_in[574]^data_in[575]^data_in[
576]^data_in[577]^data_in[578]^data_in[579]^data_in[580]^data_in[582]^dat
a_in[583]^data_in[584]^data_in[586]^data_in[588]^data_in[590]^data_in[592
]^data_in[594]^data_in[597]^data_in[599]^data_in[600]^data_in[603]^data_i
n[604]^data_in[607]^data_in[608]^data_in[612]^data_in[613]^data_in[615]^d
ata_in[616]^data_in[617]^data_in[619]^data_in[620]^data_in[621]^data_in[6
```

```
23]^data_in[624]^data_in[625]^data_in[626]^data_in[628]^data_in[629]^data
_in[631]^data_in[634]^data_in[636]^data_in[637]^data_in[639]^data_in[641]
^data_in[642]^data_in[644]^data_in[645]^data_in[649]^data_in[650]^data_in
[652]^data_in[653]^data_in[656]^data_in[659]^data_in[664]^data_in[667]^da
ta_in[671]^data_in[672]^data_in[673]^data_in[679]^data_in[680]^data_in[68
1]^data_in[686]^data_in[688]^data_in[689]^data_in[690]^data_in[691]^data_
in[692]^data_in[693]^data_in[695]^data_in[701]^data_in[702]^data_in[704]^
data_in[706]^data_in[708]^data_in[709]^data_in[716]^data_in[719]^data_in[
720]^data_in[723]^data_in[731]^data_in[732]^data_in[733]^data_in[735]^dat
a_in[736]^data_in[737]^data_in[746]^data_in[748]^data_in[749]^data_in[751
]^data_in[761]^data_in[762]^data_in[764]^data_in[765]^data_in[776]^data_i
n[779]^data_in[791]^data_in[792]^data_in[793]^data_in[806]^data_in[808]^d
ata_in[809]^data_in[810]^data_in[811]^data_in[812]^data_in[813]^data_in[8
14]^data_in[815]^data_in[816]^data_in[817]^data_in[818]^data_in[819]^data
_in[820]^data_in[823]^data_in[825]^data_in[827]^data_in[829]^data_in[831]
^data_in[833]^data_in[835]^data_in[836]^data_in[837]^data_in[840]^data_in
[841]^data_in[844]^data_in[845]^data_in[848]^data_in[849]^data_in[851]^da
ta_in[855]^data_in[859]^data_in[863]^data_in[866]^data_in[867]^data_in[86
8]^data_in[869]^data_in[874]^data_in[875]^data_in[876]^data_in[877]^data_
in[881]^data_in[883]^data_in[889]^data_in[891]^data_in[896]^data_in[897]^
data_in[904]^data_in[905]^data_in[911]^data_in[919]^data_in[926]^data_in[
927]^data_in[928]^data_in[929]^data_in[930]^data_in[931]^data_in[932]^dat
a_in[933]^data_in[941]^data_in[943]^data_in[945]^data_in[947]^data_in[956
]^data_in[957]^data_in[960]^data_in[961]^data_in[971]^data_in[975]^data_i
n[986]^data_in[987]^data_in[988]^data_in[989]^data_in[1001]^data_in[1003]
^data_in[1016]^data_in[1017];


    crc_out[7] =
data_in[9]^data_in[10]^data_in[12]^data_in[24]^data_in[27]^data_in[28]^da
ta_in[29]^data_in[30]^data_in[31]^data_in[32]^data_in[33]^data_in[34]^dat
a_in[35]^data_in[36]^data_in[37]^data_in[38]^data_in[42]^data_in[44]^data
_in[46]^data_in[48]^data_in[50]^data_in[52]^data_in[57]^data_in[58]^data_
in[61]^data_in[62]^data_in[65]^data_in[66]^data_in[72]^data_in[76]^data_i
n[80]^data_in[87]^data_in[88]^data_in[89]^data_in[90]^data_in[95]^data_in
[96]^data_in[97]^data_in[98]^data_in[99]^data_in[100]^data_in[101]^data_i
n[103]^data_in[105]^data_in[106]^data_in[107]^data_in[108]^data_in[109]^d
ata_in[111]^data_in[113]^data_in[115]^data_in[118]^data_in[119]^data_in[1
21]^data_in[123]^data_in[126]^data_in[127]^data_in[130]^data_in[131]^data
_in[132]^data_in[134]^data_in[135]^data_in[138]^data_in[139]^data_in[140]
^data_in[142]^data_in[143]^data_in[144]^data_in[146]^data_in[149]^data_in
[153]^data_in[155]^data_in[156]^data_in[158]^data_in[161]^data_in[162]^da
ta_in[163]^data_in[168]^data_in[169]^data_in[171]^data_in[172]^data_in[17
6]^data_in[178]^data_in[179]^data_in[180]^data_in[181]^data_in[182]^data_
in[184]^data_in[185]^data_in[187]^data_in[188]^data_in[189]^data_in[190]^
data_in[193]^data_in[195]^data_in[197]^data_in[198]^data_in[200]^data_in[
201]^data_in[203]^data_in[205]^data_in[206]^data_in[207]^data_in[210]^dat
a_in[211]^data_in[213]^data_in[214]^data_in[216]^data_in[217]^data_in[220
]^data_in[222]^data_in[223]^data_in[224]^data_in[226]^data_in[227]^data_i
n[229]^data_in[230]^data_in[232]^data_in[233]^data_in[234]^data_in[237]^d
ata_in[239]^data_in[240]^data_in[242]^data_in[243]^data_in[245]^data_in[2
46]^data_in[248]^data_in[252]^data_in[253]^data_in[255]^data_in[256]^data
_in[258]^data_in[259]^data_in[261]^data_in[262]^data_in[267]^data_in[270]
^data_in[273]^data_in[276]^data_in[282]^data_in[283]^data_in[284]^data_in
[288]^data_in[289]^data_in[290]^data_in[297]^data_in[299]^data_in[300]^da
ta_in[301]^data_in[302]^data_in[304]^data_in[312]^data_in[313]^data_in[31
5]^data_in[317]^data_in[318]^data_in[327]^data_in[330]^data_in[331]^data_
in[333]^data_in[334]^data_in[335]^data_in[336]^data_in[337]^data_in[338]^
data_in[339]^data_in[340]^data_in[341]^data_in[345]^data_in[348]^data_in[
350]^data_in[352]^data_in[354]^data_in[356]^data_in[357]^data_in[358]^dat
a_in[359]^data_in[363]^data_in[364]^data_in[367]^data_in[368]^data_in[371
]^data_in[373]^data_in[378]^data_in[382]^data_in[386]^data_in[387]^data_i
n[393]^data_in[394]^data_in[395]^data_in[396]^data_in[401]^data_in[408]^d
ata_in[410]^data_in[416]^data_in[417]^data_in[418]^data_in[419]^data_in[4
20]^data_in[421]^data_in[422]^data_in[425]^data_in[426]^data_in[427]^data
_in[428]^data_in[429]^data_in[430]^data_in[432]^data_in[434]^data_in[436]
^data_in[440]^data_in[442]^data_in[444]^data_in[447]^data_in[448]^data_in
[451]^data_in[452]^data_in[453]^data_in[454]^data_in[457]^data_in[458]^da
ta_in[462]^data_in[466]^data_in[468]^data_in[472]^data_in[477]^data_in[47
8]^data_in[479]^data_in[480]^data_in[483]^data_in[484]^data_in[485]^data_
in[486]^data_in[492]^data_in[494]^data_in[498]^data_in[500]^data_in[507]^
data_in[508]^data_in[513]^data_in[514]^data_in[522]^data_in[528]^data_in[
537]^data_in[538]^data_in[539]^data_in[540]^data_in[541]^data_in[542]^dat
a_in[552]^data_in[554]^data_in[556]^data_in[567]^data_in[568]^data_in[571
]^data_in[572]^data_in[573]^data_in[574]^data_in[575]^data_in[576]^data_i
n[577]^data_in[578]^data_in[579]^data_in[580]^data_in[581]^data_in[583]^d
ata_in[584]^data_in[585]^data_in[587]^data_in[589]^data_in[591]^data_in[5
93]^data_in[595]^data_in[598]^data_in[600]^data_in[601]^data_in[604]^data
_in[605]^data_in[608]^data_in[609]^data_in[613]^data_in[614]^data_in[616]
^data_in[617]^data_in[618]^data_in[620]^data_in[621]^data_in[622]^data_in
[624]^data_in[625]^data_in[626]^data_in[627]^data_in[629]^data_in[630]^da
ta_in[632]^data_in[635]^data_in[637]^data_in[638]^data_in[640]^data_in[64
2]^data_in[643]^data_in[645]^data_in[646]^data_in[650]^data_in[651]^data_
in[653]^data_in[654]^data_in[657]^data_in[660]^data_in[665]^data_in[668]^
data_in[672]^data_in[673]^data_in[674]^data_in[680]^data_in[681]^data_in[
682]^data_in[687]^data_in[689]^data_in[690]^data_in[691]^data_in[692]^dat
a_in[693]^data_in[694]^data_in[696]^data_in[702]^data_in[703]^data_in[705
]^data_in[707]^data_in[709]^data_in[710]^data_in[717]^data_in[720]^data_i
n[721]^data_in[724]^data_in[732]^data_in[733]^data_in[734]^data_in[736]^d
ata_in[737]^data_in[738]^data_in[747]^data_in[749]^data_in[750]^data_in[7
52]^data_in[762]^data_in[763]^data_in[765]^data_in[766]^data_in[777]^data
_in[780]^data_in[792]^data_in[793]^data_in[794]^data_in[807]^data_in[809]
^data_in[810]^data_in[811]^data_in[812]^data_in[813]^data_in[814]^data_in
[815]^data_in[816]^data_in[817]^data_in[818]^data_in[819]^data_in[820]^da
ta_in[821]^data_in[824]^data_in[826]^data_in[828]^data_in[830]^data_in[83
2]^data_in[834]^data_in[836]^data_in[837]^data_in[838]^data_in[841]^data_
in[842]^data_in[845]^data_in[846]^data_in[849]^data_in[850]^data_in[852]^
data_in[856]^data_in[860]^data_in[864]^data_in[867]^data_in[868]^data_in[
869]^data_in[870]^data_in[875]^data_in[876]^data_in[877]^data_in[878]^dat
a_in[882]^data_in[884]^data_in[890]^data_in[892]^data_in[897]^data_in[898
]^data_in[905]^data_in[906]^data_in[912]^data_in[920]^data_in[927]^data_i
n[928]^data_in[929]^data_in[930]^data_in[931]^data_in[932]^data_in[933]^d
ata_in[934]^data_in[942]^data_in[944]^data_in[946]^data_in[948]^data_in[9
57]^data_in[958]^data_in[961]^data_in[962]^data_in[972]^data_in[976]^data
_in[987]^data_in[988]^data_in[989]^data_in[990]^data_in[1002]^data_in[100
4]^data_in[1017]^data_in[1018];


    crc_out[6] =
data_in[10]^data_in[11]^data_in[13]^data_in[25]^data_in[28]^data_in[29]^d
ata_in[30]^data_in[31]^data_in[32]^data_in[33]^data_in[34]^data_in[35]^da
ta_in[36]^data_in[37]^data_in[38]^data_in[39]^data_in[43]^data_in[45]^dat
a_in[47]^data_in[49]^data_in[51]^data_in[53]^data_in[58]^data_in[59]^data
_in[62]^data_in[63]^data_in[66]^data_in[67]^data_in[73]^data_in[77]^data_
in[81]^data_in[88]^data_in[89]^data_in[90]^data_in[91]^data_in[96]^data_i
n[97]^data_in[98]^data_in[99]^data_in[100]^data_in[101]^data_in[102]^data
_in[104]^data_in[106]^data_in[107]^data_in[108]^data_in[109]^data_in[110]
^data_in[112]^data_in[114]^data_in[116]^data_in[119]^data_in[120]^data_in
```

```
[122]^data_in[124]^data_in[127]^data_in[128]^data_in[131]^data_in[132]^da
ta_in[133]^data_in[135]^data_in[136]^data_in[139]^data_in[140]^data_in[14
1]^data_in[143]^data_in[144]^data_in[145]^data_in[147]^data_in[150]^data_
in[154]^data_in[156]^data_in[157]^data_in[159]^data_in[162]^data_in[163]^
data_in[164]^data_in[169]^data_in[170]^data_in[172]^data_in[173]^data_in[
177]^data_in[179]^data_in[180]^data_in[181]^data_in[182]^data_in[183]^dat
a_in[185]^data_in[186]^data_in[188]^data_in[189]^data_in[190]^data_in[191
]^data_in[194]^data_in[196]^data_in[198]^data_in[199]^data_in[201]^data_i
n[202]^data_in[204]^data_in[206]^data_in[207]^data_in[208]^data_in[211]^d
ata_in[212]^data_in[214]^data_in[215]^data_in[217]^data_in[218]^data_in[2
21]^data_in[223]^data_in[224]^data_in[225]^data_in[227]^data_in[228]^data
_in[230]^data_in[231]^data_in[233]^data_in[234]^data_in[235]^data_in[238]
^data_in[240]^data_in[241]^data_in[243]^data_in[244]^data_in[246]^data_in
[247]^data_in[249]^data_in[253]^data_in[254]^data_in[256]^data_in[257]^da
ta_in[259]^data_in[260]^data_in[262]^data_in[263]^data_in[268]^data_in[27
1]^data_in[274]^data_in[277]^data_in[283]^data_in[284]^data_in[285]^data_
in[289]^data_in[290]^data_in[291]^data_in[298]^data_in[300]^data_in[301]^
data_in[302]^data_in[303]^data_in[305]^data_in[313]^data_in[314]^data_in[
316]^data_in[318]^data_in[319]^data_in[328]^data_in[331]^data_in[332]^dat
a_in[334]^data_in[335]^data_in[336]^data_in[337]^data_in[338]^data_in[339
]^data_in[340]^data_in[341]^data_in[342]^data_in[346]^data_in[349]^data_i
n[351]^data_in[353]^data_in[355]^data_in[357]^data_in[358]^data_in[359]^d
ata_in[360]^data_in[364]^data_in[365]^data_in[368]^data_in[369]^data_in[3
72]^data_in[374]^data_in[379]^data_in[383]^data_in[387]^data_in[388]^data
_in[394]^data_in[395]^data_in[396]^data_in[397]^data_in[402]^data_in[409]
^data_in[411]^data_in[417]^data_in[418]^data_in[419]^data_in[420]^data_in
[421]^data_in[422]^data_in[423]^data_in[426]^data_in[427]^data_in[428]^da
ta_in[429]^data_in[430]^data_in[431]^data_in[433]^data_in[435]^data_in[43
7]^data_in[441]^data_in[443]^data_in[445]^data_in[448]^data_in[449]^data_
in[452]^data_in[453]^data_in[454]^data_in[455]^data_in[458]^data_in[459]^
data_in[463]^data_in[467]^data_in[469]^data_in[473]^data_in[478]^data_in[
479]^data_in[480]^data_in[481]^data_in[484]^data_in[485]^data_in[486]^dat
a_in[487]^data_in[493]^data_in[495]^data_in[499]^data_in[501]^data_in[508
]^data_in[509]^data_in[514]^data_in[515]^data_in[523]^data_in[529]^data_i
n[538]^data_in[539]^data_in[540]^data_in[541]^data_in[542]^data_in[543]^d
ata_in[553]^data_in[555]^data_in[557]^data_in[568]^data_in[569]^data_in[5
72]^data_in[573]^data_in[574]^data_in[575]^data_in[576]^data_in[577]^data
_in[578]^data_in[579]^data_in[580]^data_in[581]^data_in[582]^data_in[584]
^data_in[585]^data_in[586]^data_in[588]^data_in[590]^data_in[592]^data_in
[594]^data_in[596]^data_in[599]^data_in[601]^data_in[602]^data_in[605]^da
ta_in[606]^data_in[609]^data_in[610]^data_in[614]^data_in[615]^data_in[61
7]^data_in[618]^data_in[619]^data_in[621]^data_in[622]^data_in[623]^data_
in[625]^data_in[626]^data_in[627]^data_in[628]^data_in[630]^data_in[631]^
data_in[633]^data_in[636]^data_in[638]^data_in[639]^data_in[641]^data_in[
643]^data_in[644]^data_in[646]^data_in[647]^data_in[651]^data_in[652]^dat
a_in[654]^data_in[655]^data_in[658]^data_in[661]^data_in[666]^data_in[669
]^data_in[673]^data_in[674]^data_in[675]^data_in[681]^data_in[682]^data_i
n[683]^data_in[688]^data_in[690]^data_in[691]^data_in[692]^data_in[693]^d
ata_in[694]^data_in[695]^data_in[697]^data_in[703]^data_in[704]^data_in[7
06]^data_in[708]^data_in[710]^data_in[711]^data_in[718]^data_in[721]^data
_in[722]^data_in[725]^data_in[733]^data_in[734]^data_in[735]^data_in[737]
^data_in[738]^data_in[739]^data_in[748]^data_in[750]^data_in[751]^data_in
[753]^data_in[763]^data_in[764]^data_in[766]^data_in[767]^data_in[778]^da
ta_in[781]^data_in[793]^data_in[794]^data_in[795]^data_in[808]^data_in[81
0]^data_in[811]^data_in[812]^data_in[813]^data_in[814]^data_in[815]^data_
in[816]^data_in[817]^data_in[818]^data_in[819]^data_in[820]^data_in[821]^
data_in[822]^data_in[825]^data_in[827]^data_in[829]^data_in[831]^data_in[
833]^data_in[835]^data_in[837]^data_in[838]^data_in[839]^data_in[842]^dat
a_in[843]^data_in[846]^data_in[847]^data_in[850]^data_in[851]^data_in[853
]^data_in[857]^data_in[861]^data_in[865]^data_in[868]^data_in[869]^data_i
```

```
n[870]^data_in[871]^data_in[876]^data_in[877]^data_in[878]^data_in[879]^d
ata_in[883]^data_in[885]^data_in[891]^data_in[893]^data_in[898]^data_in[8
99]^data_in[906]^data_in[907]^data_in[913]^data_in[921]^data_in[928]^data
_in[929]^data_in[930]^data_in[931]^data_in[932]^data_in[933]^data_in[934]
^data_in[935]^data_in[943]^data_in[945]^data_in[947]^data_in[949]^data_in
[958]^data_in[959]^data_in[962]^data_in[963]^data_in[973]^data_in[977]^da
ta_in[988]^data_in[989]^data_in[990]^data_in[991]^data_in[1003]^data_in[1
005]^data_in[1018]^data_in[1019];


    crc_out[5] =
data_in[0]^data_in[11]^data_in[12]^data_in[14]^data_in[26]^data_in[29]^da
ta_in[30]^data_in[31]^data_in[32]^data_in[33]^data_in[34]^data_in[35]^dat
a_in[36]^data_in[37]^data_in[38]^data_in[39]^data_in[40]^data_in[44]^data
_in[46]^data_in[48]^data_in[50]^data_in[52]^data_in[54]^data_in[59]^data_
in[60]^data_in[63]^data_in[64]^data_in[67]^data_in[68]^data_in[74]^data_i
n[78]^data_in[82]^data_in[89]^data_in[90]^data_in[91]^data_in[92]^data_in
[97]^data_in[98]^data_in[99]^data_in[100]^data_in[101]^data_in[102]^data_
in[103]^data_in[105]^data_in[107]^data_in[108]^data_in[109]^data_in[110]^
data_in[111]^data_in[113]^data_in[115]^data_in[117]^data_in[120]^data_in[
121]^data_in[123]^data_in[125]^data_in[128]^data_in[129]^data_in[132]^dat
a_in[133]^data_in[134]^data_in[136]^data_in[137]^data_in[140]^data_in[141
]^data_in[142]^data_in[144]^data_in[145]^data_in[146]^data_in[148]^data_i
n[151]^data_in[155]^data_in[157]^data_in[158]^data_in[160]^data_in[163]^d
ata_in[164]^data_in[165]^data_in[170]^data_in[171]^data_in[173]^data_in[1
74]^data_in[178]^data_in[180]^data_in[181]^data_in[182]^data_in[183]^data
_in[184]^data_in[186]^data_in[187]^data_in[189]^data_in[190]^data_in[191]
^data_in[192]^data_in[195]^data_in[197]^data_in[199]^data_in[200]^data_in
[202]^data_in[203]^data_in[205]^data_in[207]^data_in[208]^data_in[209]^da
ta_in[212]^data_in[213]^data_in[215]^data_in[216]^data_in[218]^data_in[21
9]^data_in[222]^data_in[224]^data_in[225]^data_in[226]^data_in[228]^data_
in[229]^data_in[231]^data_in[232]^data_in[234]^data_in[235]^data_in[236]^
data_in[239]^data_in[241]^data_in[242]^data_in[244]^data_in[245]^data_in[
247]^data_in[248]^data_in[250]^data_in[254]^data_in[255]^data_in[257]^dat
a_in[258]^data_in[260]^data_in[261]^data_in[263]^data_in[264]^data_in[269
]^data_in[272]^data_in[275]^data_in[278]^data_in[284]^data_in[285]^data_i
n[286]^data_in[290]^data_in[291]^data_in[292]^data_in[299]^data_in[301]^d
ata_in[302]^data_in[303]^data_in[304]^data_in[306]^data_in[314]^data_in[3
15]^data_in[317]^data_in[319]^data_in[320]^data_in[329]^data_in[332]^data
_in[333]^data_in[335]^data_in[336]^data_in[337]^data_in[338]^data_in[339]
^data_in[340]^data_in[341]^data_in[342]^data_in[343]^data_in[347]^data_in
[350]^data_in[352]^data_in[354]^data_in[356]^data_in[358]^data_in[359]^da
ta_in[360]^data_in[361]^data_in[365]^data_in[366]^data_in[369]^data_in[37
0]^data_in[373]^data_in[375]^data_in[380]^data_in[384]^data_in[388]^data_
in[389]^data_in[395]^data_in[396]^data_in[397]^data_in[398]^data_in[403]^
data_in[410]^data_in[412]^data_in[418]^data_in[419]^data_in[420]^data_in[
421]^data_in[422]^data_in[423]^data_in[424]^data_in[427]^data_in[428]^dat
a_in[429]^data_in[430]^data_in[431]^data_in[432]^data_in[434]^data_in[436
]^data_in[438]^data_in[442]^data_in[444]^data_in[446]^data_in[449]^data_i
n[450]^data_in[453]^data_in[454]^data_in[455]^data_in[456]^data_in[459]^d
ata_in[460]^data_in[464]^data_in[468]^data_in[470]^data_in[474]^data_in[4
79]^data_in[480]^data_in[481]^data_in[482]^data_in[485]^data_in[486]^data
_in[487]^data_in[488]^data_in[494]^data_in[496]^data_in[500]^data_in[502]
^data_in[509]^data_in[510]^data_in[515]^data_in[516]^data_in[524]^data_in
[530]^data_in[539]^data_in[540]^data_in[541]^data_in[542]^data_in[543]^da
ta_in[544]^data_in[554]^data_in[556]^data_in[558]^data_in[569]^data_in[57
0]^data_in[573]^data_in[574]^data_in[575]^data_in[576]^data_in[577]^data_
in[578]^data_in[579]^data_in[580]^data_in[581]^data_in[582]^data_in[583]^
data_in[585]^data_in[586]^data_in[587]^data_in[589]^data_in[591]^data_in[
593]^data_in[595]^data_in[597]^data_in[600]^data_in[602]^data_in[603]^dat
a_in[606]^data_in[607]^data_in[610]^data_in[611]^data_in[615]^data_in[616
]^data_in[618]^data_in[619]^data_in[620]^data_in[622]^data_in[623]^data_i
n[624]^data_in[626]^data_in[627]^data_in[628]^data_in[629]^data_in[631]^d
ata_in[632]^data_in[634]^data_in[637]^data_in[639]^data_in[640]^data_in[6
42]^data_in[644]^data_in[645]^data_in[647]^data_in[648]^data_in[652]^data
_in[653]^data_in[655]^data_in[656]^data_in[659]^data_in[662]^data_in[667]
^data_in[670]^data_in[674]^data_in[675]^data_in[676]^data_in[682]^data_in
[683]^data_in[684]^data_in[689]^data_in[691]^data_in[692]^data_in[693]^da
ta_in[694]^data_in[695]^data_in[696]^data_in[698]^data_in[704]^data_in[70
5]^data_in[707]^data_in[709]^data_in[711]^data_in[712]^data_in[719]^data_
in[722]^data_in[723]^data_in[726]^data_in[734]^data_in[735]^data_in[736]^
data_in[738]^data_in[739]^data_in[740]^data_in[749]^data_in[751]^data_in[
752]^data_in[754]^data_in[764]^data_in[765]^data_in[767]^data_in[768]^dat
a_in[779]^data_in[782]^data_in[794]^data_in[795]^data_in[796]^data_in[809
]^data_in[811]^data_in[812]^data_in[813]^data_in[814]^data_in[815]^data_i
n[816]^data_in[817]^data_in[818]^data_in[819]^data_in[820]^data_in[821]^d
ata_in[822]^data_in[823]^data_in[826]^data_in[828]^data_in[830]^data_in[8
32]^data_in[834]^data_in[836]^data_in[838]^data_in[839]^data_in[840]^data
_in[843]^data_in[844]^data_in[847]^data_in[848]^data_in[851]^data_in[852]
^data_in[854]^data_in[858]^data_in[862]^data_in[866]^data_in[869]^data_in
[870]^data_in[871]^data_in[872]^data_in[877]^data_in[878]^data_in[879]^da
ta_in[880]^data_in[884]^data_in[886]^data_in[892]^data_in[894]^data_in[89
9]^data_in[900]^data_in[907]^data_in[908]^data_in[914]^data_in[922]^data_
in[929]^data_in[930]^data_in[931]^data_in[932]^data_in[933]^data_in[934]^
data_in[935]^data_in[936]^data_in[944]^data_in[946]^data_in[948]^data_in[
950]^data_in[959]^data_in[960]^data_in[963]^data_in[964]^data_in[974]^dat
a_in[978]^data_in[989]^data_in[990]^data_in[991]^data_in[992]^data_in[100
4]^data_in[1006]^data_in[1019]^data_in[1020];


    crc_out[4] =
data_in[0]^data_in[1]^data_in[12]^data_in[13]^data_in[15]^data_in[27]^dat
a_in[30]^data_in[31]^data_in[32]^data_in[33]^data_in[34]^data_in[35]^data
_in[36]^data_in[37]^data_in[38]^data_in[39]^data_in[40]^data_in[41]^data_
in[45]^data_in[47]^data_in[49]^data_in[51]^data_in[53]^data_in[55]^data_i
n[60]^data_in[61]^data_in[64]^data_in[65]^data_in[68]^data_in[69]^data_in
[75]^data_in[79]^data_in[83]^data_in[90]^data_in[91]^data_in[92]^data_in[
93]^data_in[98]^data_in[99]^data_in[100]^data_in[101]^data_in[102]^data_i
n[103]^data_in[104]^data_in[106]^data_in[108]^data_in[109]^data_in[110]^d
ata_in[111]^data_in[112]^data_in[114]^data_in[116]^data_in[118]^data_in[1
21]^data_in[122]^data_in[124]^data_in[126]^data_in[129]^data_in[130]^data
_in[133]^data_in[134]^data_in[135]^data_in[137]^data_in[138]^data_in[141]
^data_in[142]^data_in[143]^data_in[145]^data_in[146]^data_in[147]^data_in
[149]^data_in[152]^data_in[156]^data_in[158]^data_in[159]^data_in[161]^da
ta_in[164]^data_in[165]^data_in[166]^data_in[171]^data_in[172]^data_in[17
4]^data_in[175]^data_in[179]^data_in[181]^data_in[182]^data_in[183]^data_
in[184]^data_in[185]^data_in[187]^data_in[188]^data_in[190]^data_in[191]^
data_in[192]^data_in[193]^data_in[196]^data_in[198]^data_in[200]^data_in[
201]^data_in[203]^data_in[204]^data_in[206]^data_in[208]^data_in[209]^dat
a_in[210]^data_in[213]^data_in[214]^data_in[216]^data_in[217]^data_in[219
]^data_in[220]^data_in[223]^data_in[225]^data_in[226]^data_in[227]^data_i
n[229]^data_in[230]^data_in[232]^data_in[233]^data_in[235]^data_in[236]^d
ata_in[237]^data_in[240]^data_in[242]^data_in[243]^data_in[245]^data_in[2
46]^data_in[248]^data_in[249]^data_in[251]^data_in[255]^data_in[256]^data
_in[258]^data_in[259]^data_in[261]^data_in[262]^data_in[264]^data_in[265]
^data_in[270]^data_in[273]^data_in[276]^data_in[279]^data_in[285]^data_in
[286]^data_in[287]^data_in[291]^data_in[292]^data_in[293]^data_in[300]^da
ta_in[302]^data_in[303]^data_in[304]^data_in[305]^data_in[307]^data_in[31
```

```
5]^data_in[316]^data_in[318]^data_in[320]^data_in[321]^data_in[330]^data_
in[333]^data_in[334]^data_in[336]^data_in[337]^data_in[338]^data_in[339]^
data_in[340]^data_in[341]^data_in[342]^data_in[343]^data_in[344]^data_in[
348]^data_in[351]^data_in[353]^data_in[355]^data_in[357]^data_in[359]^dat
a_in[360]^data_in[361]^data_in[362]^data_in[366]^data_in[367]^data_in[370
]^data_in[371]^data_in[374]^data_in[376]^data_in[381]^data_in[385]^data_i
n[389]^data_in[390]^data_in[396]^data_in[397]^data_in[398]^data_in[399]^d
ata_in[404]^data_in[411]^data_in[413]^data_in[419]^data_in[420]^data_in[4
21]^data_in[422]^data_in[423]^data_in[424]^data_in[425]^data_in[428]^data
_in[429]^data_in[430]^data_in[431]^data_in[432]^data_in[433]^data_in[435]
^data_in[437]^data_in[439]^data_in[443]^data_in[445]^data_in[447]^data_in
[450]^data_in[451]^data_in[454]^data_in[455]^data_in[456]^data_in[457]^da
ta_in[460]^data_in[461]^data_in[465]^data_in[469]^data_in[471]^data_in[47
5]^data_in[480]^data_in[481]^data_in[482]^data_in[483]^data_in[486]^data_
in[487]^data_in[488]^data_in[489]^data_in[495]^data_in[497]^data_in[501]^
data_in[503]^data_in[510]^data_in[511]^data_in[516]^data_in[517]^data_in[
525]^data_in[531]^data_in[540]^data_in[541]^data_in[542]^data_in[543]^dat
a_in[544]^data_in[545]^data_in[555]^data_in[557]^data_in[559]^data_in[570
]^data_in[571]^data_in[574]^data_in[575]^data_in[576]^data_in[577]^data_i
n[578]^data_in[579]^data_in[580]^data_in[581]^data_in[582]^data_in[583]^d
ata_in[584]^data_in[586]^data_in[587]^data_in[588]^data_in[590]^data_in[5
92]^data_in[594]^data_in[596]^data_in[598]^data_in[601]^data_in[603]^data
_in[604]^data_in[607]^data_in[608]^data_in[611]^data_in[612]^data_in[616]
^data_in[617]^data_in[619]^data_in[620]^data_in[621]^data_in[623]^data_in
[624]^data_in[625]^data_in[627]^data_in[628]^data_in[629]^data_in[630]^da
ta_in[632]^data_in[633]^data_in[635]^data_in[638]^data_in[640]^data_in[64
1]^data_in[643]^data_in[645]^data_in[646]^data_in[648]^data_in[649]^data_
in[653]^data_in[654]^data_in[656]^data_in[657]^data_in[660]^data_in[663]^
data_in[668]^data_in[671]^data_in[675]^data_in[676]^data_in[677]^data_in[
683]^data_in[684]^data_in[685]^data_in[690]^data_in[692]^data_in[693]^dat
a_in[694]^data_in[695]^data_in[696]^data_in[697]^data_in[699]^data_in[705
]^data_in[706]^data_in[708]^data_in[710]^data_in[712]^data_in[713]^data_i
n[720]^data_in[723]^data_in[724]^data_in[727]^data_in[735]^data_in[736]^d
ata_in[737]^data_in[739]^data_in[740]^data_in[741]^data_in[750]^data_in[7
52]^data_in[753]^data_in[755]^data_in[765]^data_in[766]^data_in[768]^data
_in[769]^data_in[780]^data_in[783]^data_in[795]^data_in[796]^data_in[797]
^data_in[810]^data_in[812]^data_in[813]^data_in[814]^data_in[815]^data_in
[816]^data_in[817]^data_in[818]^data_in[819]^data_in[820]^data_in[821]^da
ta_in[822]^data_in[823]^data_in[824]^data_in[827]^data_in[829]^data_in[83
1]^data_in[833]^data_in[835]^data_in[837]^data_in[839]^data_in[840]^data_
in[841]^data_in[844]^data_in[845]^data_in[848]^data_in[849]^data_in[852]^
data_in[853]^data_in[855]^data_in[859]^data_in[863]^data_in[867]^data_in[
870]^data_in[871]^data_in[872]^data_in[873]^data_in[878]^data_in[879]^dat
a_in[880]^data_in[881]^data_in[885]^data_in[887]^data_in[893]^data_in[895
]^data_in[900]^data_in[901]^data_in[908]^data_in[909]^data_in[915]^data_i
n[923]^data_in[930]^data_in[931]^data_in[932]^data_in[933]^data_in[934]^d
ata_in[935]^data_in[936]^data_in[937]^data_in[945]^data_in[947]^data_in[9
49]^data_in[951]^data_in[960]^data_in[961]^data_in[964]^data_in[965]^data
_in[975]^data_in[979]^data_in[990]^data_in[991]^data_in[992]^data_in[993]
^data_in[1005]^data_in[1007]^data_in[1020]^data_in[1021];



    crc_out[3] =
data_in[0]^data_in[1]^data_in[2]^data_in[13]^data_in[14]^data_in[16]^data
_in[28]^data_in[31]^data_in[32]^data_in[33]^data_in[34]^data_in[35]^data_
in[36]^data_in[37]^data_in[38]^data_in[39]^data_in[40]^data_in[41]^data_i
n[42]^data_in[46]^data_in[48]^data_in[50]^data_in[52]^data_in[54]^data_in
[56]^data_in[61]^data_in[62]^data_in[65]^data_in[66]^data_in[69]^data_in[
70]^data_in[76]^data_in[80]^data_in[84]^data_in[91]^data_in[92]^data_in[9
```

```
3]^data_in[94]^data_in[99]^data_in[100]^data_in[101]^data_in[102]^data_in
[103]^data_in[104]^data_in[105]^data_in[107]^data_in[109]^data_in[110]^da
ta_in[111]^data_in[112]^data_in[113]^data_in[115]^data_in[117]^data_in[11
9]^data_in[122]^data_in[123]^data_in[125]^data_in[127]^data_in[130]^data_
in[131]^data_in[134]^data_in[135]^data_in[136]^data_in[138]^data_in[139]^
data_in[142]^data_in[143]^data_in[144]^data_in[146]^data_in[147]^data_in[
148]^data_in[150]^data_in[153]^data_in[157]^data_in[159]^data_in[160]^dat
a_in[162]^data_in[165]^data_in[166]^data_in[167]^data_in[172]^data_in[173
]^data_in[175]^data_in[176]^data_in[180]^data_in[182]^data_in[183]^data_i
n[184]^data_in[185]^data_in[186]^data_in[188]^data_in[189]^data_in[191]^d
ata_in[192]^data_in[193]^data_in[194]^data_in[197]^data_in[199]^data_in[2
01]^data_in[202]^data_in[204]^data_in[205]^data_in[207]^data_in[209]^data
_in[210]^data_in[211]^data_in[214]^data_in[215]^data_in[217]^data_in[218]
^data_in[220]^data_in[221]^data_in[224]^data_in[226]^data_in[227]^data_in
[228]^data_in[230]^data_in[231]^data_in[233]^data_in[234]^data_in[236]^da
ta_in[237]^data_in[238]^data_in[241]^data_in[243]^data_in[244]^data_in[24
6]^data_in[247]^data_in[249]^data_in[250]^data_in[252]^data_in[256]^data_
in[257]^data_in[259]^data_in[260]^data_in[262]^data_in[263]^data_in[265]^
data_in[266]^data_in[271]^data_in[274]^data_in[277]^data_in[280]^data_in[
286]^data_in[287]^data_in[288]^data_in[292]^data_in[293]^data_in[294]^dat
a_in[301]^data_in[303]^data_in[304]^data_in[305]^data_in[306]^data_in[308
]^data_in[316]^data_in[317]^data_in[319]^data_in[321]^data_in[322]^data_i
n[331]^data_in[334]^data_in[335]^data_in[337]^data_in[338]^data_in[339]^d
ata_in[340]^data_in[341]^data_in[342]^data_in[343]^data_in[344]^data_in[3
45]^data_in[349]^data_in[352]^data_in[354]^data_in[356]^data_in[358]^data
_in[360]^data_in[361]^data_in[362]^data_in[363]^data_in[367]^data_in[368]
^data_in[371]^data_in[372]^data_in[375]^data_in[377]^data_in[382]^data_in
[386]^data_in[390]^data_in[391]^data_in[397]^data_in[398]^data_in[399]^da
ta_in[400]^data_in[405]^data_in[412]^data_in[414]^data_in[420]^data_in[42
1]^data_in[422]^data_in[423]^data_in[424]^data_in[425]^data_in[426]^data_
in[429]^data_in[430]^data_in[431]^data_in[432]^data_in[433]^data_in[434]^
data_in[436]^data_in[438]^data_in[440]^data_in[444]^data_in[446]^data_in[
448]^data_in[451]^data_in[452]^data_in[455]^data_in[456]^data_in[457]^dat
a_in[458]^data_in[461]^data_in[462]^data_in[466]^data_in[470]^data_in[472
]^data_in[476]^data_in[481]^data_in[482]^data_in[483]^data_in[484]^data_i
n[487]^data_in[488]^data_in[489]^data_in[490]^data_in[496]^data_in[498]^d
ata_in[502]^data_in[504]^data_in[511]^data_in[512]^data_in[517]^data_in[5
18]^data_in[526]^data_in[532]^data_in[541]^data_in[542]^data_in[543]^data
_in[544]^data_in[545]^data_in[546]^data_in[556]^data_in[558]^data_in[560]
^data_in[571]^data_in[572]^data_in[575]^data_in[576]^data_in[577]^data_in
[578]^data_in[579]^data_in[580]^data_in[581]^data_in[582]^data_in[583]^da
ta_in[584]^data_in[585]^data_in[587]^data_in[588]^data_in[589]^data_in[59
1]^data_in[593]^data_in[595]^data_in[597]^data_in[599]^data_in[602]^data_
in[604]^data_in[605]^data_in[608]^data_in[609]^data_in[612]^data_in[613]^
data_in[617]^data_in[618]^data_in[620]^data_in[621]^data_in[622]^data_in[
624]^data_in[625]^data_in[626]^data_in[628]^data_in[629]^data_in[630]^dat
a_in[631]^data_in[633]^data_in[634]^data_in[636]^data_in[639]^data_in[641
]^data_in[642]^data_in[644]^data_in[646]^data_in[647]^data_in[649]^data_i
n[650]^data_in[654]^data_in[655]^data_in[657]^data_in[658]^data_in[661]^d
ata_in[664]^data_in[669]^data_in[672]^data_in[676]^data_in[677]^data_in[6
78]^data_in[684]^data_in[685]^data_in[686]^data_in[691]^data_in[693]^data
_in[694]^data_in[695]^data_in[696]^data_in[697]^data_in[698]^data_in[700]
^data_in[706]^data_in[707]^data_in[709]^data_in[711]^data_in[713]^data_in
[714]^data_in[721]^data_in[724]^data_in[725]^data_in[728]^data_in[736]^da
ta_in[737]^data_in[738]^data_in[740]^data_in[741]^data_in[742]^data_in[75
1]^data_in[753]^data_in[754]^data_in[756]^data_in[766]^data_in[767]^data_
in[769]^data_in[770]^data_in[781]^data_in[784]^data_in[796]^data_in[797]^
data_in[798]^data_in[811]^data_in[813]^data_in[814]^data_in[815]^data_in[
816]^data_in[817]^data_in[818]^data_in[819]^data_in[820]^data_in[821]^dat
a_in[822]^data_in[823]^data_in[824]^data_in[825]^data_in[828]^data_in[830
```

```
]^data_in[832]^data_in[834]^data_in[836]^data_in[838]^data_in[840]^data_i
n[841]^data_in[842]^data_in[845]^data_in[846]^data_in[849]^data_in[850]^d
ata_in[853]^data_in[854]^data_in[856]^data_in[860]^data_in[864]^data_in[8
68]^data_in[871]^data_in[872]^data_in[873]^data_in[874]^data_in[879]^data
_in[880]^data_in[881]^data_in[882]^data_in[886]^data_in[888]^data_in[894]
^data_in[896]^data_in[901]^data_in[902]^data_in[909]^data_in[910]^data_in
[916]^data_in[924]^data_in[931]^data_in[932]^data_in[933]^data_in[934]^da
ta_in[935]^data_in[936]^data_in[937]^data_in[938]^data_in[946]^data_in[94
8]^data_in[950]^data_in[952]^data_in[961]^data_in[962]^data_in[965]^data_
in[966]^data_in[976]^data_in[980]^data_in[991]^data_in[992]^data_in[993]^
data_in[994]^data_in[1006]^data_in[1008]^data_in[1021]^data_in[1022];


    crc_out[2] =
data_in[1]^data_in[2]^data_in[3]^data_in[14]^data_in[15]^data_in[17]^data
_in[29]^data_in[32]^data_in[33]^data_in[34]^data_in[35]^data_in[36]^data_
in[37]^data_in[38]^data_in[39]^data_in[40]^data_in[41]^data_in[42]^data_i
n[43]^data_in[47]^data_in[49]^data_in[51]^data_in[53]^data_in[55]^data_in
[57]^data_in[62]^data_in[63]^data_in[66]^data_in[67]^data_in[70]^data_in[
71]^data_in[77]^data_in[81]^data_in[85]^data_in[92]^data_in[93]^data_in[9
4]^data_in[95]^data_in[100]^data_in[101]^data_in[102]^data_in[103]^data_i
n[104]^data_in[105]^data_in[106]^data_in[108]^data_in[110]^data_in[111]^d
ata_in[112]^data_in[113]^data_in[114]^data_in[116]^data_in[118]^data_in[1
20]^data_in[123]^data_in[124]^data_in[126]^data_in[128]^data_in[131]^data
_in[132]^data_in[135]^data_in[136]^data_in[137]^data_in[139]^data_in[140]
^data_in[143]^data_in[144]^data_in[145]^data_in[147]^data_in[148]^data_in
[149]^data_in[151]^data_in[154]^data_in[158]^data_in[160]^data_in[161]^da
ta_in[163]^data_in[166]^data_in[167]^data_in[168]^data_in[173]^data_in[17
4]^data_in[176]^data_in[177]^data_in[181]^data_in[183]^data_in[184]^data_
in[185]^data_in[186]^data_in[187]^data_in[189]^data_in[190]^data_in[192]^
data_in[193]^data_in[194]^data_in[195]^data_in[198]^data_in[200]^data_in[
202]^data_in[203]^data_in[205]^data_in[206]^data_in[208]^data_in[210]^dat
a_in[211]^data_in[212]^data_in[215]^data_in[216]^data_in[218]^data_in[219
]^data_in[221]^data_in[222]^data_in[225]^data_in[227]^data_in[228]^data_i
n[229]^data_in[231]^data_in[232]^data_in[234]^data_in[235]^data_in[237]^d
ata_in[238]^data_in[239]^data_in[242]^data_in[244]^data_in[245]^data_in[2
47]^data_in[248]^data_in[250]^data_in[251]^data_in[253]^data_in[257]^data
_in[258]^data_in[260]^data_in[261]^data_in[263]^data_in[264]^data_in[266]
^data_in[267]^data_in[272]^data_in[275]^data_in[278]^data_in[281]^data_in
[287]^data_in[288]^data_in[289]^data_in[293]^data_in[294]^data_in[295]^da
ta_in[302]^data_in[304]^data_in[305]^data_in[306]^data_in[307]^data_in[30
9]^data_in[317]^data_in[318]^data_in[320]^data_in[322]^data_in[323]^data_
in[332]^data_in[335]^data_in[336]^data_in[338]^data_in[339]^data_in[340]^
data_in[341]^data_in[342]^data_in[343]^data_in[344]^data_in[345]^data_in[
346]^data_in[350]^data_in[353]^data_in[355]^data_in[357]^data_in[359]^dat
a_in[361]^data_in[362]^data_in[363]^data_in[364]^data_in[368]^data_in[369
]^data_in[372]^data_in[373]^data_in[376]^data_in[378]^data_in[383]^data_i
n[387]^data_in[391]^data_in[392]^data_in[398]^data_in[399]^data_in[400]^d
ata_in[401]^data_in[406]^data_in[413]^data_in[415]^data_in[421]^data_in[4
22]^data_in[423]^data_in[424]^data_in[425]^data_in[426]^data_in[427]^data
_in[430]^data_in[431]^data_in[432]^data_in[433]^data_in[434]^data_in[435]
^data_in[437]^data_in[439]^data_in[441]^data_in[445]^data_in[447]^data_in
[449]^data_in[452]^data_in[453]^data_in[456]^data_in[457]^data_in[458]^da
ta_in[459]^data_in[462]^data_in[463]^data_in[467]^data_in[471]^data_in[47
3]^data_in[477]^data_in[482]^data_in[483]^data_in[484]^data_in[485]^data_
in[488]^data_in[489]^data_in[490]^data_in[491]^data_in[497]^data_in[499]^
data_in[503]^data_in[505]^data_in[512]^data_in[513]^data_in[518]^data_in[
519]^data_in[527]^data_in[533]^data_in[542]^data_in[543]^data_in[544]^dat
a_in[545]^data_in[546]^data_in[547]^data_in[557]^data_in[559]^data_in[561
```

```
]^data_in[572]^data_in[573]^data_in[576]^data_in[577]^data_in[578]^data_i
n[579]^data_in[580]^data_in[581]^data_in[582]^data_in[583]^data_in[584]^d
ata_in[585]^data_in[586]^data_in[588]^data_in[589]^data_in[590]^data_in[5
92]^data_in[594]^data_in[596]^data_in[598]^data_in[600]^data_in[603]^data
_in[605]^data_in[606]^data_in[609]^data_in[610]^data_in[613]^data_in[614]
^data_in[618]^data_in[619]^data_in[621]^data_in[622]^data_in[623]^data_in
[625]^data_in[626]^data_in[627]^data_in[629]^data_in[630]^data_in[631]^da
ta_in[632]^data_in[634]^data_in[635]^data_in[637]^data_in[640]^data_in[64
2]^data_in[643]^data_in[645]^data_in[647]^data_in[648]^data_in[650]^data_
in[651]^data_in[655]^data_in[656]^data_in[658]^data_in[659]^data_in[662]^
data_in[665]^data_in[670]^data_in[673]^data_in[677]^data_in[678]^data_in[
679]^data_in[685]^data_in[686]^data_in[687]^data_in[692]^data_in[694]^dat
a_in[695]^data_in[696]^data_in[697]^data_in[698]^data_in[699]^data_in[701
]^data_in[707]^data_in[708]^data_in[710]^data_in[712]^data_in[714]^data_i
n[715]^data_in[722]^data_in[725]^data_in[726]^data_in[729]^data_in[737]^d
ata_in[738]^data_in[739]^data_in[741]^data_in[742]^data_in[743]^data_in[7
52]^data_in[754]^data_in[755]^data_in[757]^data_in[767]^data_in[768]^data
_in[770]^data_in[771]^data_in[782]^data_in[785]^data_in[797]^data_in[798]
^data_in[799]^data_in[812]^data_in[814]^data_in[815]^data_in[816]^data_in
[817]^data_in[818]^data_in[819]^data_in[820]^data_in[821]^data_in[822]^da
ta_in[823]^data_in[824]^data_in[825]^data_in[826]^data_in[829]^data_in[83
1]^data_in[833]^data_in[835]^data_in[837]^data_in[839]^data_in[841]^data_
in[842]^data_in[843]^data_in[846]^data_in[847]^data_in[850]^data_in[851]^
data_in[854]^data_in[855]^data_in[857]^data_in[861]^data_in[865]^data_in[
869]^data_in[872]^data_in[873]^data_in[874]^data_in[875]^data_in[880]^dat
a_in[881]^data_in[882]^data_in[883]^data_in[887]^data_in[889]^data_in[895
]^data_in[897]^data_in[902]^data_in[903]^data_in[910]^data_in[911]^data_i
n[917]^data_in[925]^data_in[932]^data_in[933]^data_in[934]^data_in[935]^d
ata_in[936]^data_in[937]^data_in[938]^data_in[939]^data_in[947]^data_in[9
49]^data_in[951]^data_in[953]^data_in[962]^data_in[963]^data_in[966]^data
_in[967]^data_in[977]^data_in[981]^data_in[992]^data_in[993]^data_in[994]
^data_in[995]^data_in[1007]^data_in[1009]^data_in[1022]^data_in[1023];


    crc_out[1] =
data_in[1]^data_in[2]^data_in[15]^data_in[16]^data_in[17]^data_in[19]^dat
a_in[21]^data_in[23]^data_in[25]^data_in[27]^data_in[29]^data_in[30]^data
_in[31]^data_in[32]^data_in[35]^data_in[36]^data_in[39]^data_in[40]^data_
in[43]^data_in[44]^data_in[45]^data_in[46]^data_in[47]^data_in[49]^data_i
n[50]^data_in[51]^data_in[53]^data_in[54]^data_in[55]^data_in[57]^data_in
[58]^data_in[59]^data_in[60]^data_in[61]^data_in[62]^data_in[67]^data_in[
68]^data_in[69]^data_in[70]^data_in[78]^data_in[80]^data_in[86]^data_in[8
8]^data_in[90]^data_in[92]^data_in[93]^data_in[96]^data_in[98]^data_in[10
0]^data_in[101]^data_in[104]^data_in[105]^data_in[109]^data_in[112]^data_
in[113]^data_in[117]^data_in[121]^data_in[123]^data_in[124]^data_in[126]^
data_in[127]^data_in[128]^data_in[130]^data_in[133]^data_in[135]^data_in[
136]^data_in[140]^data_in[141]^data_in[142]^data_in[143]^data_in[146]^dat
a_in[149]^data_in[150]^data_in[151]^data_in[153]^data_in[159]^data_in[162
]^data_in[167]^data_in[168]^data_in[170]^data_in[172]^data_in[175]^data_i
n[178]^data_in[180]^data_in[184]^data_in[185]^data_in[188]^data_in[191]^d
ata_in[194]^data_in[195]^data_in[197]^data_in[201]^data_in[204]^data_in[2
07]^data_in[211]^data_in[212]^data_in[214]^data_in[217]^data_in[220]^data
_in[223]^data_in[225]^data_in[226]^data_in[227]^data_in[230]^data_in[233]
^data_in[236]^data_in[239]^data_in[240]^data_in[241]^data_in[242]^data_in
[244]^data_in[245]^data_in[247]^data_in[248]^data_in[250]^data_in[251]^da
ta_in[253]^data_in[254]^data_in[255]^data_in[256]^data_in[257]^data_in[26
1]^data_in[262]^data_in[263]^data_in[267]^data_in[268]^data_in[269]^data_
in[270]^data_in[271]^data_in[272]^data_in[274]^data_in[279]^data_in[281]^
data_in[282]^data_in[283]^data_in[284]^data_in[285]^data_in[286]^data_in[
287]^data_in[290]^data_in[292]^data_in[295]^data_in[296]^data_in[297]^dat
a_in[298]^data_in[299]^data_in[300]^data_in[301]^data_in[302]^data_in[304
]^data_in[305]^data_in[308]^data_in[318]^data_in[319]^data_in[320]^data_i
n[322]^data_in[323]^data_in[325]^data_in[327]^data_in[329]^data_in[331]^d
ata_in[336]^data_in[337]^data_in[338]^data_in[341]^data_in[342]^data_in[3
45]^data_in[346]^data_in[348]^data_in[350]^data_in[351]^data_in[352]^data
_in[353]^data_in[355]^data_in[356]^data_in[357]^data_in[359]^data_in[360]
^data_in[361]^data_in[364]^data_in[365]^data_in[366]^data_in[367]^data_in
[368]^data_in[373]^data_in[374]^data_in[375]^data_in[376]^data_in[378]^da
ta_in[379]^data_in[380]^data_in[381]^data_in[382]^data_in[383]^data_in[38
5]^data_in[387]^data_in[388]^data_in[389]^data_in[390]^data_in[391]^data_
in[399]^data_in[400]^data_in[407]^data_in[409]^data_in[411]^data_in[413]^
data_in[414]^data_in[415]^data_in[417]^data_in[419]^data_in[421]^data_in[
422]^data_in[425]^data_in[426]^data_in[431]^data_in[432]^data_in[435]^dat
a_in[436]^data_in[437]^data_in[439]^data_in[440]^data_in[441]^data_in[443
]^data_in[445]^data_in[446]^data_in[447]^data_in[449]^data_in[450]^data_i
n[451]^data_in[452]^data_in[457]^data_in[458]^data_in[463]^data_in[464]^d
ata_in[465]^data_in[466]^data_in[467]^data_in[469]^data_in[471]^data_in[4
72]^data_in[473]^data_in[475]^data_in[477]^data_in[478]^data_in[479]^data
_in[480]^data_in[481]^data_in[482]^data_in[485]^data_in[486]^data_in[487]
^data_in[488]^data_in[491]^data_in[492]^data_in[493]^data_in[494]^data_in
[495]^data_in[496]^data_in[497]^data_in[499]^data_in[500]^data_in[501]^da
ta_in[502]^data_in[503]^data_in[505]^data_in[506]^data_in[507]^data_in[50
8]^data_in[509]^data_in[510]^data_in[511]^data_in[512]^data_in[519]^data_
in[520]^data_in[521]^data_in[522]^data_in[523]^data_in[524]^data_in[525]^
data_in[526]^data_in[527]^data_in[529]^data_in[531]^data_in[533]^data_in[
534]^data_in[535]^data_in[536]^data_in[537]^data_in[538]^data_in[539]^dat
a_in[540]^data_in[541]^data_in[542]^data_in[545]^data_in[546]^data_in[558
]^data_in[562]^data_in[564]^data_in[566]^data_in[568]^data_in[570]^data_i
n[572]^data_in[573]^data_in[575]^data_in[578]^data_in[579]^data_in[582]^d
ata_in[583]^data_in[586]^data_in[587]^data_in[588]^data_in[591]^data_in[5
95]^data_in[599]^data_in[604]^data_in[607]^data_in[609]^data_in[610]^data
_in[612]^data_in[615]^data_in[617]^data_in[620]^data_in[623]^data_in[624]
^data_in[625]^data_in[628]^data_in[631]^data_in[632]^data_in[634]^data_in
[635]^data_in[637]^data_in[638]^data_in[639]^data_in[640]^data_in[642]^da
ta_in[643]^data_in[645]^data_in[646]^data_in[647]^data_in[651]^data_in[65
2]^data_in[653]^data_in[654]^data_in[655]^data_in[659]^data_in[660]^data_
in[661]^data_in[662]^data_in[664]^data_in[671]^data_in[673]^data_in[674]^
data_in[675]^data_in[676]^data_in[677]^data_in[680]^data_in[682]^data_in[
684]^data_in[687]^data_in[688]^data_in[689]^data_in[690]^data_in[691]^dat
a_in[692]^data_in[694]^data_in[695]^data_in[698]^data_in[699]^data_in[701
]^data_in[702]^data_in[703]^data_in[704]^data_in[705]^data_in[706]^data_i
n[707]^data_in[711]^data_in[715]^data_in[716]^data_in[717]^data_in[718]^d
ata_in[719]^data_in[720]^data_in[721]^data_in[722]^data_in[724]^data_in[7
27]^data_in[729]^data_in[730]^data_in[731]^data_in[732]^data_in[733]^data
_in[734]^data_in[735]^data_in[736]^data_in[737]^data_in[740]^data_in[743]
^data_in[744]^data_in[745]^data_in[746]^data_in[747]^data_in[748]^data_in
[749]^data_in[750]^data_in[751]^data_in[752]^data_in[754]^data_in[755]^da
ta_in[757]^data_in[758]^data_in[759]^data_in[760]^data_in[761]^data_in[76
2]^data_in[763]^data_in[764]^data_in[765]^data_in[766]^data_in[767]^data_
in[771]^data_in[772]^data_in[773]^data_in[774]^data_in[775]^data_in[776]^
data_in[777]^data_in[778]^data_in[779]^data_in[780]^data_in[781]^data_in[
782]^data_in[784]^data_in[798]^data_in[799]^data_in[801]^data_in[803]^dat
a_in[805]^data_in[807]^data_in[809]^data_in[811]^data_in[815]^data_in[816
]^data_in[819]^data_in[820]^data_in[823]^data_in[824]^data_in[827]^data_i
n[829]^data_in[830]^data_in[831]^data_in[833]^data_in[834]^data_in[835]^d
ata_in[837]^data_in[838]^data_in[839]^data_in[841]^data_in[842]^data_in[8
47]^data_in[848]^data_in[849]^data_in[850]^data_in[855]^data_in[856]^data
_in[857]^data_in[859]^data_in[861]^data_in[862]^data_in[863]^data_in[864]
^data_in[865]^data_in[867]^data_in[869]^data_in[870]^data_in[871]^data_in
```

```
[872]^data_in[875]^data_in[876]^data_in[877]^data_in[878]^data_in[879]^da
ta_in[880]^data_in[883]^data_in[884]^data_in[885]^data_in[886]^data_in[88
7]^data_in[889]^data_in[890]^data_in[891]^data_in[892]^data_in[893]^data_
in[894]^data_in[895]^data_in[897]^data_in[898]^data_in[899]^data_in[900]^
data_in[901]^data_in[902]^data_in[911]^data_in[912]^data_in[913]^data_in[
914]^data_in[915]^data_in[916]^data_in[917]^data_in[919]^data_in[921]^dat
a_in[923]^data_in[925]^data_in[926]^data_in[927]^data_in[928]^data_in[929
]^data_in[930]^data_in[931]^data_in[932]^data_in[935]^data_in[936]^data_i
n[939]^data_in[940]^data_in[941]^data_in[942]^data_in[943]^data_in[944]^d
ata_in[945]^data_in[946]^data_in[947]^data_in[949]^data_in[950]^data_in[9
51]^data_in[953]^data_in[954]^data_in[955]^data_in[956]^data_in[957]^data
_in[958]^data_in[959]^data_in[960]^data_in[961]^data_in[962]^data_in[967]
^data_in[968]^data_in[969]^data_in[970]^data_in[971]^data_in[972]^data_in
[973]^data_in[974]^data_in[975]^data_in[976]^data_in[977]^data_in[979]^da
ta_in[981]^data_in[982]^data_in[983]^data_in[984]^data_in[985]^data_in[98
6]^data_in[987]^data_in[988]^data_in[989]^data_in[990]^data_in[991]^data_
in[992]^data_in[995]^data_in[996]^data_in[997]^data_in[998]^data_in[999]^
data_in[1000]^data_in[1001]^data_in[1002]^data_in[1003]^data_in[1004]^dat
a_in[1005]^data_in[1006]^data_in[1007]^data_in[1009]^data_in[1010]^data_i
n[1011]^data_in[1012]^data_in[1013]^data_in[1014]^data_in[1015]^data_in[1
016]^data_in[1017]^data_in[1018]^data_in[1019]^data_in[1020]^data_in[1021
]^data_in[1022];


    crc_out[0] =
data_in[0]^data_in[2]^data_in[3]^data_in[16]^data_in[17]^data_in[18]^data
_in[20]^data_in[22]^data_in[24]^data_in[26]^data_in[28]^data_in[30]^data_
in[31]^data_in[32]^data_in[33]^data_in[36]^data_in[37]^data_in[40]^data_i
n[41]^data_in[44]^data_in[45]^data_in[46]^data_in[47]^data_in[48]^data_in
[50]^data_in[51]^data_in[52]^data_in[54]^data_in[55]^data_in[56]^data_in[
58]^data_in[59]^data_in[60]^data_in[61]^data_in[62]^data_in[63]^data_in[6
8]^data_in[69]^data_in[70]^data_in[71]^data_in[79]^data_in[81]^data_in[87
]^data_in[89]^data_in[91]^data_in[93]^data_in[94]^data_in[97]^data_in[99]
^data_in[101]^data_in[102]^data_in[105]^data_in[106]^data_in[110]^data_in
[113]^data_in[114]^data_in[118]^data_in[122]^data_in[124]^data_in[125]^da
ta_in[127]^data_in[128]^data_in[129]^data_in[131]^data_in[134]^data_in[13
6]^data_in[137]^data_in[141]^data_in[142]^data_in[143]^data_in[144]^data_
in[147]^data_in[150]^data_in[151]^data_in[152]^data_in[154]^data_in[160]^
data_in[163]^data_in[168]^data_in[169]^data_in[171]^data_in[173]^data_in[
176]^data_in[179]^data_in[181]^data_in[185]^data_in[186]^data_in[189]^dat
a_in[192]^data_in[195]^data_in[196]^data_in[198]^data_in[202]^data_in[205
]^data_in[208]^data_in[212]^data_in[213]^data_in[215]^data_in[218]^data_i
n[221]^data_in[224]^data_in[226]^data_in[227]^data_in[228]^data_in[231]^d
ata_in[234]^data_in[237]^data_in[240]^data_in[241]^data_in[242]^data_in[2
43]^data_in[245]^data_in[246]^data_in[248]^data_in[249]^data_in[251]^data
_in[252]^data_in[254]^data_in[255]^data_in[256]^data_in[257]^data_in[258]
^data_in[262]^data_in[263]^data_in[264]^data_in[268]^data_in[269]^data_in
[270]^data_in[271]^data_in[272]^data_in[273]^data_in[275]^data_in[280]^da
ta_in[282]^data_in[283]^data_in[284]^data_in[285]^data_in[286]^data_in[28
7]^data_in[288]^data_in[291]^data_in[293]^data_in[296]^data_in[297]^data_
in[298]^data_in[299]^data_in[300]^data_in[301]^data_in[302]^data_in[303]^
data_in[305]^data_in[306]^data_in[309]^data_in[319]^data_in[320]^data_in[
321]^data_in[323]^data_in[324]^data_in[326]^data_in[328]^data_in[330]^dat
a_in[332]^data_in[337]^data_in[338]^data_in[339]^data_in[342]^data_in[343
]^data_in[346]^data_in[347]^data_in[349]^data_in[351]^data_in[352]^data_i
n[353]^data_in[354]^data_in[356]^data_in[357]^data_in[358]^data_in[360]^d
ata_in[361]^data_in[362]^data_in[365]^data_in[366]^data_in[367]^data_in[3
68]^data_in[369]^data_in[374]^data_in[375]^data_in[376]^data_in[377]^data
_in[379]^data_in[380]^data_in[381]^data_in[382]^data_in[383]^data_in[384]
```

```
^data_in[386]^data_in[388]^data_in[389]^data_in[390]^data_in[391]^data_in
[392]^data_in[400]^data_in[401]^data_in[408]^data_in[410]^data_in[412]^da
ta_in[414]^data_in[415]^data_in[416]^data_in[418]^data_in[420]^data_in[42
2]^data_in[423]^data_in[426]^data_in[427]^data_in[432]^data_in[433]^data_
in[436]^data_in[437]^data_in[438]^data_in[440]^data_in[441]^data_in[442]^
data_in[444]^data_in[446]^data_in[447]^data_in[448]^data_in[450]^data_in[
451]^data_in[452]^data_in[453]^data_in[458]^data_in[459]^data_in[464]^dat
a_in[465]^data_in[466]^data_in[467]^data_in[468]^data_in[470]^data_in[472
]^data_in[473]^data_in[474]^data_in[476]^data_in[478]^data_in[479]^data_i
n[480]^data_in[481]^data_in[482]^data_in[483]^data_in[486]^data_in[487]^d
ata_in[488]^data_in[489]^data_in[492]^data_in[493]^data_in[494]^data_in[4
95]^data_in[496]^data_in[497]^data_in[498]^data_in[500]^data_in[501]^data
_in[502]^data_in[503]^data_in[504]^data_in[506]^data_in[507]^data_in[508]
^data_in[509]^data_in[510]^data_in[511]^data_in[512]^data_in[513]^data_in
[520]^data_in[521]^data_in[522]^data_in[523]^data_in[524]^data_in[525]^da
ta_in[526]^data_in[527]^data_in[528]^data_in[530]^data_in[532]^data_in[53
4]^data_in[535]^data_in[536]^data_in[537]^data_in[538]^data_in[539]^data_
in[540]^data_in[541]^data_in[542]^data_in[543]^data_in[546]^data_in[547]^
data_in[559]^data_in[563]^data_in[565]^data_in[567]^data_in[569]^data_in[
571]^data_in[573]^data_in[574]^data_in[576]^data_in[579]^data_in[580]^dat
a_in[583]^data_in[584]^data_in[587]^data_in[588]^data_in[589]^data_in[592
]^data_in[596]^data_in[600]^data_in[605]^data_in[608]^data_in[610]^data_i
n[611]^data_in[613]^data_in[616]^data_in[618]^data_in[621]^data_in[624]^d
ata_in[625]^data_in[626]^data_in[629]^data_in[632]^data_in[633]^data_in[6
35]^data_in[636]^data_in[638]^data_in[639]^data_in[640]^data_in[641]^data
_in[643]^data_in[644]^data_in[646]^data_in[647]^data_in[648]^data_in[652]
^data_in[653]^data_in[654]^data_in[655]^data_in[656]^data_in[660]^data_in
[661]^data_in[662]^data_in[663]^data_in[665]^data_in[672]^data_in[674]^da
ta_in[675]^data_in[676]^data_in[677]^data_in[678]^data_in[681]^data_in[68
3]^data_in[685]^data_in[688]^data_in[689]^data_in[690]^data_in[691]^data_
in[692]^data_in[693]^data_in[695]^data_in[696]^data_in[699]^data_in[700]^
data_in[702]^data_in[703]^data_in[704]^data_in[705]^data_in[706]^data_in[
707]^data_in[708]^data_in[712]^data_in[716]^data_in[717]^data_in[718]^dat
a_in[719]^data_in[720]^data_in[721]^data_in[722]^data_in[723]^data_in[725
]^data_in[728]^data_in[730]^data_in[731]^data_in[732]^data_in[733]^data_i
n[734]^data_in[735]^data_in[736]^data_in[737]^data_in[738]^data_in[741]^d
ata_in[744]^data_in[745]^data_in[746]^data_in[747]^data_in[748]^data_in[7
49]^data_in[750]^data_in[751]^data_in[752]^data_in[753]^data_in[755]^data
_in[756]^data_in[758]^data_in[759]^data_in[760]^data_in[761]^data_in[762]
^data_in[763]^data_in[764]^data_in[765]^data_in[766]^data_in[767]^data_in
[768]^data_in[772]^data_in[773]^data_in[774]^data_in[775]^data_in[776]^da
ta_in[777]^data_in[778]^data_in[779]^data_in[780]^data_in[781]^data_in[78
2]^data_in[783]^data_in[785]^data_in[799]^data_in[800]^data_in[802]^data_
in[804]^data_in[806]^data_in[808]^data_in[810]^data_in[812]^data_in[816]^
data_in[817]^data_in[820]^data_in[821]^data_in[824]^data_in[825]^data_in[
828]^data_in[830]^data_in[831]^data_in[832]^data_in[834]^data_in[835]^dat
a_in[836]^data_in[838]^data_in[839]^data_in[840]^data_in[842]^data_in[843
]^data_in[848]^data_in[849]^data_in[850]^data_in[851]^data_in[856]^data_i
n[857]^data_in[858]^data_in[860]^data_in[862]^data_in[863]^data_in[864]^d
ata_in[865]^data_in[866]^data_in[868]^data_in[870]^data_in[871]^data_in[8
72]^data_in[873]^data_in[876]^data_in[877]^data_in[878]^data_in[879]^data
_in[880]^data_in[881]^data_in[884]^data_in[885]^data_in[886]^data_in[887]
^data_in[888]^data_in[890]^data_in[891]^data_in[892]^data_in[893]^data_in
[894]^data_in[895]^data_in[896]^data_in[898]^data_in[899]^data_in[900]^da
ta_in[901]^data_in[902]^data_in[903]^data_in[912]^data_in[913]^data_in[91
4]^data_in[915]^data_in[916]^data_in[917]^data_in[918]^data_in[920]^data_
in[922]^data_in[924]^data_in[926]^data_in[927]^data_in[928]^data_in[929]^
data_in[930]^data_in[931]^data_in[932]^data_in[933]^data_in[936]^data_in[
937]^data_in[940]^data_in[941]^data_in[942]^data_in[943]^data_in[944]^dat
a_in[945]^data_in[946]^data_in[947]^data_in[948]^data_in[950]^data_in[951
]^data_in[952]^data_in[954]^data_in[955]^data_in[956]^data_in[957]^data_i
n[958]^data_in[959]^data_in[960]^data_in[961]^data_in[962]^data_in[963]^d
ata_in[968]^data_in[969]^data_in[970]^data_in[971]^data_in[972]^data_in[9
73]^data_in[974]^data_in[975]^data_in[976]^data_in[977]^data_in[978]^data
_in[980]^data_in[982]^data_in[983]^data_in[984]^data_in[985]^data_in[986]
^data_in[987]^data_in[988]^data_in[989]^data_in[990]^data_in[991]^data_in
[992]^data_in[993]^data_in[996]^data_in[997]^data_in[998]^data_in[999]^da
ta_in[1000]^data_in[1001]^data_in[1002]^data_in[1003]^data_in[1004]^data_
in[1005]^data_in[1006]^data_in[1007]^data_in[1008]^data_in[1010]^data_in[
1011]^data_in[1012]^data_in[1013]^data_in[1014]^data_in[1015]^data_in[101
6]^data_in[1017]^data_in[1018]^data_in[1019]^data_in[1020]^data_in[1021]^
data_in[1022]^data_in[1023];


end


endmodule
```

C AIB Interoperability

Implementations are permitted to design a superset stack to be interoperable with UCIe/AIB PHY. This section details the UCIe interoperability criteria with AIB.

C.1 AIB Signal Mapping

C.1.1 Data path

Data path signal mapping for AIB 2.0 and AIB 1.0 are shown in Table 118 and Table 119 respectively. AIB sideband is sent over an asynchronous path on UCIe main band.

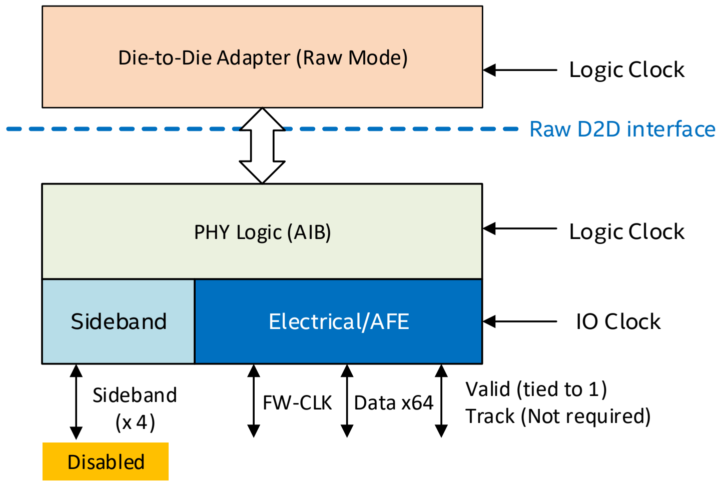
C.1.2 Always high Valid

Always high Valid is an optional feature that is only applicable to AIB interoperability applications. This must be negotiated prior to main band Link training through parameter exchange. Raw mode must be used in such applications.

C.1.3 Sideband

AIB sideband is sent using UCIe main band signals. UCIe sideband is not required in AIB interoperability mode and it is disabled (Transmitters are Hi-Z and Receivers are disabled)

Figure 129. AIB interoperability



C.1.4 Raw Die-to-Die interface

AIB Phy logic block shown in Figure 129 presents a subset of RDI to next layer up.

Note: More details will be shown in a later revision of this specification

Table 118. AIB 2.0 Datapath mapping for Advanced Package

UCIe Interface	AIB 2.0	Note
TXDATA[39:0]	TX[39:0]	
TXDATA[47:40]	AIB Sideband Tx	Asynchronous path
TXDATA[63:48]	NA	Disabled (Hi-Z)
RXDATA[39:0]	RX[39:0]	
RXDATA[47:40]	AIB Sideband Rx	Asynchronous path
RXDATA[63:48]	NA	
TXDATASB	NA	Disabled (Hi-Z)
RXDATASB		
TXCKSB		
RXCKSB		
TXDATASBRD		
RXDATASBRD		

Table 119. AIB 1.0 Datapath mapping for Advanced Package

UCIe Interface	AIB 1.0	Note
TXDATA[19:0]	TX[19:0]	
TXDATA[42:20]	AIB Sideband Tx	Asynchronous path
TXDATA[63:43]	NA	Disabled (Hi-Z)
RXDATA[19:0]	RX[19:0]	
RXDATA[42:20]	AIB Sideband Rx	Asynchronous path
RXDATA[63:43]	NA	
TXDATASB	NA	Disabled (Hi-Z)
RXDATASB		
TXCKSB		
RXCKSB		
TXDATASBRD		
RXDATASBRD		

C.2 Initialization

AIB Phy logic block shown in Figure 129 contains all the AIB Link logic and state machines. Please see AIB specification (Section 2 and Section 3) for initialization flow.

C.3 Bump Map

Note: More details will be shown in a future revision this specification