XpressRICH3

Reference Manual

Version 1.9.6 October 2018

XpressRICH3
Reference Manual

Document Change History

Date	Version Number	Change
October 2018	1.9.6	• Updated Receive and Transmit buffer descriptions. • Updated Function-Level Reset description.
August 2018	1.9.5	• Added information about supported topologies. • Updated ARI description.
April 2018	1.9.4	• Added support for 512-bit TL datapath. • Updated tl_rx_bardec0 and tl_rx_mchit0 signal descriptions. • Added constant G_NPBUF_WIDTH. • Updated G_NPBUF_SIZE description.
February 2018	1.9.3	• Compliant with PCIe Specification Revision 4.0. • Added support for Lame Margining at the Receiver. • Updated Power Management support. • Updated Test Interface. • Updated Transaction Layer Bypass description.
October 2017	1.9.2	• Updated equalization interface parameters. • Updated descriptions of G_RXBUF_SIZE and G_RXBUF_LATENCY. • Updated description of k_pipe variable.
July 2017	1.9.1	• Added support for Tx/Rx buffer latency.
June 2017	1.9.0	• Added support for 512 virtual functions.
May 2017	1.8.1	• Added support for PASID. • Updated sections on Receive and Transmit Interfaces. • Separated Register Content into Physical and Virtual Functions sections. • Updated core variables. • Updated test interface signals. • Updated Received Error Processing and TLP Event Processing sections.
February 2017	1.8.0	• Updated Deskew Block information. • Updated Transaction Layer Bypass signals. • Updated core constants. • Updated Transmit interface signals. • Updated Reporting Interface signals. • Updated Test Interface signals. • Added Lane Turn-Off support. • Updated Configuration Space registers. • Updated Tx Error Management.
September 2016	1.7.5	• Added support for 64b PIPE. • Added and updated PHY Interface signals and description. • Added new bit for test_out.

Date	Version Number	Change
June 2016	1.7.4	• Added support for Crosslink. • Added support for DPC. • Added and updated PHY Interface signals and description. • Updated TLB and Bridge Interface signals and description. • Added and updated Core Variables description. • Added test_out signals.
April 2016	1.7.3	• Added Physical Function Support for Switch Upstream Port. • Added Support for Multicast for all PCIe devices • Added & Updated Core Constants and Core Variables description. • Added PIPE modes. • Added new debug bits for test_out signal. • Updated tl_rx_bardec0 signal and tl_report_status0 signal.
February 2016	1.7.2	• Updated SHB_TRANS_LTSSM_DIRECT signal, and added SHB_PHYDL_L2_P2 and SHB_PHYDL_L2_P2 signals. • Updated Core Constants description. • Added signals pl_equ_phase, pipe_rxeqinprogress, tl_rx_credhsize, and pl_equ_phase. • Added new Rx Stream mode waveform. • Updated Power Management signals and tl_report_status0 signal. • Removed tl_cfg_pslverr0 from APB Configuration signals. • Replaced tl_cfg_devinfo0, tl_cfg_rbsinfo, and tl_cfg_wininfo signals with tl_cfg_regs for Direct Access Configuration Registers. • Added Alternate Routing-ID Interpretation (ARI) capability. • Updated Unsynchronized Paths table in Appendix D.2.
December 2015	1.7.1.4	• Added information about deskew. • Added QUERY_TIMEOUT to preset-to-coefficient values. • Updated k_pexconf[52] description.
November 2015	1.7.1.3	• Added k_gen[37] description. • Added Test_In[21] and Test_Out[263 - 264] descriptions. • Added Note about CDC and clock gating implementation. • Updated tables in Appendices C and D.
August 2015	1.7.1.1	• Updated tlb_trans and tlb_phydl signals. • Updated RX/Tx Latency and cut-through. • Updated TL_CLK and TL_PM_CLKSTATUS signals and added TL_CLKP signal. • Added TL_RSTNP and PL_RSTNP signals. • Added TL_REPORT_STATUS[17].
June 2015	1.7.1	• Added autonomous bandwidth change support. • Updated tlb_trans and tlb_phydl signals. • Updated section on Equalization Problems. • Added pipe_invalidrequest signal to equalization interface.
May 2015	1.7.0	• Added signals K_FINETUNE_ERR and TL_CFG_PSLVERR. • Added core constant G_DATAPATH. • Updated list of Other Publications.

Date	Version Number	Change
February 2015	1.6.5	• Added support for resizable BARs. • Added support for PTM. • Updated TL_Bypass signals. • Updated tl_brsw_in[1] description. • Updated descriptions of LTR and OBFF messages. • Updated section on Transmit buffer layout.
December 2014	1.6.4	• Added support for extended simulation mode. • Added support for ACS. • Added support for PIPE gapped mode. • Updated tl_tx_proterr0 description. • Added k_equpreset16 variable. • Added section on equalization problems.
October 2014	1.6.3	• Added support for TX error management. • Added support for RX stream watchdog. • Removed support for multiple virtual channels.
September 2014	1.6.2	• Added support for ID-based ordering. • Added information about external interrupt mode. • Updated CDC description and added appendix about CDC implementation. • Added support for Retimer detection. • Modified description of reset strategies and clock and reset signals. • Added support for TLP Prefixes. • Added support for TLP Processing Hints.
June 2014	1.6.1	• Added chapter on Data Protection.
April 2014	1.6.0	• Corrected k_gen[29:31] description.
March 2014	1.6.0	• Added information about Electrical Idle Usage. • Added pl_ltssm_enable signal. • Added information about ECRC generation mode. • Modified local FS/LF/Preset-to-coefficient values description.
February 2014	1.5.9	• Added support for PIPE 3.0 rev 9/4.0/4.2. • Updated power management descriptions. • Added description of simulation mode and updated test_out description.
November 2013	1.5.7	• Added support for RX Stream Mode. • Updated description of Rx/Tx Latency and Cut-Through. • Added support for PIPE4-PIE8 module.
October 2013	1.5.5	• Added support for SRIS functionality. • Updated tl_tx_err0 description. • Updated TL Bypass signal description.
August 2013	1.5.4	• Updated K_PIPE[19], TL_REPORT_STATUS[14], and TL_REPORT_LATENCY descriptions. • Amended description of Per-vector masking.
June 2013	1.5.3	• Updated test_in description, and corrected G_NPBUF_SIZE description.
May 2013	1.5.2	• Added note to test_in description, updated Transaction Clock Frequency description, and corrected MSI Capability Structure byte offset.

Date	Version Number	Change
April 2013	1.5.2	• The following signals have been updated: K_PEXCONF, TL_FLR_REQX, TL_FLR_ACKX, TL_INT_VFNUMX, TL_CFG_PADDRX, TL_CFG_EXPADDRX, TL_INT_REQ0, and TL_RX_BARDEC. • The following signals have been added: TL_PM_CLKCONTROL, PL_CLKREQ_IN, TL_PM_CLKSTATUS, and TL_REPORT_LATENCY. • The behavior of several TL_REPORT_EVENT bits has changed. • An appendix has been added to summarize the interface modification.
March 2013	1.5.1	• Corrected minor documentation errors.
December 2012	1.5.0	• Added support for Latency Tolerance Reporting. • Added support for Optimized Buffer Flash Fill (OBFF). • Added support for L1 PM Substates with CLKREQ.
October 2012	1.4.3	• Corrected documentation errors.
September 2012	1.4.2	• Corrected RAM interface Read waveform.
August 2012	1.4.2	• Added support for Equalization interface. • Added information on sizing Tx and Rx buffers.
June 2012	1.4.1	• Added support for Transaction Layer Bypass mode.
June 2012	1.4.0	• Added support for Virtual Functions. • Added support for Multicast
March 2012	1.4.0	• Added support for SR-IOV. • Added information about reset strategies. • Added information about error and message handling.
February 2012	1.3.0	• Updated k_pipe[17].
December 2011	1.3.0	• Added support for PIE-8 interface. • Added description of RAM interface.
October 2011	1.2	• Updated k_pipe signal and added appendix on differences with XpressRich2.
September 2011	1.2	• Updated Core parameters and signals.
July 2011	1.1	• Corrected minor documentation errors.
June 2011	1.1	• Updated Switch/Bridge interface and associated support. • Added information about cut-through handling for Switches. • Updated Core parameters and signals.
March 2011	1.0	• First release

Proprietary Notice

Table of Contents

List of Tables

List of Figures

Preface

About this document

This document has been written for design managers, system engineers, and designers of ASICs and FPGAs who are evaluating or using the PLDA XpressRICH3 Core.

Additional Reading

This section lists additional resources from PLDA and third-parties.

PLDA periodically updates its documentation. Please contact PLDA Technical Support or check the Web site at http://www.plda.com for current versions.

PLDA Publications

Please refer to the following documents for further information:

- *XpressRICH3 Getting Started*: The *Getting Started* guide provides information to enable designers to integrate PLDA PCI Express Expert Core into their design flow as quickly as possible (installing, customizing, integrating, and simulating the Core).
- *XpressRICH3 Build History*: The *Build History* lists changes made in each version and build of the Core.
- *PLDA Bus Functional Model 3 Reference Manual*: This document describes PLDA's PCI Express BFM (Testbench).

Other Publications

Please refer to the following documents for information on specification standards:

- *PCI Express® Base Specification Revision 4.0 Version 1.0*
- *PHY Interface for the PCI Express, SATA and USB 3.1 Architectures, Version 4.4.1*

Feedback and Contact Information

Feedback about this Document

PLDA welcomes comments and suggestions about this documentation. Please contact PLDA Technical Support and provide the following information:

- the title of the document
- the page number to which your comments refer
- a description of your comments

Contact information

Corporate Headquarters

PLDA
805, avenue J.R.G.G. de la Lauzière
13290 Aix-en-Provence
France

Tel: USA +1 408 273 4528 - International +33 442 393 600
Fax: +33 442 394 902

Sales

For sales questions, please contact sales@plda.com.

Technical Support

For technical support questions, please contact PLDA Support at https://www.plda.com/support using the Support Center if you have a PLDA online account.

If you don't have a PLDA account, contact https://www.plda.com/user/register.

Chapter 1 PCI Express XpressRICH3 Core Features

PLDA's XpressRICH3 provides an integrated and customizable solution for designing Endpoint or Rootport components for both ASICs and FPGAs.

The Core is compliant with *PCI Express Base Specification Revision 4.0 v1.0* and implements all required and most optional features of this specification, as well as some enhanced features, to provide a highly customizable solution. The Core is also compliant with the Intel® PIPE interface, allowing integration with PIPE-compliant PHY layers.

The following table describes the features of the XpressRICH3 Core:

General	• x1, x2, x4, x8, x16 PCI Express Core • 256 or 512-bit application interface datapath • Supports link rate of 2.5, 5.0, and 8.0 GT/s per lane • Suitable for Rootport, Endpoint, Bridge, Switch and dual mode/shared silicon • P*CI Express Base Specification Revision 4.0 v1.0* compliant • *PHY Interface for PCI Express (PIPE) 4.4.1* compliant • 1 Virtual Channel (VC) • Up to 8 Physical Functions (see G_NUM_FUNC description in Section 4.1) • 8-bit, 16-bit, and 32-bit PIPE interface • Receive and Replay buffer size configurable • Receive/Transmit User Application Interface • Up to six BARs; resizable • Advanced Error Reporting (AER) support • ECRC generation and check support • Test port functionality • User clock - integrated Clock Domain Crossing to support user-selected frequency at the Application layer • Lane reversal • Crosslink • Multicast • Alternate Routing ID Interpretation (ARI) • ID-based Ordering (IDO) • Retimer (extension device) presence detection • TLP Processing Hints (TPH) • Access Control Services (ACS) • Downstream Port Containment (RP Extensions for DPC are not supported) • Precision Time Measurement (PTM) • Autonomous link speed/width change support
Customization	• Easy customization with the IP Wizard • User backdoor access to the PCIe Configuration Space • Unused features not implemented in silicon • Transaction Layer can be bypassed
Virtualization	• SR-IOV support with up to 512 virtual functions • Address Translation Service, including Page Request interface • Process Address Space ID (PASID)
Data transfer	• Up to 4KB data payload transfer • All Memory, I/O, Configuration, and Message transactions • Up to 4 End-End TLP prefixes • Vendor-Defined Messages (enables MCTP over PCIe)

Configuration	• Implements Type 0 Configuration space for Endpoint designs • Implements Type 1 Configuration space for Rootport, Switch, and Bridge designs • Up to 6 BARs plus expansion ROM can be implemented for Endpoints • All I/O and memory windows implemented for Rootport
Power Management and Interrupt	• All Power State and associated logic implemented • Native Active State Power Management L0s and L1 state support • Power Management Event (PME message) and Beacon (Wake-Up) support • MSI (up to 32) and INT message support • MSI-X Capability Support • Latency Tolerance Reporting (LTR) • Optimized Buffer Flush Fill (OBFF) • L1 PM Substates with CLKREQ

1.1 Supported Topologies

The following picture shows the different topologies supported by the Core:

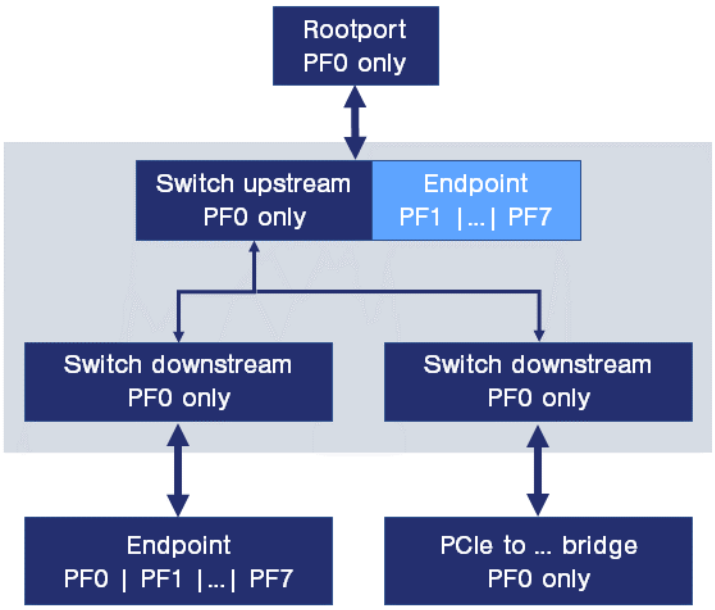


Figure 1: Core Topologies

- Rootport: when the Core is configured as a Rootport, then only Physical Function #0 may be implemented.
- Endpoint: both Legacy and Endpoint modes are supported, and up to 8 Physical Functions can be implemented.
- Switch upstream port: when the Core is configured in this mode, Physical Function #0 is a switch upstream port function; Physical Functions #1 - 7 can be optionally implemented as Native Endpoint functions.
- Switch downstream port: when the Core is configured in this mode, then only Physical Function #0 may be implemented.
- PCI Express to PCI/PCI-X bridge: when the Core is configured in this mode, then only Physical Function #0 may be implemented.

In addition, a single core instance can be “dual-role” meaning that the port type can be changed using k_.. signals when the device is under reset. This makes it possible, for example, to build a single device that can be either Rootport or Endpoint depending on the programming at power-up:

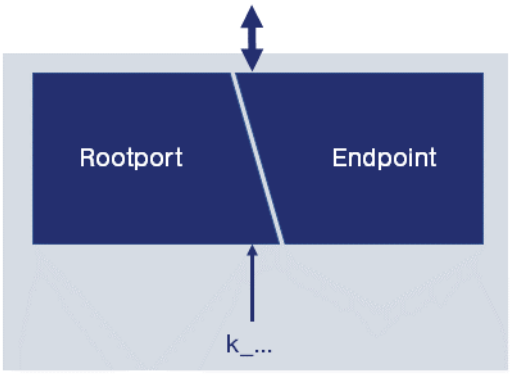


Figure 2: Dual-Role Topology

Chapter 2 XpressRICH3 Core Architecture

The following figure shows an overview of the XpressRICH3 Core architecture:

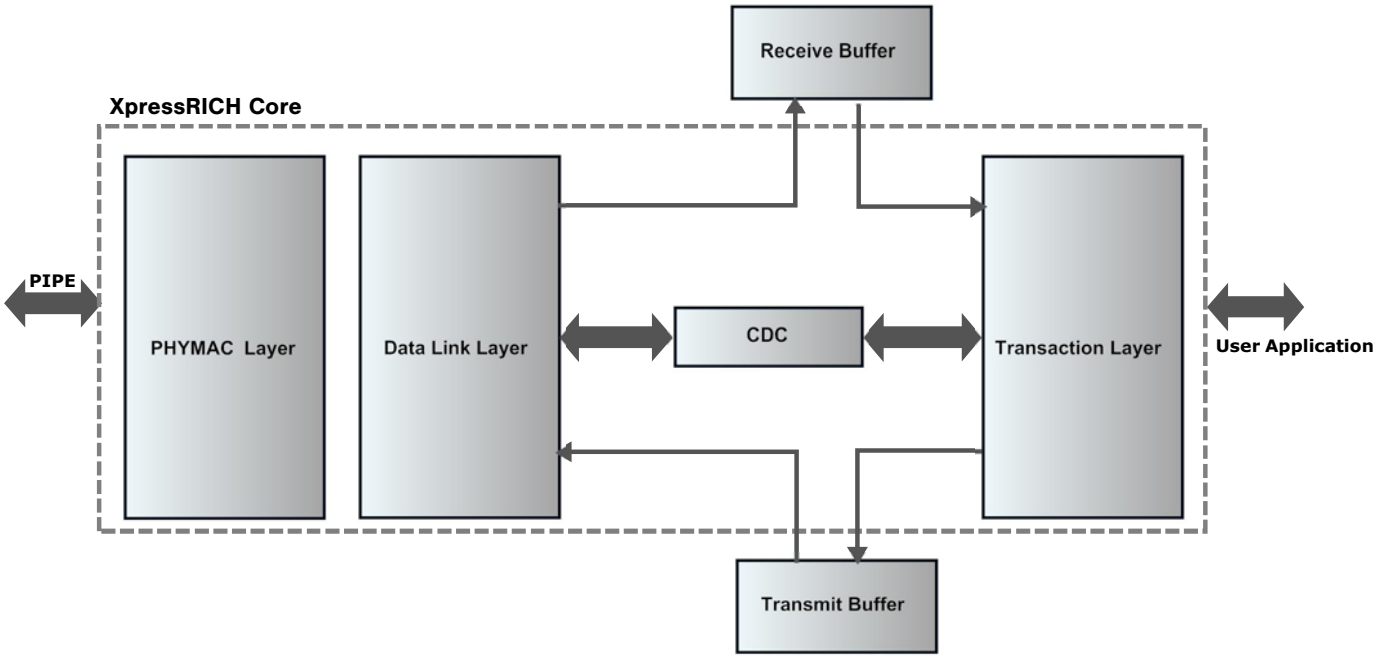


Figure 3: XpressRICH3 Core Architecture

- The Physical Media Access Controller Layer, or **PHYMAC Layer**, manages the initialization of the PCI Express link, as well as the physical events that occur during normal operation. This layer functions at the PIPE clock frequency.
- The **Data Link Layer** manages communication at the link level between the two connected PCIe components in order to ensure the correct transmission of packets. Its main roles are: generation and check of the CRC and Sequence Number of the TLP; initialization and update of flow control credits; generation of ACK/NAK; and the management of the Receive and Retry Buffers.
- The **Clock Domain Crossing** (CDC) handles the transition between the PIPE PCI Express clock domain and the User Application and Core clock domains.
- The **Transaction Layer** manages the generation of TLPs from both the Application Layer and the Configuration Space on the transmit side and checks for flow control credits before transmitting the TLPs to the Data Link Layer.
  On the receive side, the transaction layer extracts received TLPs from the Receive buffer, checks their format and type, and then routes them to the Configuration Space or to the Application Layer. It also manages the calculation of credits for the Receive buffer.
  Even though it is not part of the Transaction Layer function, this layer also includes the Configuration Space, which can receive and generate dedicated TLP.

2.1 PHYMAC Layer

The PHYMAC Layer consists of:

- The LTSSM block, which is the central intelligence of the physical layer.
- On the receive side, one Deskew FIFO and eight similar RxLane blocks.
- On the transmit side, one TxAlign and LTSTX block and eight TxLane blocks.

2.1.1 LTSSM Block

The LTSSM (Link Training and Status State Machine) block

- Checks and memorizes what is received on each lane.
- Determines what should be transmitted on each lane.
- Transitions from one state to another.

On the receive path, the LTSSM block manages the received PLP (Physical Layer Packet) reported by each RxLane block, and is also responsible for enabling multi-lane deskew.

On the transmit side, the LTSSM requests the LTSTX sub-block to generate specific PLP, transmit Electrical Idle, perform receiver detect operations, and generate the compliance pattern. The LTSTX acknowledges these requests, and returns the receiver detection result to the LTSSM.

The LTSSM waits until the transmission of the last packet is completed before moving to recovery or low power states. It can be directed by a higher layer to move to the recovery, disable, hot reset or low power states using a simple request/acknowledge protocol.

The LTSSM block is also responsible for reporting the physical layer status to the higher layer.

2.1.2 RxLane Block

The PHYMAC Layer implements an identical RxLane block for each lane.

The RxLane block descrambles and decodes the received PLP and reports the type of OS (Ordered Set) received, its parameters and the number of consecutive OS to the LTSSM.

The decoded flow is transmitted to the Deskew module as an 18-bit parallel interface (16 bits for data and 2 bits for control) at 125, 250, or 500 MHz, depending on the negotiated link speed.

2.1.3 Deskew Block

In a multi-lane device, and when enabled by the LTSSM block, the Deskew block performs lane-lane deskew. This is achieved using independent FIFOs that are implemented for each lane using registers.

The table below shows FIFO size (per-lane) and maximum allowable skew:

PIPE Interface Max Width (bits)	64		32		16	
G_DESKEW_EXT	0	1	0	1	0	1
FIFO width (symbols)	8	8	4	4	2	2
FIFO depth	4	8	4	8	8	16
Max symbols 2.5/5.0 GT	8	16	16	32	16	32
Max symbols 8.0 GT	32	64	16	32	16	32
Max skew at 2.5 GT PIPE 8-bit (ns)			60	60	60	60
Max skew at 2.5 GT PIPE 16-bit (ns)	24	56	56	56	56	56
Max skew at 2.5 GT PIPE 32-bit (ns)			48	48		
Max skew at 5.0 GT PIPE 8-bit (ns)			30	30	30	30
Max skew at 5.0 GT PIPE 16-bit (ns)	12	28	28	28	28	28
Max skew at 5.0 GT PIPE 32-bit (ns)			24	24		
Max skew at 8.0 GT PIPE 16-bit (ns)			14	30	14	30
Max skew at 8.0 GT PIPE 32-bit (ns)	28	60	12	28		
FIFO memory size (bits)	320	640	160	320	160	320

Table 1: Deskew FIFO size and lane-lane deskewing capability

The Deskew block then sorts the packets according to type:

- PLP (Physical Layer Packets) are discarded.
- DLLP (Data Link Layer Packets) are forwarded to the DLLP Decode block.
- TLP (Transaction Layer Packets) are forwarded to the TLP Decode block.

The Deskew block is also responsible for detecting and reporting framing errors to the LTSSM and Configuration blocks.

2.1.4 TxAlign and LTSTX Blocks

The TxAlign and LTSTX blocks align and arbitrate between:

- PLPs (Physical Layer Packets) generated when an LTSSM request is received.
- DLLPs (Data Link Layer Packets) received from the DLLP Encode block.
- TLPs (Transaction Layer Packets) received from the TxSeqNum and CRC Generation block.

Depending on the number of initialized lanes, more than one DLLP/TLP can be sent in the same symbol time.

When the TxAlign and LTSTX block receives an LTSSM block request, it:

- Generates the PLP (Physical Layer Packet) and acknowledges the request.
- Performs the receiver detection process, and reports the result of the receiver detection to the LTSSM.

2.1.5 TxLane Block

The PHYMAC Layer implements an identical TxLane block for each lane; which receives PLPs from the LTSTX block and transmits them.

2.2 Data Link Layer

The Data Link Layer consists of:

- The DLCMSM block, which initializes the Data Link Layer and Flow Control credits.
- On the receive side; a DLLP Decode block, and an RxSeqNum and CRC Checking block.
- On the transmit side; a DLLP Encode block and a TxSeqNum and CRC Generation block.

2.2.1 DLCMSM Block

The DLCMSM block implements the Data Link Control and Management State Machine. This state machine initializes the Data Link Layer to “dl_up” status, after having initialized Flow Control credits for Virtual Channel 0.

In order to do this, the DLCMSM:

- Is informed by the DLLP Decode block of the reception of Initialization Flow Control packets.
- Orders the DLLP Encode block to transmit Initialization Flow Control packets.

2.2.2 DLLP Decode Block

DLLPs (Data Link Layer Packets) are received from the Deskew block.

The DLLP Decode block checks and decodes the following types of DLLP:

- Initialization Flow Control DLLP: the DLLP Decode block informs the DLCMSM block of their reception and content.
- Ack/Nak DLLP: the DLLP Decode block informs the TxSeqNum and CRC Generation of their reception and content so that the TLP contained in the Retry Buffer can be discarded or replayed.
- Update Flow Control DLLP: the DLLP Decode block reports their content to the Tx Flow Control Credit block.
- Power Management DLLP: the DLLP Decode block informs the Configuration block of their reception and content.

2.2.3 RXSEQNUM & CRC Checking Block

TLPs are received from the Deskew FIFO, and the RxSeqNum and CRC Checking block then:

- Checks both the LCRC and the Sequence Number in order to validate the TLP write in the Receive Buffer (a Data Link Layer requirement).
- Checks if the field length of the received TLP corresponds to the effective length of the TLP (a Transaction Layer requirement).
- Reports detected errors to the Configuration module.
- Reports the Sequence Number and the result of the received TLP checking to the DLLP Encode block to enable Ack/Nak DLLP generation.
- Writes the received TLP in the Receive Buffer, without SeqNum and CRC fields.
- If TLP checking is successful, the RxSeqNum and CRC Checking block informs the Rx TLP Checking block that a new TLP is available in the Received Buffer.

2.2.4 DLLP Encode Block

The DLLP Encode block encodes the following different types of DLLP (Data Link Layer Packets):

- Initialization Flow Control DLLPs; when a request is received from the DLCMSM block.
- High Priority Ack/Nak DLLPs; when the RxSeqNum and CRC Checking block reports that an incorrect TLP has been received, or when no Ack DLLP has been sent during a predefined time slot.
- High Priority Update Flow Control DLLPs; based on the available Flow Control Credits reported by the Rx Flow Control Credit block.
- Power Management DLLPs; when a request is received from the Configuration block.
- Low Priority Update Flow Control DLLPs; based on the available Flow Control Credits reported by the Rx Flow Control Credit block.

- Low Priority Ack/Nak DLLPs; based on the information reported by the RxSeqNum and CRC Checking block.

The encoded DLLPs are transmitted to the TxAlign and LTSTX block.

If one or more DLLP is not acknowledged by the TxAlign and LTSTX block during a symbol time, the DLLP Encode block can change the DLLP content and its priority level to ensure that it always transmits the DLLP with the highest priority to the TxAlign and LTSTX block.

2.2.5 TXSEQNUM & CRC Generation Block

The TxSeqNum and CRC Generation block is notified that TLPs are available in the Retry Buffer by the write address pointer of the last TLP written by the Tx TLP Arbitrating block.

The TxSeqNum and CRC Generation block:

- Extracts the TLP to transmit from the Retry Buffer, from its internal read address pointer to the write address pointer reported by the Tx TLP Arbitrating block.
- Adds a Sequence Number and Generate CRC.
- Nullifies the TLP if a parity error is detected. To do this, the CRC value is inversed and the Physical Layer is notified that this TLP must be nullified.
- Writes the Retry Buffer read address pointer associated with the generated Sequence Number into a small buffer.
- Discards TLPs from the Retry Buffer when an Ack DLLP is reported by the DLLP Decode block. To do this, the TxSeqNum and CRC Generation block notifies the Tx TLP Arbitrating block that the Retry Buffer is now free up to the read address pointer associated with the received Sequence Number.
- Replays TLPs in the Retry Buffer when a Nak DLLP is reported by the DLLP Decode block. To do this, the internal read address pointer is loaded with the read address pointer associated with the received Sequence Number, and TLP extraction is restarted from this point. Previous TLPs are discarded.

2.3 Clock Domain Crossing Layer

A Clock Domain Crossing Layer is required to handle different clock domain transitions.

The Clock Domain Crossing Layer consists of:

- The Receive Buffer; which stores the received TLPs to forward to the user back-end interface.
- The Retry Buffer, which stores the TLPs to transmit on the PCI Express link.
- A set of registers, which resynchronize information between the PIPE PCI Express clock and the PCI Express Core clock domains.

2.3.1 Receive Buffer

The size of the Receive Buffer depends on the allocated credits for Posted/Non-Posted/Completion Header and Data, and whether ECRC is supported.

Typically, its size is computed as given below:

- RxBuffer size (Bytes) = (PH + NPH + CPLH + PD + NPD + CPLD) x 16 Bytes

ECC (Error Code Correction) is used when reading from the buffer. If an uncorrectable error is detected by the ECC mechanism, this error is reported to the Rx TLP Checking block and forwarded to the application back-end interface.

2.3.2 Clock Domain Crossing Block

The Clock Domain Crossing (CDC) block is implemented with registers.

Depending on the type of signal, different resynchronization logic is implemented:

- the 'Slow-bit' method is used for nearly static bits: it consists of just two resynchronization flip-flops.
- the 'Vector' method is used for multiple-bit signals: data itself is not resynchronized but a request/acknowledgment mechanism ensures it is sampled in the destination clock domain only when it is safe to do so.
- the 'Pulse' method is used for single pulse bits
- the 'Address' method is used for address pointers: it uses a FIFO-like mechanism to resynchronize pointer values with minimal latency.

For more information, see Appendix D: CDC Implementation Considerations.

2.4 Transaction Layer

The Transaction Layer consists of:

- An Rx TLP checking block and an Rx Flow Control Credit block on the receive side.
- A Tx TLP Arbitrating block and a Tx Flow Control Credit block on the transmit side.
- The Configuration Space, which can receive and generate dedicated TLPs (even though the Configuration Space is not one of the Transaction Layer functions).

2.4.1 Rx TLP Checking block

The Rx TLP Checking block is notified by the RxSeqNum and CRC Checking block that new TLPs are available in the Receive Buffer.

The Rx TLP Checking block then:

- Extracts TLPs to process and realign.
- Routes the TLP to the Configuration block, if required (for unsupported requests, configuration requests, or messages, for example) or to the User Application Back-end Interface.
- Notifies the Rx Flow Control Credit block that the TLP has been processed.

2.4.2 Rx Flow Control Credit Block

The Rx Flow Control Credit block computes the remaining flow control credits associated with the Core Receive Buffer.

When a TLP is fully extracted from the Receive Buffer (that is, forwarded to the Configuration block, discarded, or transmitted to the back-end interface), the flow control credits are updated according to the type and length of the TLP. The available flow control credits are reported to the DLLP Encode block, to generate appropriate Update Flow Control DLLPs.

2.4.3 Tx TLP Arbitrating block

The Tx TLP Arbitrating block:

- Performs arbitration between the TLP to process, if any (for User Application TLPs, Configuration TLPs, unsupported requests, or messages, for example).
- Checks that enough flow control credits are available to process the TLP, based on the information reported by the Tx Flow Control Credit block.
- Checks for Transmit Buffer availability before writing the TLP to the Transmit Buffer.
- Notifies the Tx Flow Control Credit block that TLP credits have been consumed.

2.4.4 Tx Flow Control Credit Block

The Tx Flow Control Credit block computes the available flow control credits associated with the PCI Express Core, based on the Update Flow Control DLLP content reported by the DLLP Decode block, and the credits consumed when a TLP is processed by the Tx TLP Arbitrating block.

2.4.5 Configuration Block

The Configuration block:

- Implements the Configuration Space registers.
- Checks for malformed TLP in coordination with the Rx TLP Checking block, and defines whether the TLP is routed to the Configuration Space or to the User Application Back-end Interface.
- Handles all errors detected in the Core and, depending on the error classification, specifies if they must be logged in the Configuration Space status registers and/or if they must be reported (serr_out or MSI if AER is enabled).
- Receives TLPs (Configuration Read/Write, Messages, and Unsupported Requests), and updates the Configuration Space registers accordingly.
- Generates TLPs (MSI, Completion, and INT/PM/Error/Hot-Plug Messages)

2.4.6 Transmit Buffer

The Transmit Buffer consists of simple Dual-Port block RAM. It is used to store TLPs in the PCI Express Core, even if there are not enough flow control credits available to send these TLPs to the PCI Express link. See Section 2.5.1 for more information.

2.5 Memory Specifications

The following table describes each type of memory in the XpressRICH3 Core.

Memory	Bit Width	Depth (words)	No. of instances	RAM Type
Receive Buffer	(33+G_DATA_PROT)* G_TL_DATAPATH	• 2^(G_RXBUF_SIZE-5) for 256-bit TL datapath • 2^(G_RXBUF_SIZE-6) for 512-bit TL datapath	1	• Dual-port RAM (1 write port and 1 read port) • Single-clock when CDC is not implemented, dual-clock when CDC is implemented
Transmit Buffer	(33+G_DATA_PROT)* G_TL_DATAPATH	• 2^(G_TXBUF_SIZE-5) for 256-bit TL datapath • 2^(G_TXBUF_SIZE-6) for 512-bit TL datapath,	1	• Dual-port RAM (1 write port and 1 read port) • Single-clock when CDC is not implemented, dual-clock when CDC is implemented
Deskew FIFO	• 40 bits in 32-bit PIPE • 20 bits in 16-bit PIPE	• 4 in 32-bit PIPE • 8 in 16-bit PIPE	• 1 per lane in x2, x4, x8, and x16 • 0 in x1	Register file(s)
Replay FIFO	G_TXBUFSIZE-5	2^G_NUM_TLPRPL	1	Register file
Non-Posted Buffer (optional)	(33+G_DATA_PROT)xG_NPBUF_WIDTH+26	2^G_NPBUF_SIZE	1	Register file or single-clock dual-port RAM (1 write port and 1 read port)

Table 2: Memory specifications

2.5.1 Sizing the Transmit Buffer

The Transmit buffer is used to store TLPs until they are sent and acknowledged by the link partner.

It's important to specify an appropriate size for the Transmit buffer using the G_TXBUF_SIZE core constant (see Section 4.1) in order to achieve expected performances and to keep memory usage as low as possible.

The maximum size in bytes that a transmitted TLP (header + payload) can have is: 2^G_RXBUF_SIZE-160.

So, for example with a 4KB buffer (G_RXBUF_SIZE=12), the maximum TLP size (header + payload) is 3936 bytes. However, nearly filling TX buffer with a single TLP would lead to poor performances.

The appropriate size for your Transmit buffer will depend on the expected throughput for your application and what kind of packets the application is designed to transfer. You should therefore take into account:

- The number of packets the application will transfer. The Transmit buffer can contain up to 2^(G_TXBUF_SIZE-5) packets. For high throughput, the buffer must be able to store a lot of packets, because packets are only removed from the Transmit buffer when the link partner acknowledges them.
- The size of the transferred packets: an application that transfers TLPs of the maximum payload size will usually need a lot of buffer space.

As a general guideline, we recommend using the following settings:

Device Transmit Profile	Transmit Buffer Size
Low throughput	2 x maximum payload size
Medium throughput	4 x maximum payload size
High throughput	8 or 16 x maximum payload size

Table 3: Recommended Transmit Buffer sizes

2.5.2 Sizing the Receive Buffer

The Receive buffer is used to store TLPs received from the PCIe link before they are treated by the Core and the application. The size of the Receive buffer size, which is set using the G_RXBUF_SIZE core constant (see Section 4.1) has a direct impact on device performance.

At the very minimum, the buffer must store at least one TLP of maximum size for each TLP type (Posted, Non-posted, Completion)

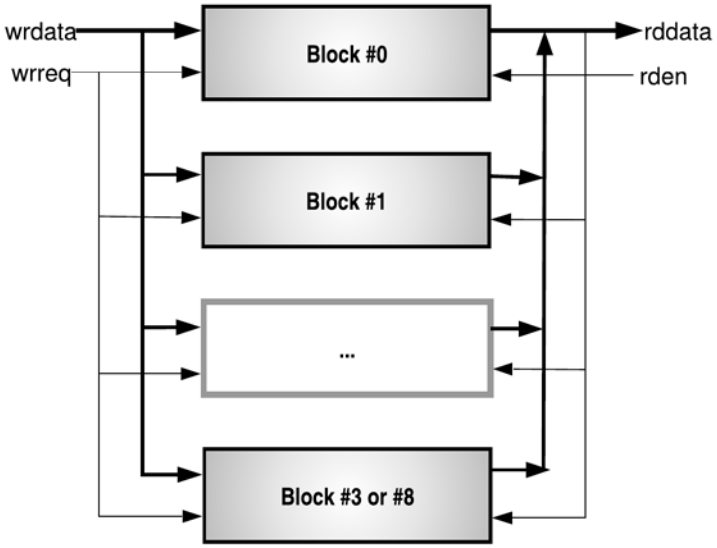
As a general guideline, we recommend using the following settings:

Device Transmit Profile	Receive Buffer Size
Low throughput	4 x maximum payload size
Medium throughput	8 x maximum payload size
High throughput	16 or 32 x maximum payload size

Table 4: Recommended Receive Buffer sizes

2.6 Receive and Transmit Buffer Interface

The Receive and Transmit buffers are dual-clock DPRAM with one read port and one write port. Each of these buffers are implemented as four (when TL datapath is 256-bit) or eight (when TL datapath is 512-bit) independent memory blocks connected in parallel that can be read/written independently by the Core.



The following table shows how data is split between the memory blocks, depending on the data protection settings:

G_DATA_PROT	Block #0	...	Block #n
0	Bits 65:0	...	n x 66+65 : n x 66
1	Bits 67:0	...	n x 68+67 : n x 68
4	Bits 73:0	...	n x 74+73 : n x 74

Table 5: Data protection settings and memory

The following table describes read and write interface signals:

Signal	I/O	Width	Description
wrclk	in		**Write Clock**
wrreq	in	4 or 8	**Write Request**: Data must be written to memory within two clock cycles following the assertion of this signal. Each control bit writes to one of the memory blocks. This signal is 4 bits if the write side of the buffer is connected to a 256-bit datapath, or 8 bits if connected to a 512-bit datapath.
wraddr	in	ADDR_WIDTH	**Write Address**
wrdata	in	(33+G_DATA_PROT)x4 or x8	**Write Data**
rdclk	in	1	**Read Clock**
rden	in	4 or 8	**Read Enable:** Each control bit writes to one of the memory blocks. This signal is 4 bits if the write side of the buffer is connected to a 256-bit datapath, or 8 bits if connected to a 512-bit datapath.

Table 6: RAM interface signals

Signal	I/O	Width	Description
rdaddr	in	ADDR_WIDTH	**Read Address**
rddata	out	(33+G_DATA_PROT)x4 or x8	**Read Data**: Memory can return any data when rdaddr equals wraddr; no specific logic is necessary to handle this case.
rdderr	in	4 or 8	**Read Data Error**: This optional signal is used if the memory features ECC protection in order to indicate that an uncorrectable error was detected when reading data from a memory block. It must be tied to 0s if memory has no data error checking capability. This signal is 4 bits if the read side of the buffer is connected to a 256-bit datapath, or 8 bits if connected to a 512-bit datapath.

Table 6: RAM interface signals

Note: ADDR_WIDTH depends on buffer size settings:

- For the transmit buffer, it is G_TXBUF_SIZE-5
- For the receive buffer, it is G_RXBUF_SIZE-5

2.6.1 RAM Read/Write Latency

Latency for the Receive and Transmit buffers is defined by the G_RXBUF_LATENCY and G_TXBUF_LATENCY Core constants (see Section 4.1). These constants define read/write latency independently for each buffer.

Adding latency cycles on read and/or write operations may relax timing constraints on the memory module, but creates additional latency on receive/transmit data paths.

The following waveform illustrates a write operation with one cycle of latency, followed by a read with one latency cycle (G_xxBUF_LATENCY=11):

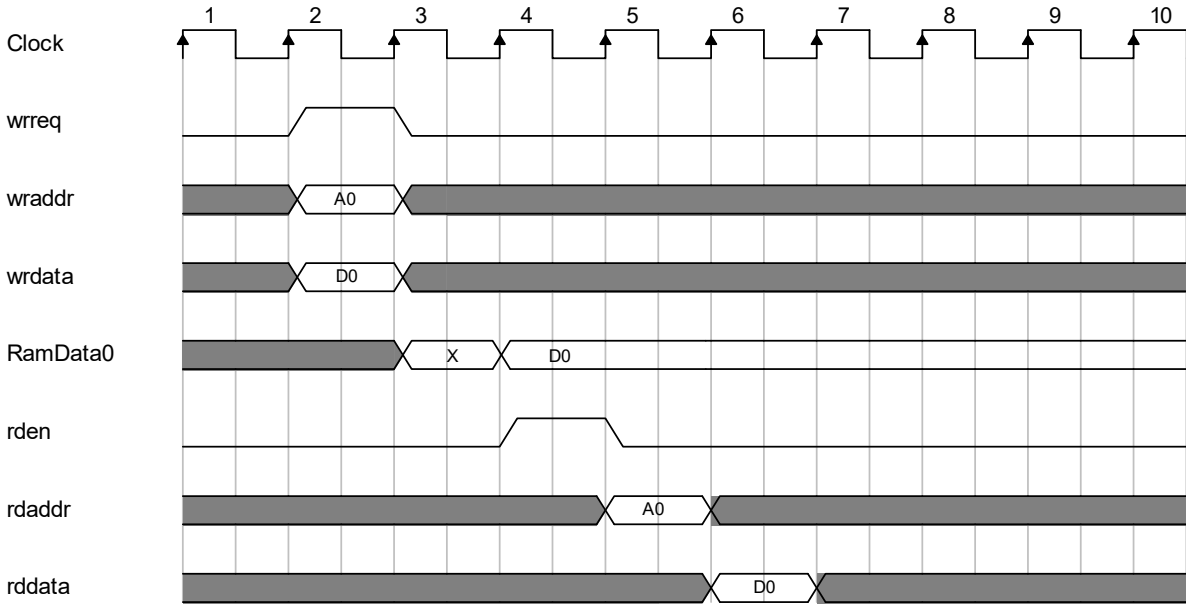


Figure 4: 1 cycle latency write, 1 cycle latency read

The following waveform illustrates a write operation with three cycles of latency, followed by a read with one latency cycle(G_xxBUF_LATENCY=13):

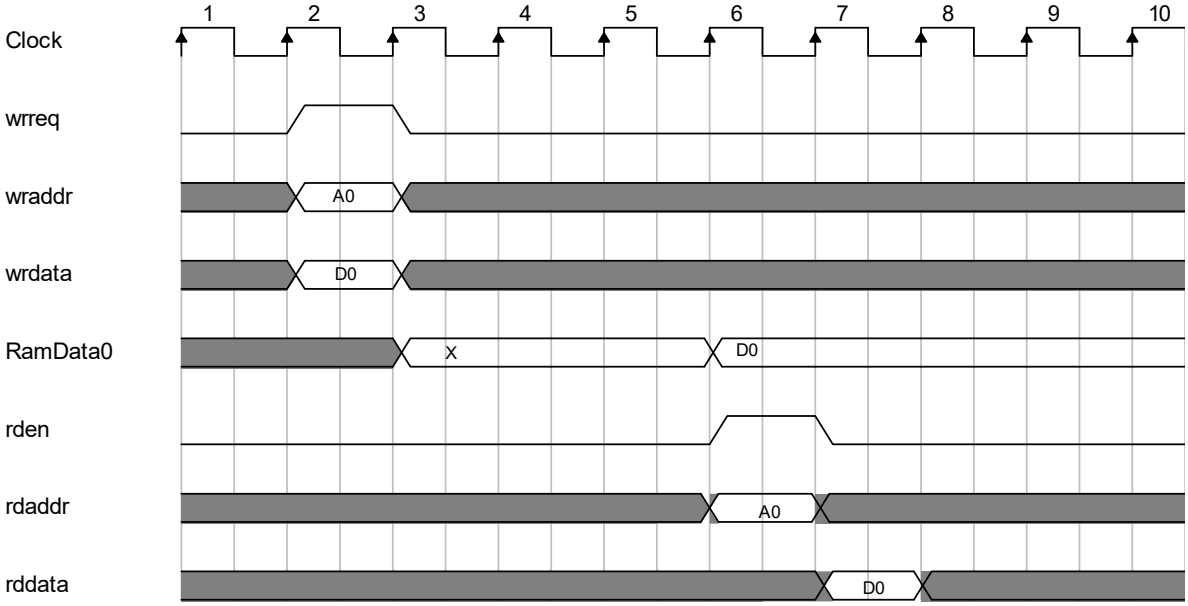


Figure 5: 3 cycle latency write, 1 cycle latency read

The following waveform illustrates a write operation with three cycles of latency, followed by a read with two latency cycles (G_xxBUF_LATENCY=23):

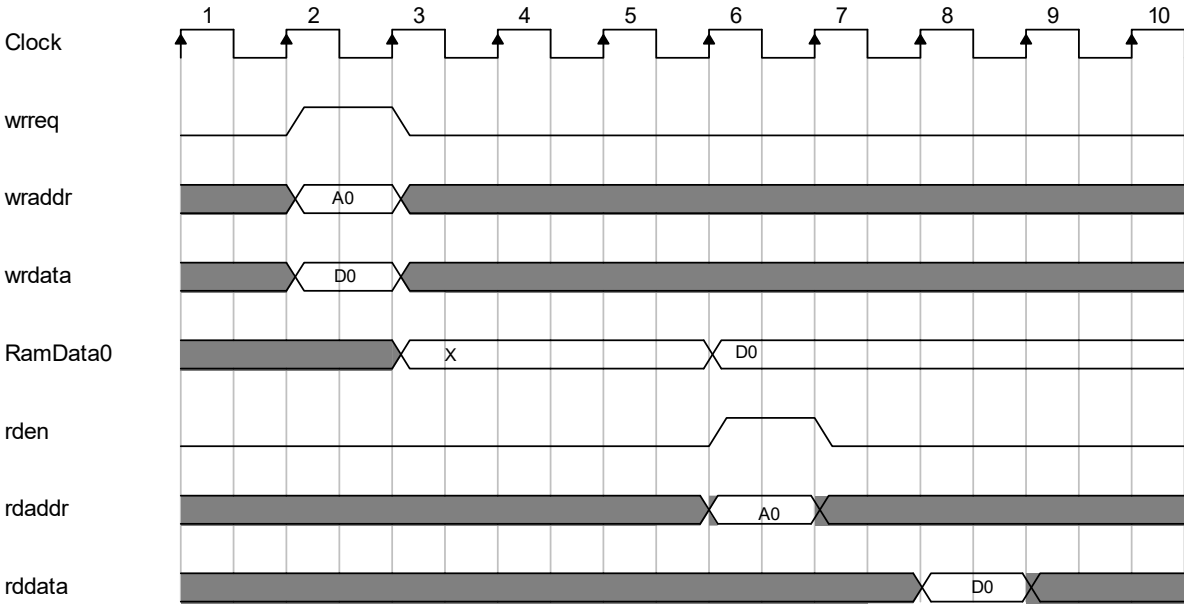


Figure 6: 3 cycle latency write, 2 cycle latency read

The following waveform illustrates a read operation with one cycle of latency with errors detected on bits [255:128] of data at address 2:

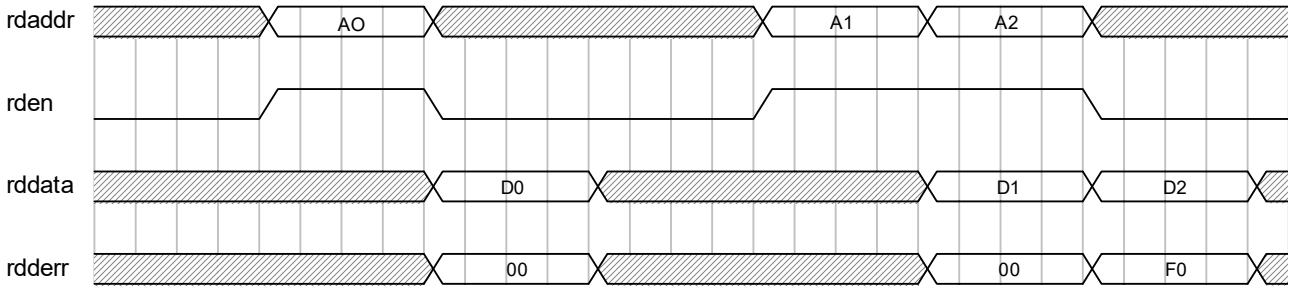


Figure 7: 1 cycle latency reads with data error

2.7 Transaction Layer Bypass

You can set Transaction Layer Bypass mode using the XpressRICH3 TXRX Wizard. This mode enables you to implement and use the XpressRICH3 Core without its transaction layer. This mode is suitable for designers who do not need a transaction layer or who want to implement a custom layer.

The following figure shows the architecture of the XpressRICH3 Core when Transaction Layer Bypass mode is implemented:

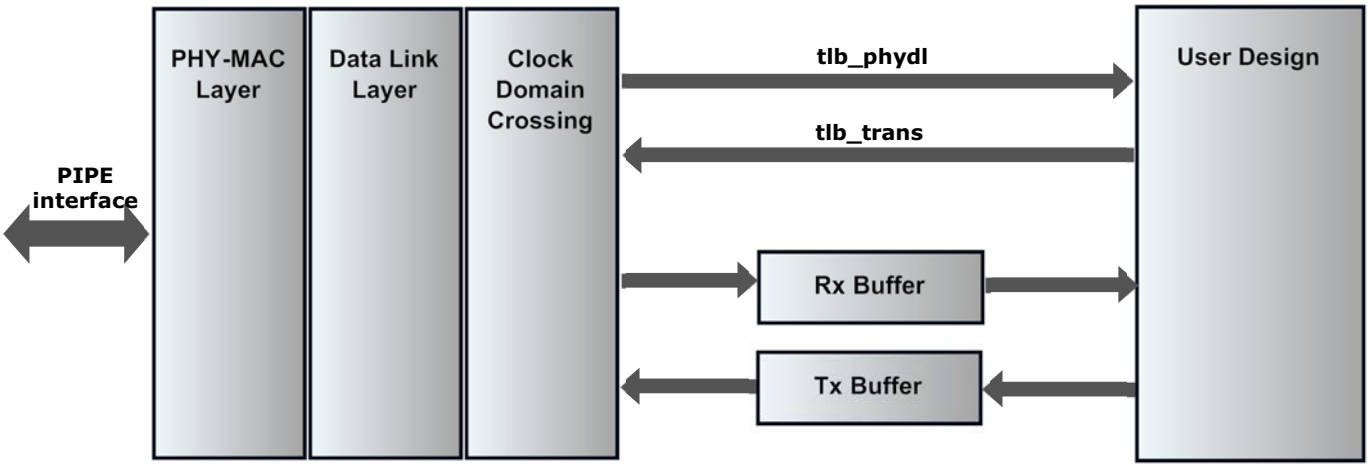


Figure 8: XpressRICH3 Core Architecture in Transaction Layer Bypass mode

In this mode, the transaction layer signals (tl_xxx, see Section 3.2) are no longer available and communication between the Core and the user design is through the tlb_trans and tlb_phydl signals, as well as the Receive and Transmit buffers.

2.7.1 Transaction Layer Bypass Signals

2.7.1.1 tlb_trans

The TLB_TRANS[370:0] input signal contains information from the transaction layer to the PHYMAC and Data Link layers. It is synchronous to the TL_CLK clock. Indexes for this signal are defined in the include file pcie3_sharedbus_h.v, and are described in the following table:

Index	Size	Description
SHB_TRANS_TLPAVAIL_ADDR	16	**Transmit buffer TLP available address**: This indicates that memory lines before this address contain valid TLPs that can be sent. When G_TL_DATAPATH=8: • Bits 15:1: memory address (higher bits are filled with 0 if unused) • Bit 0: group number When G_TL_DATAPATH=16: • Bits 15:2: memory address (higher bits are filled with 0 if unused) • Bits 1:0: group number See Section 2.7.3: Transmit Buffer Layout for more information.

Table 7: tlb_trans signal

Index	Size	Description
SHB_TRANS_RXBUF_CUR_PTR0	16	**Receive buffer read pointer position**: This indicates that blocks of data before this address have been read by the transaction layer. This information is used to detect overflow and check when the buffer is empty. When G_TL_DATAPATH=8: • Bits 15:2: memory address (higher bits are filled with 0 if unused) • Bit 1:0: block number When G_TL_DATAPATH=16: • Bits 15:3: memory address (higher bits are filled with 0 if unused) • Bits 2:0: block number See Section 2.7.2: Receive Buffer Layout for more information.

SHB_TRANS_RXUFC_REQ	24	**Update Flow Control DLLP transmission request information**:
		• Bits 11:0: Data credits (must use the same data scale as in K_RX_CRED)
		• Bits 19:12: Header credits (must use the same header scale as in K_RX_CRED)
		• Bits 21:20: Data type:
		  • 00: Posted
		  • 01: Non-Posted
		  • 10: Completion
		• Bit 22: Request Enable
		• Bit 23: Priority (0: low, 1: high)
		Acknowledgment is provided by the SHB_PHYDL_RXUFC_ACK index of the TLB_PHYDL signal.
		See Figure 9 and Figure 10 after this table for examples of how this interface is implemented.

SHB_TRANS_PM_REQ	4	**Request transmission of power-management DLLPs**: When one of the following bits is set, the Data Link layer continuously transmits the specified DLLPs: • PMREQ_NONE: do not send power management DLLPs • PMREQ_ENT_L1: send PM_Enter_L1 • PMREQ_ENT_L23: send PM_Enter_L23 • PMREQ_ASREQ_L1: send PM_Active_State_Request_L1 • PMREQ_ASREQ_ACK send PM_Request_Ack **Note:** There is no feedback signal indicating that the Core is transmitting the requested DLLP: the application must determine when to stop.

Table 7: tlb_trans signal

Index	Size	Description

SHB_TRANS_LTSSM_DIRECT	19	**Request LTSSM to change state**: Only one bit can be asserted at a time.
		• Bit 0: Request L0s: this bit must be asserted when the device is ready to enter L0s, and must be kept asserted as long as the device wishes to remain in L0s.
		• Bit 1: Request L1 entry: this bit causes LTSSM to go to the L1 state. It must be kept asserted when L1/ASPM L1 entry negotiation is complete and the device is ready to enter L1. De-asserting this bit when LTSSM is in L1 causes LTSSM to return to L0.
		• Bit 2: Request L2 entry: this bit causes LTSSM to go to the L2 state. It must be kept asserted when L2 entry negotiation is complete and the device is ready to enter L2. This bit must be de-asserted when the device wishes to exit L2 or, if CDC is implemented, when one or more receiver lanes exit electrical idle or when a WAKE# pin is asserted.
		Bit 3: Retrain link: this bit must be asserted when the Retrain Link bit is set in the Link Control register.
		• Bit 4: Request hot-reset (downstream only; must be 0 for upstream)
		• Bit 5: Request equalization (downstream only; must be 0 for upstream)
		• Bit 6: Request link width change: this bit must be asserted to request an autonomous link width change to one of the link widths specified by bits 15:12 (there is no bit for x1 since this link width is always supported). This signal must be asserted when LTSSM is in L0 or L0s state, and deasserted when the LTSSM is no longer in these states. Changes to link speed and width should not be requested concurrently.
		• Bit 7: Tcommon_mode timer not expired: This bit is only used when L1 sub-states are implemented, and must otherwise be tied to 0. It indicates that the Tcommon_mode timer has not yet expired when exiting L1.2, and that therefore the LTSSM must remain in Recovery.RcvrLock.
		• Bit 8: Request CLKREQ# assertion for L1 exit
		• Bit 9: Wake request (upstream only): this bit causes LTSSM to request a wake-up when in L2 state, either by asserting the WAKE# pin or by instructing the PHY to send a beacon. This bit must be kept asserted as long as LTSSM is in the L2/P2 state, as indicated by SHB_PHYDL_L2_P2.
		• Bit 10: Instructs data link layer to stop processing Tx buffer as soon as it is becomes empty, until this bit is de-asserted. This is used when the Core initiates ASPM L1 entry.
		The data Link layer asserts SHB_PHYDL_TXBUF_INH_ACKD when TLP transmission has stopped and all transmitted TLPs are acknowledged.
		• Bit 11: Disable Link: this bit must be asserted when link disable is requested from the upper layers and must be kept asserted until the link no longer needs to be in the Disable state. This bit must be asserted when either:
		  • the Disable Link bit is set in the Link Control register
		  • the application wants to disable the link
		• Bit 12: Link width change to x2 allowed.
		• Bit 13: Link width change to x4 allowed
		• Bit 14: Link width change to x8 allowed
		• Bit 15: reserved
		• Bit 16: Request speed width change: this bit must be asserted to request an autonomous link speed change to the speed specified by bits 18:17. This signal must be asserted when LTSSM is in L0 or L0s state, and deasserted when the LTSSM is no longer in these states. Changes to link speed and width should not be requested concurrently.
		• Bits 18:17: Maximum Link speed for autonomous speed change: 00=2.5GT/s, 01=5.0GT/s, 10=8.0 GT/s

SHB_TRANS_TIMER_100US	1	This input is used as a time reference; it must be asserted for one clock cycle every 100 µs in normal mode, or every 1 µs in simulation mode.

Table 7: tlb_trans signal

Index	Size	Description
SHB_TRANS_TIMER_1US	1	This input is used as a time reference; it must be asserted for one clock cycle every 1 µs.
SHB_TRANS_MAX_PAYLOAD	3	Maximum Payload Size value for the PCI Express Capability Device Control register.
SHB_TRANS_LINKCSR	5	PCI Express Capability Link Control register Bits 8:4 value: • Bit 0: reserved; the application must use SHB_TRANS_LTSSM_DIRECT[11] when the Link Disable bit is set. • Bit 1: reserved; the application must use SHB_TRANS_LTSSM_DIRECT[3] when the Retrain Link bit is set. • Bit 2: Common Clock Configuration bit • Bit 3: Extended Synch bit • Bit 4: Enable Clock Power Management bit
SHB_TRANS_LINKCSR2	16	PCI Express Capability Link 2 Control register value.
SHB_TRANS_BRSW_IN	4	Same as `TL_BRSW_IN[3:0]` (see Chapter 15: Bridge/Switch Interface).
SHB_TRANS_CLKREM	1	Same as `TL_PM_CLKCONTROL` (see Section 10.1).
SHB_TRANS_LOWERSKPEN	4	Indicates that SRIS lower SKP generation is enabled: • Bit 0: enabled at 2.5 GT/s • Bit 1: enabled at 5.0 GT/s • Bit 2: enabled at 8.0 GT/s • Bit 3: reserved
SHB_TRANS_TXPROTACK	1	Tx Protection Error acknowledge from the application (see Section 8.5).

Table 7: tlb_trans signal

The following waveform shows a simple example of a low-priority Update Flow Control DLLP transmission request, followed by a high-priority request.

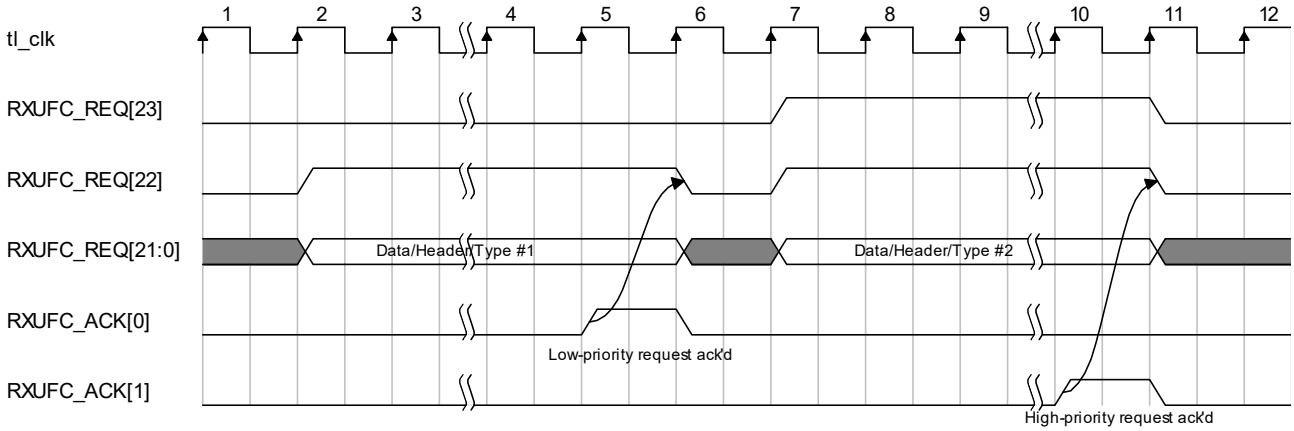


Figure 9: Simple example of an Update Flow Control DLLP transmission request

The following waveform shows the sending of a low-priority Update Flow Control DLLP transmission request. However, this request is then dropped to enable a high-priority request to be sent instead. In this case, if an acknowledgment for the low-priority request is received, it should be ignored.

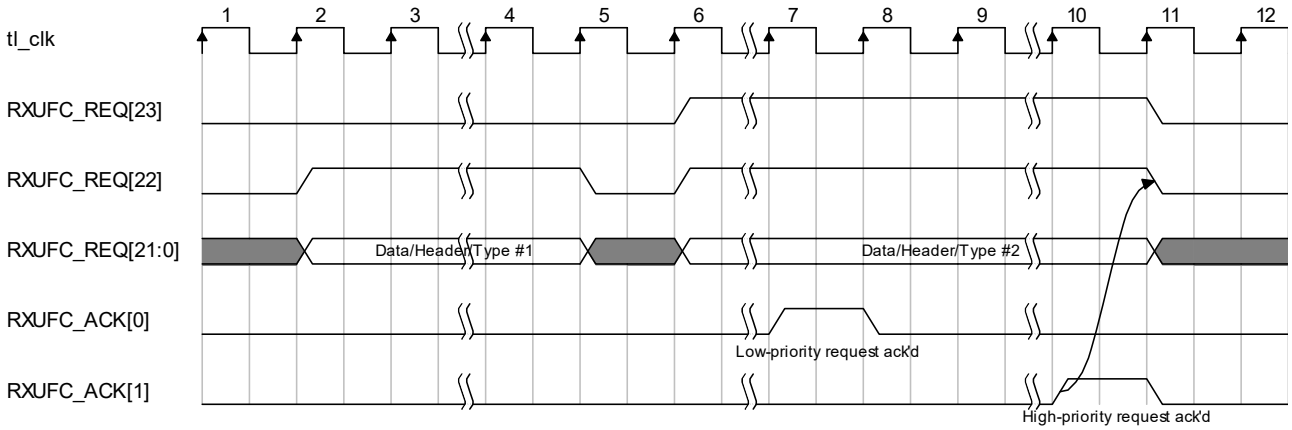


Figure 10: Example of a priority change for an Update Flow Control DLLP transmission request

2.7.1.2 tlb_phydl

The TLB_PHYDL[819:0] output signal contains information from the PHYMAC and Data Link layers to the Transaction Layer. It is synchronous to the TL_CLK clock. Indexes for this signal are defined in the include file pcie3_sharedbus_h.v, and are described in the following table:

Index	Size	Description
SHB_PHYDL_LINK_WIDTH	5	**Reports negotiated link width**: • 00001: x1 • 00010: x2 • 00100: x4 • 01000: x8 • 10000: x16
SHB_PHYDL_LINK_SPEED	2	**Reports negotiated link speed**: • 00: 2.5 GT/s • 01: 5 GT/s • 10: 8 GT/s • 11: 16 GT/s
SHB_PHYDL_LINK_TRAINING	1	Asserted when link training is in progress.
SHB_PHYDL_RECEIVER_ERROR	2	• Bit 0: pulsed when an 8B/10B or 128B/130B encoding error has been detected in LTSSM Config or L0 state. • Bit 1: pulsed when a framing error has been detected.
SHB_PHYDL_UPCONFIG_CAP	1	Reports that both link partners support Link up-configuration.
SHB_PHYDL_TXL0S_EXIT	1	Asserted for one clock cycle to indicate that LTSSM is exiting TX L0s.
SHB_PHYDL_BW_MGT_SET	1	Asserted for one clock cycle to indicate that the Link Bandwidth Management Status bit of the PCI Express Capability Link Status register must be set.
SHB_PHYDL_AUTO_BW_SET	1	Asserted for one clock to indicate that the Link Autonomous Bandwidth Status bit of the PCI Express Capability Link Control Register must be set.
SHB_PHYDL_TXMARGIN_RST	1	Asserted for one clock cycle to indicate that the Transmit Margin field of the PCI Express Capability Link Control 2 register must be reset.
SHB_PHYDL_ENTERCOMP_RST	1	Asserted to indicate that the Enter Compliance field of the PCI Express Capability Link Control 2 register must be reset.
SHB_PHYDL_LK_STAT2_REG	13	Contains information to be reported via the PCI Express Link Status registers: • Bit 0: Current TX de-emphasis value when operating at 5.0 GT/s • Bit 1: 8 GT/s Equalization complete • Bit 2: 8 GT/s Equalization phase 1 success • Bit 3: 8 GT/s Equalization phase 2 success • Bit 4: 8 GT/s Equalization phase 3 success • Bit 8:5: reserved • Bit 9: Retimer detected • Bit 10: Two retimers detected • Bit 12:11: Crosslink resolution
SHB_PHYDL_LK_STAT2_EQR	2	Asserted for one clock to indicate an equalization request has been received: • Bit 0: 8GT Equalization request • Bit 1: reserved
SHB_PHYDL_LTSSM_STATE	5	LTSSM state machine state (same as PL_LTSSM, see Chapter 16)
SHB_PHYDL_RLOCK_TS1	1	Indicates that LTSSM is in Recovery.Rcvrlock and that TS1's are both received and transmitted.

Table 8: tlb_phydl signal

Index	Size	Description
SHB_PHYDL_RXEIOS	1	Indicates that EIOS have been received on one or more configured lanes.
SHB_PHYDL_L2_P2	1	Indicates that the LTSSM is in L2.idle and its power state is P2.
SHB_PHYDL_L1_P2	1	Indicates that the LTSSM is in L1.entry or L1.idle and its power state is P2 (for use with Clock Power Management).
SHB_PHYDL_L2_EXIT	1	reserved
SHB_PHYDL_HOTRST_EXIT	1	reserved
SHB_PHYDL_DLUP_EXIT	1	reserved
SHB_PHYDL_BRSW_OUT	4	Same as `TL_BRSW_OUT[3:0]` (see Chapter 15: Bridge/Switch Interface).
SHB_PHYDL_LANE_ERROR	16	For each lane, asserted for one clock cycle to report that an error must be logged to the Secondary PCI Express Capability Lane Error Status register. • Bit 0: Lane 0 • ... • Bit 15: Lane 15
SHB_PHYDL_EQU_UP_RXPHTXP	112	Reports Rx Preset Hint and Tx Preset value at 8GT for each lane (upstream only): • Bits 3:0: Lane 0 TxPreset • Bits 6:4: Lane 0 RxPresetHint • ... • Bits 108:105: Lane 15 TxPreset • Bits 111:109: Lane 15 RxPresetHint
SHB_PHYDL_TURNOFF_LANES	16	For each lane, this signal is set if the lane is turned off: • Bit 0: Lane 0 • ... • Bit 15: Lane 15
SHB_PHYDL_LASTACKD_ADDR	16	**Transmit buffer last acknowledged TLP address**: This indicates that TLPs stored before this address have been successfully sent and acknowledged, so this space is available to write new TLPs. When G_TL_DATAPATH=8: • Bits 15:1: memory address (higher bits are filled with 0 if unused) • Bit 0: group number When G_TL_DATAPATH=16: • Bits 15:2: memory address (higher bits are filled with 0 if unused) • Bits 1:0: group number See Section 2.7.3: Transmit Buffer Layout for more information.
SHB_PHYDL_ACKNAK_ERROR	1	Asserted for one clock cycle to report that an ACK/ NAK protocol error has been detected.
SHB_PHYDL_REPLAY_ERROR	2	Asserted for one clock cycle to report replay error: • Bit 0: Replay timeout error • Bit 1: Replay number error
SHB_PHYDL_TXPATH_EOT_ACK	1	Reserved
SHB_PHYDL_TXPROTERR	32	Tx Protection Error reporting to the application (see Section 8.5).

Table 8: tlb_phydl signal

Index	Size	Description
SHB_PHYDL_TXBUF_INH_ACKD	1	Indicates that TX buffer is inhibited, as requested by SHB_TRANS_LTSSM_DIRECT[10], and that all transmitted TLPs are acknowledged.
SHB_PHYDL_CREDCONS	84	Consumed Credits: this signal reports the credits consumed by a TLP transmitted by the data link layer. It can be used to determine the exact number of credits consumed after one or more TLPs have been nullified by the data link layer in TX Error Protection, Error Nullification, or Error Flush modes (see Section 8.5). • Bits 15:0: Posted data consumed credits • Bits 27:16: Posted header consumed credits • Bits 43:28: Non-Posted data consumed credits • Bits 55:44: Non-Posted header consumed credits • Bits 71:56: Completion data consumed credits • Bits 83:72: Completion header consumed credits
SHB_PHYDL_RXUFC_ACK	2	Asserted for one clock to acknowledge transmission of Update Flow Control DLLP requested by TLB_TRANS: • Bit 0: Low-priority acknowledge • Bit 1: High-priority acknowledge **Note**: Low-priority acknowledge should be ignored if a high-priority request is in progress.
SHB_PHYDL_DATA_LINK_UP	1	Asserted when data link is up. This index is equivalent to the *PCIe Specification*'s DL_up flag.
SHB_PHYDL_DLCMSM_ACT	1	Indicates that the data link layer is active. This is equivalent to the *PCIe Specification*'s DL_active flag.
SHB_PHYDL_VC_INIT_CPLT	1	Asserted when VC0 flow-control initialization is complete.
SHB_PHYDL_REMOTE_DLF	24	Remote Data Link Feature: • Bit 22:0: Remote Data link Feature Supported • Bit 23: Remote Data Link Feature Supported Valid
SHB_PHYDL_SCALED_FC_EN	1	Indicates that scaled flow-controls are enabled.
SHB_PHYDL_TLPAVAIL_ADDR	16	**Receive buffer TLP available pointer position**: Indicates that blocks of data before this pointer contain valid TLPs that can be read. When G_TL_DATAPATH=8: • Bits 15:2: memory address (higher bits are filled with 0 if unused) • Bit 1:0: block number When G_TL_DATAPATH=16: • Bits 15:3: memory address (higher bits are filled with 0 if unused) • Bits 2:0: block number See Section 2.7.2: Receive Buffer Layout for more information.
SHB_PHYDL_RXBUF_NOTEMPTY	1	Indicates that the Rx buffer is not empty.
SHB_PHYDL_BADTLP_ERROR	1	Asserted for one clock cycle to report that a bad TLP has been received.
SHB_PHYDL_RXOVF_ERROR	1	Asserted for one clock cycle to report that a Receive buffer overflow error has been detected.
SHB_PHYDL_RXTLP	1	Reserved

Table 8: tlb_phydl signal

Index	Size	Description
SHB_PHYDL_PM_DET	4	Indicates that a power-management DLLP has been received: • PMREQ_NONE: no power management DLLP • PMREQ_ENT_L1: received PM_Enter_L1 • PMREQ_ENT_L23: received PM_Enter_L23 • PMREQ_ASREQ_L1: received PM_Active_State_Request_L1 • PMREQ_ASREQ_ACK received PM_Request_Ack
SHB_PHYDL_BADDLLP_ERROR	1	Asserted for one clock cycle to report that a bad DLLP has been received.
SHB_PHYDL_FCP_ERROR	1	Asserted for one clock cycle to report that a Flow-Control Protocol error has been detected.
SHB_PHYDL_FC_TIMEOUT	1	reserved

SHB_PHYDL_TXUFC_VECTOR	81	Reports most current flow-control values received:
		• Bits 26:0: posted credits information:
		  • Bits 11:0: data credits
		  • Bits 19:12: header credits
		  • Bit 21:20: data credits scale (11: x16, 10:x4, 0x:x1)
		  • Bit 23:22: header credits scale (11: x16, 10:x4, 0x:x1)
		  • Bit 24: data credits are infinite
		  • Bit 25: header credits are infinite
		  • Bit 26: set when credits information is valid and can be taken into account
		• Bits 53:27: non-posted credits information (same information as for posted credits, above)
		• Bit 80:54: completion data credits information (same information as for posted credits, above)

SHB_PHYDL_TLP_DLLP_REQ	1	Asserted to indicate that the Data Link layer has a TLP or DLLP to transmit. This information can be used by the transaction layer for low-power management.

Table 8: tlb_phydl signal

2.7.2 Receive Buffer Layout

Each line of the Receive buffer is organized as 4 (G_TL_DATAPATH=8) or 8 (G_TL_DATAPATH=16) blocks of (33+G_DATA_PROT)x2 bits:

- Offset 0: TLP data bits [31:0] (first byte received is on bits 63:56, last byte on bits 7:0)
- Offset 32: Parity bit(s) for TLP data bits [31:0], if data protection is enabled
- Offset 32+G_DATA_PROT: TLP data bits [63:32] (first byte received is on bits 63:56, last byte on bits 7:0)
- Offset 64+G_DATA_PROT: Parity bit(s) for TLP data bits [63:32], if data protection is enabled
- Offset 64+G_DATA_PROTx2: 2-bit packet framing indicator
  - 00: start or middle of packet
  - 01: end of packet, 2 dwords in last block
  - 10: end of packet, 1 dword in last block
  - 11: end of packet: TLP is incorrect and must be discarded

The examples below show Receive buffers (no data protection) containing 4 TLPs:

Memory Address	Block 0 [263:198]	Block 1	Block 2	Block 3 [65:0]
000				
001				00.H.H
010	10.H.H	00.H.H	00.H.D	00.D.D
011	00.D.D	00.D.D	00.D.D	10.D.-
100	00.H.H	00.H.H	00.D.D	01.D.D
101	00.H.H	00.H.D		
110				
111				

Table 9: Receive buffer (256-bit TL datapath) containing 4 TLPs

Memory Address	Block 0 [572:462]	Block 1	Block 2	Block 3 [329:264]	Block 4 [263:198]	Block 5	Block 6	Block 7 [65:0]
00x								00.H.H
01x	10.H.H	00.H.H	00.H.D	00.D.D	00.D.D	00.D.D	00.D.D	10.D.-
10x	00.H.H	00.H.H	00.D.D	01.D.D	00.H.H	00.H.D		
11x								

Table 10: Receive buffer (512-bit TL datapath) containing 4 TLPs

- TRANS_RXBUF_CURPTR0 = 00111b, indicating that all TLPs stored before this pointer have been read:
  - With a 256-bit TL datapath, this value points to address 001b, block 11b.
  - With a 512-bit TL datapath, this value points to address 00xb, block 111b.

- PHYDL_TLPAVAIL_ADDR = 10110b, indicating that blocks before this pointer contain TLPs that are ready to be read.
  - With a 256-bit TL datapath, this value points to address 101b, block 10b.
  - With a 512-bit TL datapath, this value points to address 10xb, block 110b.

Note: The Receive buffer cannot be full; when the read and write addresses point to the same location it means that the buffer is empty.

2.7.3 Transmit Buffer Layout

Each line of the Transmit buffer is organized as 4 (G_TL_DATAPATH=8) or 8 (G_TL_DATAPATH=16) blocks of (33+G_DATA_PROT)x2 bits

- Offset 0: TLP data bits [31:0] (first byte received is on bits 63:56, last byte on bits 7:0)
- Offset 32: Parity bit(s) for TLP data bits [31:0], if data protection is enabled
- Offset 32+G_DATA_PROT: TLP data bits [63:32] (first byte received is on bits 63:56, last byte on bits 7:0)
- Offset 64+G_DATA_PROT: Parity bit(s) for TLP data bits [63:32], if data protection is enabled
- Offset 64+G_DATA_PROTx2: 2-bits of attributes

Blocks work 2 by two (0&1, 2&3, as well as 4&5, and 6&7 if TL datapath is 512-bit) and each group of blocks contains the same TLP. Each of these groups contain up to 128 bits of data and a total of four attribute bits that are encoded as detailed below:

First Blocks of a TLP		Subsequent Blocks of a TLP	
11ix	**End/Nullified:** • Bit 1: TLP is internally generated • Bit 0: Last valid Dword position	11xx	**End/Nullified:** • Bit 1:0: Last valid Dword position
10ix	**End/LCRC Error:** • Bit 1: TLP is internally generated • Bit 0: Last valid Dword position	10xx	**End/LCRC Error:** • Bit 1:0: Last valid Dword position
01ix	**End:** • Bit 1: TLP is internally generated • Bit 0: Last valid Dword position	01xx	**End:** • Bit 1:0: Last valid Dword position
00ix	• Bit 1: TLP is internally generated • Bit 0:-	00xx	-

Table 11: Transmit Buffer Blocks Encoding

Last valid Dword position is defined as:

- On first blocks of a TLP:
  - 0: TLP ends on 3rd Dword
  - 1: TLP ends on 4th Dword
- On subsequent blocks of a TLP:
  - 00: TLP ends on 1st Dword
  - 11: TLP ends on 4th Dword

The examples below show a Transmit buffer containing four TLPs:

Memory Address	Block 0 [263:198]	Block 1	Block 2	Block 3 [65:0]
	Group 0		Group 1	
000				
001				
010	00.H.H	00.H.D	01.D.D {END}	01.-.-
011	01.H.H {END}	01.H.H	00.H.H	00.H.D
100	00.D.D	00.D.D	01.D.D {END}	10.-.-
101	01.H.H {END}	00.H.-		
110				
111				

Table 12: Transmit buffer (256-bit TL datapath) containing 4 TLPs

Memory Address	Block 0 [572:462]	Block 1	Block 2	Block 3 [329:264]	Block 4 [263:198]	Block 5	Block 6	Block 7 [65:0]
	Group 0		Group 1		Group 2		Group 3	
00x								
01x	00.H.H	00.H.D	01.D.D {END}	01.-.-	01.H.H {END}	01.H.H	00.H.H	00.H.D
10x	00.D.D	00.D.D	01.D.D {END}	10.-.-	01.H.H {END}	00.H.-		
11x								

Table 13: Transmit buffer (512-bit TL datapath) containing 4 TLPs

- PHYDL_LASTACKD_ADDR = 0100b, indicating that all TLPs stored before this pointer have been transmitted and acknowledged:
  - With a 256-bit TL datapath, this value points to address 010b, group 0b.
  - With a 512-bit TL datapath, this value points to address 01xb, group 00b.
-
- TRANS_TLPAVAIL_ADDR = 1011b, indicating that blocks before this pointer contain TLPs that are ready to be read.
  - With a 256-bit TL datapath, this value points to address 101b, group 1b.
  - With a 512-bit TL datapath, this value points to address 10xb, group 11b.

The application should update the TRANS_TLPAVAIL_ADDR pointer value, either:

- when the packet has been completely written to the buffer, or
- continuously (in streaming or cut-through mode), when the first 4xG_TL_DATAPATH DWORDs of the current TLP have been written to the TX buffer; the application should use this mode only when its throughput is >= current PCI Express link throughput.

Note: PHYDL_LASTACKD_ADDR=TRANS_TLPAVAIL_ADDR means that the Transmit buffer is empty; it cannot mean that the Transmit buffer is full.

Chapter 3 Clocks and Resets

3.1 Physical Layer Clocks and Resets

The following table describes clock and reset signals for the physical layer:

Signal	I/O	Description
pl_pclk	in	**Physical Layer PHY Clock**: This signal is the clock for the physical and data link layers. Its frequency depends on the PIPE interface configuration and signaling rate (see Table 17: PIPE Configurations).
pl_rstn	in	**Asynchronous Reset**: This active-low reset signal clears the physical and data-link layer logic. Assertion of this signal can be asynchronous with PL_PCLK but deassertion must be synchronous. When CDC is implemented, this signal must be kept asserted for a minimum of two PL_PCLK clock cycles when reset occurs.
pl_rstnp	in	**Asynchronous Reset for power-gated logic**: This active-low reset signal clears the physical and data-link layer power-gated logic. It is identical to PL_RSTN except that it can be additionally asserted when power is restored to power-gated logic. PL_RSTN and PL_RSTNP must be connected to the same reset source if power gating is not used.
pl_srst	in	**Synchronous Reset**: This active-high reset signal clears the physical and data-link layer logic. When CDC is implemented, this signal must be kept asserted for a minimum of two PL_PCLK clock cycles when reset occurs.
pl_npor	in	**Asynchronous Power-on Reset**: This signal is the active-low reset used to clear all sticky registers. Assertion of this signal can be asynchronous with PL_PCLK but deassertion must be synchronous. It is typically asserted at power-up and when the PCI Express fundamental reset (MPERST#) is asserted.
pl_spor	in	**Synchronous Power-on Reset:** This signal is the active-high reset used to clear all sticky registers. It is typically asserted at power-up and when the PCI Express fundamental reset (MPERST#) is asserted.
pl_exit[3:0]	out	**Physical Layer Exits**: Internal reset conditions: each bit in this active-low signal indicates a PCI Express event caused by an internally generated reset condition. • Bit 0: L2 exit: Asserted after an exit from L2 low-power state; all Core logic except sticky registers must be cleared. Resetting application logic is optional. • Bit 1: Hot Reset exit: Asserted when device exits Hot Reset LTSSM state, except if device was directed into this state by the Secondary Bus Reset register being set; all Core logic except sticky registers must be cleared. Resetting application logic is optional. • Bit 2: Data Link down: Asserted after the data link has gone down: all Core logic except sticky registers, and configuration registers if Core is not a Rootport, must be cleared. Resetting application logic is optional. • Bit 3: reserved

Table 14: Physical Layer Clock and Reset Signals

3.2 Transaction Layer Clocks and Resets

The following table describes clock and reset signals for the transaction layer:

Signal	I/O	Description
tl_clk	in	**Transaction Layer Clock** (can be gated): This signal is the clock for the transaction layer. It must be identical in phase and frequency to TL_CLKP, with the exception that this clock can be turned off when TL_PM_CLKSTATUS[3] is asserted (see Section 10.2.1 for more information). Designs that do not implement clock gating can connect TL_CLK and TL_CLKP to the same clock source. **Note:** CDC must be enabled if clock gating is implemented.
tl_clk_gate	out	**Transaction Layer Clock Gating Enable:** This signal is asserted when the Core is in either the L1 or L2 states and both the RX and TX buffers are empty. It is de-asserted when the Core exits L1/L2 states, or if the application starts transmitting a TLP or requests a power management event. This signal indicates that TL_CLK can be turned off in order to save power (TL_CLKP must run continuously, however). The application should not turn-off TL_CLK if it has transactions pending, and can restart TL_CLK at any time even if this signal is still asserted.
tl_clkp	in	**Transaction Layer Clock** (must run continuously): This signal is the clock for the transaction layer. It must be identical in phase and frequency to TL_CLK, but this clock cannot be gated and must always be running in all low-power states. Designs that do not implement clock gating can connect TL_CLK and TL_CLKP to the same clock source.
tl_clock_freq[21:0]	in	**Transaction Layer Clock Frequency**: The value of this signal indicates the frequency of TL_CLK, in units of MHz. Any value greater than 8 MHz is allowed. • When CDC is implemented: The application can reduce or increase transaction layer clock frequency as needed.The value of this signal must always reflect current clock frequency. • When CDC is not implemented: The application must use PL_RATE to report current PIPE clock frequency on this signal.
tl_rstn	in	**Asynchronous Reset**: This active-low reset signal clears all logic in the transaction layer except configuration registers. Assertion of this signal can be asynchronous with TL_CLK but deassertion must be synchronous. When CDC is implemented, this signal must be kept asserted for a minimum of two TL_CLK clock cycles when reset occurs; it must not be deasserted before physical layer resets are deasserted.
tl_rstnp	in	**Asynchronous Reset for power-gated logic**: This active-low reset signal clears the transaction layer power-gated logic. It is identical to TL_RSTN except that it can be additionally asserted when power is restored to power-gated logic. TL_RSTN and TL_RSTNP must be connected to the same reset source if power gating is not used.
tl_crstn	in	**Asynchronous Reset for Configuration Registers**: This active-low reset signal clears all configuration registers in the transaction layer, except sticky registers. Assertion of this signal can be asynchronous with TL_CLK but deassertion must be synchronous. When CDC is implemented, this signal must be kept asserted for a minimum of two TL_CLK clock cycles when reset occurs; it must not be deasserted before physical layer resets are deasserted. This signal must be asserted and deasserted at the same time as TL_RSTN, when applicable.
tl_srst	in	**Synchronous Reset**: This active-high reset signal clears all logic in the transaction layer except configuration registers. When CDC is implemented, this signal must be asserted for a minimum of two TL_CLK clock cycles when reset occurs; it must not be deasserted before physical layer resets are deasserted.

Table 15: Transaction Layer Clock and Reset Signals

Signal	I/O	Description
tl_crst	in	**Synchronous Reset for Configuration Registers**: This signal is the active-high synchronous reset of the Configuration Space. It clears all configuration registers, except sticky registers. When CDC is implemented, this signal must be asserted for a minimum of two TL_CLK clock cycles when reset occurs; it must not be deasserted before physical layer resets are deasserted. This signal must be asserted and deasserted at the same time as TL_SRST when applicable.
tl_npor	in	**Asynchronous Power on Reset**: This signal is the active-low reset used to clear all sticky registers. It is typically asserted at power-up and when the PCI Express fundamental reset (MPERST#) is asserted. Assertion of this signal can be asynchronous with TL_CLK but deassertion must be synchronous.
tl_spor	in	**Synchronous Power-on Reset**: This signal is the active-high reset used to clear all sticky registers. It is typically asserted at power-up and when the PCI Express fundamental reset (MPERST#) is asserted.
tl_flr_req0 (also for functions 1 - 7)	out	• **Function-Level Request from Physical Function**: This signal is asserted when an FLR is requested for a physical function. The application must prepare to reset and assert the FLR acknowledge bit in TL_REPORT_STATE when it is ready.

Table 15: Transaction Layer Clock and Reset Signals

3.2.1 Function-Level Reset

Function-Level Reset is a PCI Express feature that makes it possible to reset a device's physical functions or virtual functions independently.

For physical functions:

1. When the 'Initiate Function Level Reset' bit of the Device Control Register is set, TL_FLR_REQ is asserted. The application can also optionally initiate FLR by asserting the TL_REPORT_STATE 'Trigger FLR' bit.
2. The application must then perform any necessary preparation or clean-up tasks, and assert the 'FLR acknowledge' bit in TL_REPORT_STATE when it is ready to proceed.
3. This resets the physical function's configuration space, and means the application can reset any required logic.

Note: During the FLR process, when the PLDA controller is waiting to receive the 'FLR acknowledge' from the application, all IO, Memory, or Configuration requests received are treated as unsupported, although other types of TLP are still received normally. Transmitted requests are not affected.

For virtual functions:

1. When the 'Initiate Function Level Reset' bit of the Device Control Register is set, the 'FLR Request' bit of TL_CFG_VFREGS is asserted. The application can also optionally initiate FLR by asserting the TL_REPORT_VFSTATE 'Trigger FLR' bit.
2. The application must then perform any necessary preparation or clean-up tasks, and assert the 'FLR acknowledge' bit in TL_REPORT_VFSTATE when it is ready to proceed.
3. This resets the virtual function's configuration space, and means the application can reset any required logic.

Note: During the FLR process, when the PLDA controller is waiting to receive the 'FLR acknowledge' from the application, all Configuration requests received are treated as unsupported, although other types of TLP are still received normally. Transmitted requests are not affected.

3.3 Reset Strategies

3.3.1 Reset Conditions

The following table shows the different reset events and how each reset input should be controlled. Note that either synchronous or asynchronous resets should be used; not both.

	pl_npor & tl_npor or pl_spor & tl_spor	pl_rstn & tl_rstn or pl_srst & tl_srst	tl_crstn or tl_crst
Power-on reset	assert	assert for at least 2 clock cycles	assert for at least 2 clock cycles
MPERST# assertion	_[1]	assert for at least 2 clock cycles	assert for at least 2 clock cycles
pl_exit[0]	-	assert for at least 2 clock cycles	assert for at least 2 clock cycles
pl_exit[1]	-	assert for at least 2 clock cycles	assert for at least 2 clock cycles
pl_exit[2]	-	assert for at least 2 clock cycles	Upstream port: assert for at least 2 clock cycles. Downstream port: -

Figure 11: Reset Conditions

[1] Using MPERST# as a power-on-reset, it is possible to simplify a reset scheme if the device is not designed to be able to request wake-up from L2 when main power has been removed.

Note: TL_CRSTN/TL_CRST must be asserted and deasserted at the same time as TL_RSTN/TL_SRST when applicable.

3.3.2 Example Reset Code without CDC

The following general-purpose Verilog example code shows how to control asynchronous resets when CDC is not enabled. Note that `TL_CRSTN` is identical to `TL_RSTN` if the device is not a Rootport.

```
              assign pl_rdy = mperst & active_low_power_on_reset;

                always @(posedge pl_pclk or negedge pl_rdy)
                begin
                   if (pl_rdy==1'b0) begin
                      pl_rstn <= 1'b0;
                      pl_npor <= 1'b0;
                      tl_crstn <= 1'b0;
                   end else begin
                      // Deassert synchronously
                      pl_npor <= 1'b1;

                      // Reset controlled by PCIe core
                      if (pl_exit[2:0]!=3'b111)
                         pl_rstn <= 1'b0;
                      else
                         pl_rstn <= 1'b1;

                      if (pl_exit[1:0]!=2'b11 || (pl_exit[2]!=1'b1 &&
            device_is_not_rootport))
                         tl_crstn <= 1'b0;
                      else
                         tl_crstn <= 1'b1;
                   end
                end

                assign tl_npor = pl_npor;
                assign tl_rstn = pl_rstn;
```

3.3.3 Example Reset Code with CDC

The following general-purpose Verilog example code shows how to control asynchronous resets when CDC is enabled. Note that TL_CRSTN is identical to TL_RSTN if the device is not a Rootport.

```
assign pl_rdy = mperst & active_low_power_on_reset;

  always @(posedge pl_pclk or negedge pl_rdy)
  begin
     if (pl_rdy==1'b0) begin
        pl_rstn <= 1'b0;
        pl_npor <= 1'b0;
        pl_cnt  <= 2'b00;
     end else begin
        // Deassert synchronously
        pl_npor <= 1'b1;

        // Reset controlled by PCIe core
        if (pl_exit[2:0]!=3'b111) begin
           pl_rstn <= 1'b0;
           pl_cnt  <= 2'b00;
        // Keep reset asserted for at least 2 clock cycles
        end else if (pl_cnt!=2'b11)
           pl_cnt <= pl_cnt+1'b1;
        else
           pl_rstn <= 1'b1;
    end
  end

     always @(posedge tl_clk or negedge pl_rdy)
   begin
     if (pl_rdy==1'b0)
        tl_npor <= 1'b0;
     else
       // Deassert synchronously
       tl_npor <= 1'b1;
  end
  // Make sure all layers are reset together
  assign tl_rdy = mperst & active_low_power_on_reset & pl_rstn;

  always @(posedge tl_clk or negedge tl_rdy)
  begin
     if (tl_rdy==1'b0) begin
        tl_rstn <= 1'b0;
        tl_cnt  <= 2'b00;
     end else begin
        // Keep reset asserted for at least 2 clock cycles
        if (tl_cnt!=2'b11)
           tl_cnt <= tl_cnt+1'b1;
        else
           tl_rstn <= 1'b1;
     end
  end

// Following code is NOT necessary if device is not a rootport,
// pl_crst is identical to pl_rstn
     always @(posedge pl_pclk or negedge pl_rdy)
     begin
        if (pl_rdy==1'b0) begin
           pl_crstn <= 1'b0;
           pl_crstcnt  <= 2'b00;
        end else begin
```

```
               // Reset controlled by PCIe core
                       if (pl_exit[1:0]!=2'b11 || (pl_exit[2]!=1'b1 &&
              device_is_not_rootport))
                          pl_crstn <= 1'b0;
                          pl_crstcnt  <= 2'b00;
                       // Keep reset asserted for at least 2 clock cycles
                       end else if (pl_crstcnt!=2'b11)
                          pl_crstcnt <= pl_crstcnt+1'b1;
                       else
                          pl_crstn <= 1'b1;
                    end
                 end

                 assign tl_crstrdy = mperst & active_low_power_on_reset & pl_crstn;

                 always @(posedge tl_clk or negedge tl_crstrdy)
                 begin
                    if (tl_crstrdy==1'b0) begin
                       tl_crstn <= 1'b0;
                       tl_crstcnt  <= 2'b00;
                    end else begin
                       // Keep reset asserted for at least 2 clock cycles
                       if (tl_crstcnt!=2'b11)
                          tl_crstcnt <= tl_crstcnt+1'b1;
                       else
                          tl_crstn <= 1'b1;
                    end
                 end
```

Chapter 4 Core Parameters

4.1 Core Constants

Core constants are the basic Core settings defined when the Core is implemented. They can be set using the XpressRICH3 Configuration Wizard. Unlike Core signals, constants cannot be dynamically modified. In RTL code, these are known as Generics for VHDL and Parameters for Verilog.

The following table describes Core constants:

Constant	Value	Description
G_NUM_FUNC	1 - 8	**Number of Physical Functions**: This constant is used to specify the number of physical functions implemented in the Core. Three different configurations are possible: • **G_NUM_FUNC=1**: Any type of port is possible, port type is specified by `K_GEN[3:0]`. • **G_NUM_FUNC>1 and all functions are Native Endpoint / Legacy Endpoint**: Port type is specified by `K_GEN[3:0]`and is common to all functions. • **G_NUM_FUNC>1, Function #0 is Switch Upstream, all other functions are Native Endpoint:** This configuration is selected when **k_gen[3:0]** indicates that port is Switch Upstream.
G_MAX_VF	0 - 512	**Virtual Function Implementation**: Maximum number of virtual functions that can be implemented in the Core.
G_NUM_LANES	1 - 16	**Number of Lanes**: This constant defines the number of PCI Express lanes. Possible values are 1, 2, 4, 8 and 16.
G_RXBUF_SIZE	8/11 - 16	**Receive Buffer Size** is 2^G_RXBUF_SIZE in bytes. The value of G_RXBUF_SIZE is 11-16 (in normal mode) or 8-16 (in RX stream mode). This value must be >=9 when RX buffer read latency is more than 1 cycle.
G_RXBUF_LATENCY	11,12,13	**Receive Buffer Latency**: • 11: 1 read cycle, 1 write cycle • 12: 1 read cycle, 2 write cycles • 13: 1 read cycle, 3 write cycles • 21: 2 read cycles, 1 write cycle • 22: 2 read cycles, 2 write cycles • 23: 2 read cycles, 3 write cycles See Section 2.6.1 for more information.
G_TXBUF_SIZE	10 - 16	**Transmit Buffer Size** is 2^G_TXBUF_SIZE in bytes. **Note:** TX buffer size can be reduced at reset using `K_GEN[59:56]`; this makes it possible to change the TX buffer size according to how many memory blocks are available in the application for a specific configuration.
G_TXBUF_LATENCY	11,12,13, 21, 22, 23	**Transmit Buffer Latency**: • 11: 1 read cycle, 1 write cycle • 12: 1 read cycle, 2 write cycles • 13: 1 read cycle, 3 write cycles • 21: 2 read cycles, 1 write cycle • 22: 2 read cycles, 2 write cycles • 23: 2 read cycles, 3 write cycles See Section 2.6.1 for more information.

Table 16: Core constants

Constant	Value	Description
G_CDC_PRESENT	0 - 1	**CDC Implementation**: This constant is used to implement the CDC in the Core. If set to 1, the CDC is implemented. **Note:** A device that does not implement the CDC faces some limitations in low-power mode (see Chapter 10 and 195 Unsynchronized Paths). The CDC is mandatory if clock gating is implemented.
G_NUM_TLPRPL	4 - 10	**Max Number of TLPs**: This constant specifies the maximum number of TLPs in the Replay Buffer. The Replay Buffer can contain up to 2^G_NUM_TLPRPL TLPs. The value of G_NUM_TLPRPL must be less than or equal to the value of G_TXBUF_SIZE-6.
G_NPBUF_SIZE	1 - 8	**Non-Posted Buffer Size**: This constant is used to specify the depth of the non-posted reordering buffer. You must set non-posted header credits so that all received non-posted TLPs can be stored in the Non-Posted buffer. This ensures that no non-posted TLPs can block the RX interface when `TL_RX_MASKNP` is asserted. • **1**: buffer is not implemented. • **2 - 8**: buffer can contain 2^G_NBUF_SIZE-1 entries
G_NPBUF_WIDTH	4 - 13	**Non-Posted Buffer Width**: This constant is used to specify the width in units of DWords of the non-posted reordering buffer. The width must be set so that it is sufficient to store the largest possible non-posted TLP that the Core supports, and is not to exceed TL datapath width: necessary_width = MIN (4 + (1 if ECRC support) + *n*, G_TL_DATAPATH) Where *n* is: • 8 if CAS 128bit is supported • else, 4 if CAS 64bit is supported • else, 2 if FAdd/Swap 64-bit is supported • else, 1 if FAdd/Swap 32-bit is supported • else 0 (no Atomic Ops are supported)
G_PIPE_IF_W	2 or 4	**Maximum PIPE Interface Width**: • 2: maximum PIPE interface width is 16-bits • 4: maximum PIPE interface width is 32-bits See Table 17 for more information.
G_PIPE_IF_GEN3	2 or 4	**PIPE interface width 8.0 GT/s link speed**; see Table 17 for supported modes.
G_PIPE_IF_GEN2	1, 2 or 4	**PIPE interface width at 5.0 GT/s link speed**; see Table 17 for supported modes.
G_PIPE_IF_GEN1	0, 1, 2 or 4	**PIPE interface width at 2.5 GT/s link speed**; see Table 17 for supported modes.
G_RX_STREAM	0 or 1	**RX Stream**: Enables RX Stream mode. See Section 7.2 for more information.
G_DATA_PROT	0, 1, or 4	**Data Protection Configuration**: • 0: not implemented • 1: 1-bit parity per 32-bit word • 4: 4-bit parity per 32-bit word See Chapter 8 for more information.
G_DATAPATH	2 or 8	**Select Internal Receive Side Datapath Width**: • 2: 64-bit datapath • 8: 256-bit datapath (default) The 64-bit datapath reduces logic usage, power consumption, and latency and is available for x1 (all G_PIPE_IF values), x2 (all G_PIPE_IF values) or x4 devices where G_PIPE_IF = 2.
G_TL_DATAPATH	8 or 16	**Select Transaction Layer Datapath Width**: • 8: 256-bit datapath (default) • 16: 512-bit datapath

Table 16: Core constants

Constant	Value	Description
G_DESKEW_EXT	0 - 1	**Enable Extended Lane-Lane Deskew Support**: • 0: Normal lane-lane deskew • 1: Extended lane-lane deskew See Section 2.1.3: Deskew Block for more information.

Table 16: Core constants

The following table shows the various PIPE possible configurations, depending on the PIPE speed and link width:

Possible Link Widths	G_PIPE_IF_W (Max. PIPE width)	G_PIPE_IF_GEN3 (Gen3 PIPE config)	G_PIPE_IF_GEN2 (Gen2 PIPE config)	G_PIPE_IF_GEN1 (Gen1 PIPE config)
x1-x16	2	2 (16 bits 500 MHz)	2 (16 bits 250 MHz)	2 (16 bits 125 MHz)
x1-x16	2	2 (16 bits 500 MHz)	2 (16 bits 250 MHz)	1° (8 bits 250 MHz)
x1-x16	2	2 (16 bits 500 MHz)	1° (8 bits 500 MHz)	1° (8 bits 250 MHz)
x1-x16	2	2 (16 bits 500 MHz)	1° (8 bits 500 MHz)	0° (8 bits gapped[1] 500 MHz)
x1-x8	4	4 (32 bits 250 MHz)	4 (32 bits 125 MHz)	4 (32 bits 62.5 MHz)
x1-x8	4	4 (32 bits 250 MHz)	4 (32 bits 125 MHz)	2* (16 bits 125 MHz)
x1-x8	4	4 (32 bits 250 MHz)	2* (16 bits 250 MHz)	2* (16 bits 125 MHz)
x1-x8	4	4 (32 bits 250 MHz)	2* (16 bits 250 MHz)	1*° (8 bits 250 MHz)

Table 17: PIPE Configurations

*In these modes, `TXDATA[31:16]`, `RXDATA[31:16]`, `TXDATAK[3:2]`, and `RXDATAK[3:2]` are not used.

°In these modes, `TXDATA[15:8]`, `RXDATA[15:8]`, `TXDATAK[1]`, and `RXDATAK[1]` are not used.

[1]In this mode, `TXDATAVALID/RXDATAVALID` are used to enable one data byte every 2 clock cycles.

4.2 Core Variables

Core variables can be set using the XpressRICH3 Configuration Wizard.

However, ASIC customers are encouraged to implement a programming mechanism that enables them to access and modify these settings if needed.

Note: These variables should only be modified when the Core is in reset mode. Do not modify these variables if reset is not active as this may cause a malfunction of the Core.

Signal	I/O	Description

k_bar0[227:0]
(also for functions 1 - 7)
	In	**Default Bar Settings**: Indicates the size and attributes of base address registers and windows:
		• BAR0[31:0]:
		  • BAR0[0]: BAR type (0=memory, 1=IO)
		  • If BAR type is IO:
		  • BAR0[1]: reserved
		  • BAR0[31:2]: BAR size mask
		  • If BAR type is memory:
		  • BAR0[1]: resizable BAR
		  • BAR0[2]: 64-bit address space
		  • BAR0[3]: prefetchable
		  • If memory BAR is not resizable:
		  • BAR0[31:4]: Bar size mask
		  • If memory BAR is resizable:
		  • BAR0[23:4]: Supported BAR sizes (bit 4 = 1MB supported, bit 5 = 2MB supported,...bit 23 = 512GB supported)
		  • BAR0[28:24]: Default BAR size (0= 1MB, 1=2MB,.. 19=512 GB)
		  • BAR0[31:29]: reserved
		• BAR1 - BAR5[191: 32] have exactly the same settings as BAR0.
		• Expansion ROM
		  • k_bar0[202:192]: reserved
		  • k_bar0[223:203]: Expansion ROM size mask
		• Windows
		  • k_bar0[224]: IO window implemented
		  • k_bar0[225]: IO window 32-bit addressing support
		  • k_bar0[226]: Prefetchable memory window implemented
		  • k_bar0[227]: Prefetchable memory window 64-bit addressing support
		Note: Resizable BAR extended capability is automatically implemented if one or more BARs are set as resizable.

Table 18: Core Variables

Signal	I/O	Description

k_pciconf0[287:0]
(also for functions 1 - 7)
	In	**PCI Standard Configuration Settings**: Configures the value of standard PCI registers.
		
		• **Identification**
		  • k_pciconf0[15:0]: Vendor ID
		  • k_pciconf0[31:16]: Device ID
		  • k_pciconf0[39:32]: Revision ID
		  • k_pciconf0[63:40]: Class code
		  • k_pciconf0[79:64]: Sub-system vendor ID
		  • k_pciconf0[95:80]: Sub-system device ID
		
		• **Bridge**
		  • k_pciconf0[97:96]: Bridge secondary DEVSEL timing
		  • k_pciconf0[98]: Bridge secondary PCI 66 support
		  • k_pciconf0[99]: Bridge secondary PCI-X support
		  • k_pciconf0[100]: Bridge VGA addressing support
		  • k_pciconf0[102:101]: reserved
		
		• **Extended Capabilities**
		  • k_pciconf0[103]: Application-specific capability is implemented in the Physical Function's Enhanced Configuration Space at address 800h.
		  • k_pciconf0[111:104]: Offset for the first capability implemented in the Physical Function's Configuration Space application-specific area. Set to 00h if no application-specific capability is implemented.
		  • k_pciconf[112]: Address Translation Service support
		  • k_pciconf[113]: Page Request Interface support
		  • k_pciconf[114]: Multicast support
		  • k_pciconf[116:115]: reserved
		
		• **Legacy Power Management** *(see the Power Management 1.2 specification)*
		  • k_pciconf0[117]: DSI
		  • k_pciconf0[120:118]: Auxiliary current
		  • k_pciconf0[121]: D1 support
		  • k_pciconf0[122]: D2 support
		  • k_pciconf0[127:123]: PME support
		
		• **Interrupt & MSI**
		  • k_pciconf0[130:128]: Interrupt Pin
		  ◦ 000: none
		  ◦ 001: INTA
		  ◦ 010: INTB
		  ◦ 011: INTC
		  ◦ 100: INTD
		  • k_pciconf0[131]: Disable MSI capability in physical function: when this bit is set, the MSI capability of the PF is not implemented internally; it can be implemented in application-specific configuration registers using External Registers Configuration interface.
		  • k_pciconf0[134:132]: Number of MSI messages supported by the physical function (000:1,...101:32). This setting is not available when k_pciconf0[131]=1.
		  • k_pciconf0[135]: MSI per-vector masking supported by physical function (must be 1 if virtual functions are implemented). This setting is not available when k_pciconf0[131]=1.

Table 18: Core Variables

Signal	I/O	Description

k_pciconf0[287:0]
(also for functions 1 - 7)
(continued)
	In	• **PASID**
		  • k_pciconf[136]: PASID is supported
		  • k_pciconf[137]: Execution Permission supported
		  • k_pciconf[138]: Privileged Mode supported
		  • k_pciconf[143:139]: Max PASID Width (0 - 20)
		
		• **MSI-X**
		  • k_pciconf0[154:144]: Table size
		  • k_pciconf0[158:155]: reserved
		  • k_pciconf0[159]: Implement MSI-X capability in physical function
		  • k_pciconf0[162:160]: Table BIR
		  • k_pciconf0[191:163]: Table offset
		  • k_pciconf0[194:192]: PBA BIR
		  • k_pciconf0[223:195]: PBA offset
		
		• **Miscellaneous**
		  • k_pciconf[228:224]: ATS invalidate queue depth
		  • k_pciconf[229]: ATS page aligned request
		  • k_pciconf[230]: Implement TPH Requester extended capability
		  • k_pciconf[231]: TPH Requester Interrupt vector mode supported
		  • k_pciconf[232]: TPH Requester Device specific mode supported
		  • k_pciconf[233]: Extended TPH Requester supported
		  • k_pciconf[234]: PRI Global Invalidate support
		  • k_pciconf[235]: PRG Response PASID Required (PRI capability)
		  • k_pciconf[239:236]: reserved
		
		• **Multicast**
		  • k_pciconf[245:240]: MC Max Group
		  • k_pciconf[247:246]: reserved
		  • k_pciconf[253:248]: MC Window Size Requested
		  • k_pciconf[254]: reserved
		  • k_pciconf[255]: ECRC Regeneration Supported
		
		  • k_pciconf[275:256]: PRI outstanding page request capacity
		  • k_pciconf[286:276]: TPH Requester ST table size
		  • k_pciconf[287]: TPH Requester ST table location (0: 'not present' or 1: 'located in MSI-X structure'. The 'located in TPH Requester Capability' option is not supported.)

Table 18: Core Variables

Signal	I/O	Description
k_pexconf[383:0]	In	**PCI Express Configuration Settings**: Configures the value of PCI Express configuration registers. See the *PCI Express 3.0 Specifications* for more information about these registers. **Note:** Values in this signal are not used in transaction layer bypass mode, except for those fields preceded by a + symbol, which indicates that the fields are used by the physical and/or the data link layer. • **Device Capabilities** • +k_pexconf[2:0]: Maximum payload size • k_pexconf[5:3]: reserved • k_pexconf[8:6]: Endpoint L0s acceptable latency • k_pexconf[11:9]: Endpoint L1 acceptable latency • k_pexconf[27:12]: reserved • k_pexconf[28]: Function-level reset capability • k_pexconf[31:29]: reserved • **Device 2 Capabilities** • k_pexconf[35:32]: Completion timeout value • k_pexconf[36]: Completion timeout disable • k_pexconf[37]: ARI forwarding supported • k_pexconf[38]: AtomicOp routing supported • k_pexconf[39]: 32-bit AtomicOp completer support • k_pexconf[40]: 64-bit AtomicOp completer support • k_pexconf[41]: 128-bit AtomicOp completer support • k_pexconf[42]: No RO-enabled PR-PR passing • k_pexconf[43]: LTR mechanism supported • k_pexconf[45:44]: TPH completer supported • k_pexconf[47:46]: reserved • k_pexconf[48]: 10-bit tag completer supported • k_pexconf[49]: 10-bit tag requester supported • k_pexconf[50]: OBFF Message signaling supported • k_pexconf[51]: OBFF WAKE# signaling supported • k_pexconf[52]: **Disable IDO support**: the IDO Request Enable and IDO Completion Enable bits in the Device Control 2 Register are hardwired to 0 when this bit is set. • +k_pexconf[53]: End-End TLP Prefix supported • k_pexconf[55:54]: Max End-End TLP prefixes supported (01=1, 10=2, 11=3, 00=4) • k_pexconf[63:56]: reserved • **Link & Link 2 Capabilities** • k_pexconf[67:64]: SRIS Lower SKP OS Generation Supported Speeds Vector bits 3:0 (bits 6:4 are hardwired to 0) • - k_pexconf[71:68] SRIS Lower SKP OS Reception Supported Speeds Vector bits 3:0 (bits 6:4 are hardwired to 0) • k_pexconf[72]: Retimer and two retimers detection supported (downstream ports only) • +k_pexconf[73]: Crosslink supported • k_pexconf[74]: ASPM L0 supported • k_pexconf[75]: ASPM L1 supported • k_pexconf[78:76]: L0s exit latency • k_pexconf[81:79]: L1 exit latency • k_pexconf[82]: Clock power management supported • k_pexconf[83]: Surprise down error reporting capable • k_pexconf[84]: DLL active reporting capable • k_pexconf[87:85]: reserved • +k_pexconf[95:88]: Port number

Signal	I/O	Description

k_pexconf[383:0]
(Continued)
	In	• **Slot Capabilities**
		  • k_pexconf[96]: Attention button present
		  • k_pexconf[97]: Power controller present
		  • k_pexconf[98]: MRL sensor present
		  • k_pexconf[99]: Attention indicator present
		  • k_pexconf[100]: Power indicator present
		  • k_pexconf[101]: Hot-plug surprise
		  • k_pexconf[102]: Hot-plug capable
		  • k_pexconf[110:103]: Slot power limit value
		  • k_pexconf[112:111]: Slot power limit scale
		  • k_pexconf[113]: Electromechanical interlock present
		  • k_pexconf[114]: No command complete support
		  • k_pexconf[127:115]: Physical slot number
		
		• **Root Capabilities**
		  • k_pexconf[128]: Root Capabilities CRS software visibility
		  • k_pexconf[134:129]: reserved
		  • k_pexconf[135]: Rootport supports forwarding of End-End TLP prefixes
		  • k_pexconf[143:136]: Bus number for BFM mode (PLDA internal use only)
		
		• **Access Control Services**
		  • k_pexconf[144]: ACS Source Validation supported
		  • k_pexconf[145]: ACS Translation Blocking supported
		  • k_pexconf[146]: ACS P2P Request Blocking supported
		  • k_pexconf[147]: ACS P2P Completion Redirect supported
		  • k_pexconf[148]: ACS Upstream Forwarding supported
		  • k_pexconf[149]: ACS P2P Egress Control supported
		  • k_pexconf[150]: ACS Direct Translated P2P supported
		  • k_pexconf[151]: reserved
		  • k_pexconf[157:152]: Egress Control Vector Size (must be in the range 1 - 32 when ACS P2P Egress Control is supported; otherwise, value is 0).
		
		• **Precision Time Measurement**
		  • k_pexconf[158]: PTM Requester capable
		  • k_pexconf[159]: PTM Responder capable
		  • k_pexconf[160]: PTM Root capable
		  • k_pexconf[161]: -
		  • k_pexconf[169:162]: Local clock granularity
		
		• k_pexconf [170]: Vendor-defined End-End TLP prefixes are supported
		• k_pexconf [171]: reserved
		• k_pexconf[172]: Slot register implemented
		• k_pexconf[173]: Slot clock configuration (0:independent, 1:refclk)
		• +k_pexconf[174] Link selectable de-emphasis
		• k_pexconf[175]: Rootport RCB
		• k_pexconf[180:176]: Device number for downstream ports and BFM mode (PLDA internal use only)
		• k_pexconf[181]: Implement MSI-X capability in virtual functions; in this case VFs use the MSI-X settings from k_pciconf of the PF they are associated with.
		• k_pexconf[182]: Disable MSI capability in virtual functions: when this bit is set, MSI capability of VFs are not implemented internally; it can be implemented in application-specific configuration registers using Configuration External registers interface.

Table 18: Core Variables

Signal	I/O	Description

k_pexconf[383:0] (Continued)		• k_pexconf[185:183]: Number of MSI messages supported by virtual functions (000:1,...101:32). This setting is not available when k_pexconf[182]=1.
		• k_pexconf[189:186]: reserved
		• k_pexconf[190]: CAS 128-bit transmitter: must be set if the application can transmit 128-bit Compare-And-Swap TLPs; this affects the way transmit credits are checked.
		Note: ASPM L1 entry delay must always be > ASPM L0s entry delay.
		
		• **Error Checking and Reporting**
		  • k_pexconf[191]: AER implemented
		  • k_pexconf[192]: Header is logged for completion timeout errors reported by the application
		  • k_pexconf[193]: ECRC generation support
		  • k_pexconf[194]: ECRC checking support
		  • k_pexconf[199:195]: AER MSI message number
		  • k_pexconf[204:200]: PCI Express MSI message number
		
		• **ASPM**
		  • k_pexconf[209:205]: ASPM L0s entry delay (in steps of 256ns, from 1 - 31). A value of 0 indicates an entry delay of 32x256ns.
		  • k_pexconf[219:210]: ASPM L1 entry delay (in steps of 256ns, from 1 - 1023). For backward compatibility purposes, a value of 0 indicates an entry delay of 32x256ns. k_pexconf[222:220]: reserved
		  • k_pexconf[223]: Downstream port requires minimum Tx credits to accept ASPM L1 entry request.
		  • +k_pexconf[231:224]: Number of fast training sequences at 2.5 GT/s.
		  • +k_pexconf[239:232]: Number of fast training sequences at 5.0 GT/s.
		  • +k_pexconf[247:240]: Number of fast training sequences at 8.0 GT/s.
		  • +k_pexconf[255:248]: reserved
		
		• **L1 PM Substates**
		  • k_pexconf[256]: PCI-PM L1.2 supported
		  • k_pexconf[257]: PCI-PM L1.1 supported
		  • k_pexconf[258]: ASPM L1.2 supported
		  • k_pexconf[259]: ASPM L1.1 supported
		  • k_pexconf[260]: L1 PM substates supported (the L1 PM substates capability is only implemented if this bit is set)
		  • k_pexconf[263:261]: T_POWEROFF value in units of 256ns (000 = 256ns, 110 =7*256ns)
		  • k_pexconf[271:264]: Port common mode restore time
		  • k_pexconf[273:272]: Port T_POWER_ON scale
		  • k_pexconf[274]: reserved
		  • k_pexconf[279:275]: Port T_POWER_ON value
		
		• k_pexconf[287:280]: reserved
		
		• **Downstream Port Containment (DPC)**
		  • k_pexconf[292:288]: DPC interrupt message number
		  • k_pexconf[293]: reserved
		  • k_pexconf[294]: Poisoned TLP egress blocking supported
		  • k_pexconf[295]: DPC software triggering supported
		  • k_pexconf[299:296]: reserved
		  • k_pexconf[300]: DL_Active ERR_COR signaling supported
		  • k_pexconf[302:301]: reserved
		  • k_pexconf[303]: DPC is supported

Table 18: Core Variables

Signal	I/O	Description

k_pexconf[383:0] (Continued)		• k_pexconf[319:304]: reserved
		• **Data Link Feature**
		  • k_pexconf[342: 320]: Local Data Link Feature Supported (bit 0: Local Scaled Flow Control Supported, bit 22:1: reserved)
		  • +k_pexconf[343]: Data Link Feature Exchange Enable
		  • +k_pexconf[344]: Data Link Feature extended capability is implemented
		• k_pexconf[383: 345]: reserved

k_gen[63:0]	In	**General Settings**: Configures the value of PCI Express configuration registers:
		• k_gen[3:0]: PCI Express version
		  • 0011 : 3.0/3.1 compliant
		• k_gen[7:4]: Port type:
		  • 0000 : Native endpoint
		  • 0001 : Legacy endpoint
		  • 0100 : Rootport
		  • 0101 : Switch upstream
		  • 0110 : Switch downstream
		  • 0111 : PCIe to PCI/PCI-X bridge
		  • 1000 : PCI/PCI-X to PCIe bridge
		• k_gen[8]: Indicates that the device is operating in SRIS mode.
		• k_gen[9]: 2.5 GT/s speed supported
		• k_gen[10]: 5.0 GT/s speed supported
		• k_gen[11]: 8.0 GT/s speed supported
		• k_gen[15:12]: reserved for future speed support
		• k_gen[16]: BFM mode (PLDA internal use only)
		• k_gen[17]: Enable lane reversal support
		• k_gen[20:18]: reserved (was x2/x4/x8 down configuration capability disable)
		• k_gen[21]: Enable cut-through on receive path
		• k_gen[22]: Enable cut-through on transmit path
		• k_gen[23]: Perform equalization phases 2/3
		  • For downstream ports: setting this bit makes the Core execute phases 2/3; otherwise these phases are skipped.
		  • For upstream ports: setting this bit makes the Core perform remote transmitter adjustment during phase 2; otherwise the Core does not perform adjustment during this phase.
		• k_gen[27:24]: Disable lanes: forces a device to use fewer lanes than are physically implemented.
		  • 1111: lanes 15:1 are disabled
		  • 1110: lanes 15:2 are disabled
		  • 1100: lanes 15:4 are disabled
		  • 1000: lanes 15:8 are disabled
		  • 0000: all lanes are active
		• k_gen[28]: Use RxElecIdle to detect electrical idle entry at 2.5/5.0 GT/s (see 193 Electrical Idle Usage for more information)
		• k_gen[29]: reserved. **Note**: This bit was 'Auto speed jump' in earlier versions of the Core; however this is no longer applicable as the Core now always automatically trains to the highest possible speed after initial link up when it is a downstream port.

Table 18: Core Variables

Signal	I/O	Description

k_gen[63:0] (Continued)	In	• k_gen[30]:reserved (was PIE-8 compatibility mode until v175)
		• k_gen[31]: ECRC generation mode (see Section 7.3.1)
		• k_gen[32]: Physical functions interrupts are transmitted by the application using the TL_TX_... interface. See Section 9.4 for more information.
		• k_gen[34:33]: Tx Error Management Mode:
		  • 00: Reporting-only
		  • 01: Error Nullification
		  • 11: Error Flush
		• k_gen[35]: Enable RX stream watchdog
		• k_gen[36]: Up-configure capable: indicates that the Core supports link up-configuration
		• k_gen[37]: Enable RxValid Filter: This special logic can be used to solve issues if the PHY does not always deassert PL_RXVALID after EIOS is received.
		• k_gen[38]: Virtual functions interrupts are transmitted by the application using the TL_TX_... interface. See Section 9.4 for more information.
		• k_gen[39]: reserved
		• k_gen[47:40]: Mask Physical Functions: This setting make it possible to keep one or more function's logic in reset so that they cannot be accessed and configured from the PCIe bus or from application logic.
		  • k_gen[40]: n/a (Function #0 cannot be masked)
		  • k_gen[41]: Function #1 is masked
		  • ....
		  • k_gen[47]: Function #7 is masked
		• k_gen[51:48]: RX stream watchdog buffer low-level, used to deactivate RX stream watchdog:
		  • 0000: use default level (25% or less of space used, same as value 0011)
		  • 0001: 1/16th or less of space used
		  • ...
		  • 1111: 15/16th or less of space used
		• k_gen[55:52]: RX stream watchdog buffer high-level, used to activate RX stream watchdog:
		  • 0000: use default level (75% or more of space used, same as value 1100)
		  • 0001: 1/16th or more of space used
		  • ...
		  • 1111: 15/16th or more of space used
		• k_gen[59:56]: Limit TX Buffer Size:
		  • 1: limit to 2^10 bytes
		  • 2: limit to 2^11 bytes
		  • 3: limit to 2^12 bytes
		  • 4: limit to 2^13 bytes
		  • 5: limit to 2^14 bytes
		  • 6: limit to 2^15 bytes
		  • other values: no limit
		If the selected value limits the buffer size to a size that is >= to the size indicated by G_TXBUF_SIZE, then there is no limit.
		• k_gen[63:60]: reserved

Table 18: Core Variables

Signal	I/O	Description
k_equpreset[255:0]	In	**Equalization Preset Values for 8GT Equalization**: This signal indicates the default Preset and Receiver Preset Hint used for 8GT Equalization; if the Core is a downstream port then these values are also reported in the Lane Equalization Control Register of the Secondary PCI Express Extended Capability. 16-bits are defined for each lane: • Bits [15:0]: values for lane 0 • Bits [31:16]: values for lane 1 • ... • Bits [255:240]: values for lane 15 Per-lane values are described below: • Bits [3:0]: Port's default transmitter preset: this is the default preset that will be used until another preset/coefficient is requested by the link partner. • Bits [6:4]: Port's default receiver preset hint: this is the default preset hint that will be used until another value is requested by the link partner. • Bits [11:8]: Link partner's transmitter preset(1). • Bits [14:12]: Link partner's receiver preset hint(1). (1). If the Core is an upstream port then these values are used only if the Core becomes a Loopback master: in this case these values will be transmitted to the link partner when directing it to Loopback at 8GT/s. • Bit [15]: reserved **Note**: Transmitter preset and receiver preset values must be set to legal values, as specified in the *PCI Express Specification, section 4.2.3.2*.
k_rx_cred[95:0]	in	**Available Credits Settings**: k_rx_cred[15:0]: Posted Data credits k_rx_cred[27:16]: Posted Header credits k_rx_cred[31:28]: reserved k_rx_cred[47:32]: Non-Posted Data credits k_rx_cred[59:48]: Non-Posted Header credits k_rx_cred[63:60]: reserved k_rx_cred[79:64]: Completion Data credits k_rx_cred[91:80]: Completion Header credits k_rx_cred[95:92]: reserved You must observe the following rules when setting this signal: • Infinite credits are advertised if credits value is 0. • If scaled flow-controls are not supported, then header credits must be <=7Fh / data credits <=7FFh; otherwise header credits must be <=7FFh / data credits <=7FFFh • If a x16 scale is used, then credit values must be a multiple of 16, when header credits >=200h / data credits >=2000h; otherwise a x4 scale is used, and credit values must be a multiple of 4, when header credits >=80h / data credits >=800h; otherwise a x1 scale is used.

Table 18: Core Variables

Signal	I/O	Description
k_pipe[63:0]	in	**Optional Pipeline Settings**: This setting is used to enable or bypass register levels in the Core. • Enabling register levels helps achieve higher frequency at the expense of higher receive/transmit latency. • Bypassing registers allows lower receive/transmit latency but can cause problems with synthesis on slow device technologies. • k_pipe[0]: Disable TX transaction layer pipeline • k_pipe[1]: Disable RX transaction layer pipeline • k_pipe[15:2]: reserved • k_pipe[16]: Reserved (was RxElecidle resynchronization to PL_PCLK until v175) • k_pipe[17]: PIPE input signals are registered • k_pipe[18]: Tx aligned data is pipelined • k_pipe[19]: DC offset calculation is pipelined **Note**: When this bit is set then the TS1/TS2 DC symbols replacement logic is slightly modified to enable logic to run at much higher frequency. In this case, it no longer strictly conforms to the PCI Express specifications, however, this cannot cause any functional issue. • k_pipe[20]: De-Scrambled Data and 128b/130b OS Detection signals are pipelined • k_pipe[27:21]: reserved • k_pipe[28]: Reserved (was DLLP Insertion with TLP of atypical length is disabled until v175) • k_pipe[31:29]: reserved • k_pipe[32]: DLLP Status Stream Bus signals are pipelined • k_pipe[39:33]: reserved • k_pipe[40]: TxLCRC Data Xor Computation is pipelined • k_pipe[41]: TxLCRC Field Generation is pipelined • k_pipe[42]: RxLCRC Field Computation is pipelined (for ECRC only) • k_pipe[55:43]: reserved • k_pipe[56]: TLP decoder input data are pipelined (NA when G_DATAPATH = 2 [64-bit datapath]) • k_pipe[57]: SeqNumber and RxLCRC Data XOR Computation are pipelined (NA when G_DATAPATH = 2) • k_pipe[58]: RxLCRC Results are pipelined (NA when G_DATAPATH = 2) • k_pipe[59]: RxLCRC DWord to check is pipelined (NA when G_DATAPATH = 2) • k_pipe[60]: RxLCRC Checking Result is pipelined (NA when G_DATAPATH = 2) • k_pipe[61]: TLP Decoding signals are pipelined • k_pipe[63:62]: reserved **Note**: You should use the following values for this signal: • 3F000701 001E0000h for devices where meeting timing requirements is difficult, or if latency is not important. • 08000001 00000000h for devices where timing requirements can be met easily, or if latency has to be kept to a minimum.

Table 18: Core Variables

Signal	I/O	Description

k_sriov0[383:0]
(also for functions 1 - 7)
	in	**SR-IOV Virtual Functions Settings**:
		• k_sriov0[15:0]: VFs device ID
		• k_sriov0[31:16]: VFs subsystem device ID
		• k_sriov0[43:32]: Supported page sizes [11:0] (Mandatory page sizes are always supported regardless of this value.)
		• k_sriov0[45:44]: reserved
		• k_sriov[46]: VF 10-bit tag requester supported
		• k_sriov0[47]: Application-specific extended capabilities implementation:
		  • 0: application-specific extended capability is not implemented
		  • 1: application-specific extended capabilities are implemented, and the first capability is located at offset 800h of the Configuration Space.
		• k_sriov0[55:48]: Function dependency (This value must be equal to the function ID if there is no dependency.)
		• k_sriov0[63:56]: Application-specific capabilities implementation:
		  • 00h: application-specific capability is not implemented.
		  • 40..7Ch: application -specific capabilities are implemented, and the value set here indicates the offset of the first capability in the Configuration Space.
		• k_sriov0[95:64]: VF BAR0:
		  • Bit 0: reserved
		  • Bits 2:1: memory space type (10: 64-bit address, 00: 32 bit address)
		  • Bit 3: prefetchable
		  • Bits 11:4: reserved
		  • Bits 31:12: BAR size mask
		• k_sriov0[127:96]: VF BAR1 (as for VF BAR0)
		• k_sriov0[159:128]: VF BAR2 (as for VF BAR0)
		• k_sriov0[191:160]: VF BAR3 (as for VF BAR0)
		• k_sriov0[223:192]: VF BAR4 (as for VF BAR0)
		• k_sriov0[255:224]: VF BAR5 (as for VF BAR0)
		• k_sriov0[265:256]: VF base: FID of first VF assigned to this PF.
		• k_sriov0[275:266]: VF top: FID of last VF assigned to this PF +1.
		• k_sriov0[383:276]: reserved
		Note: VF BAR0 - 5 settings must be zeroed if no VF is assigned to this PF.

Table 18: Core Variables

4.3 Allocating Credits in the Receive Buffer

The Receive buffer must be divided into reserved spaces for both header and payload for posted, non-posted and completion TLPs. The size of each space is expressed in credits (1 credit = 16 bytes). Credits should use all the buffer space, as shown below:

- For devices not supporting ECRC:
  - Receive buffer size (bytes) = (PH + NPH + CPLH)* (16 + 4*Max_#_supported_End-End_TLP_prefixes) + (PD + NPD + CPLD)*16
- For devices supporting ECRC:
  - Receive buffer size (bytes) = (PH + NPH + CPLH)* (24 + 4*Max_#_supported_End-End_TLP_prefixes) + (PD + NPD + CPLD)*16

The following table shows an example of credit allocation that is appropriate for most general purpose devices (if ECRC and End-End TLP prefixes are not supported):

Rx Buffer	CPLD		CPLH		NPD		NPH		PD		PH	
Size (KB)	Credits	Bytes	Credits	Bytes	Credits	Bytes	Credits	Bytes	Credits	Bytes	Credits	Bytes
1	32	512	8	128	2	32	2	32	16	256	4	64
2	64	1024	16	256	4	64	4	64	32	512	8	128
4	128	2048	32	512	8	128	8	128	64	1024	16	256
8	256	4096	64	1024	16	256	16	256	128	2048	32	512
16	512	8192	128	2048	32	512	32	512	256	4096	64	1024
32	1024	16384	256	4096	64	1024	64	1024	512	8192	127	2032

Table 19: Credit allocation in the Receive buffer

Available credits must be reported via the core variable `K_VC_CRED0` (see Section 4.2); however the CPLD/CPLH credits field of this signal must always be 0 (infinite) for Endpoint devices. An Endpoint device must never issue non-posted requests unless it has enough Receive buffer space to receive all related completions.

Note: Incorrect values for `K_VC_CRED0` can lead to Receive buffer overflow (if the credits are higher than the actual available space in the buffer), or decreased performances (if the credits are lower than the actual available space in the buffer).

Chapter 5 PHY Interface

5.1 PIPE Interface Signals

The Core can be connected to any PIPE-compliant PHY through the PIPE interface, with the PIE-8 interface being used for communication between the MAC and the PHY during equalization. All PHY interface signals are synchronous to the PHY Clock (PL_PCLK).

PIPE interface signals for all lanes are combined into a bus. The signal or bus corresponding to lane 0 is on the least significant portion of the combined bus, and the signal or bus corresponding to lane G_NUM_LANES-1 is on the most significant portion of the combined bus

Note: In the following table, the notation N_L is used to represent [G_NUM_LANES-1:0].

Signal	I/O	Width	Description
PIPE signals common to all lanes			
pl_powerdown (common to all lanes)	out	[1:0]	**Power Down**: determines the power state (P0, P0s, P1, or P2).
pl_rate (common to all lanes)	out	[1:0]	**Link Signaling Rate**: controls the link signaling rate: • 00: use 2.5 GT/s signaling rate • 01: use 5.0 GT/s signaling rate • 10: use 8.0 GT/s signaling rate • 11: reserved
pl_width	out	[1:0]	**PIPE Interface Width:** indicates current PIPE interface width: • 00: 8-bit • 01: 16-bit • 10: 32-bit This signal can be left unconnected if not present on the PHY.
pl_pclk_rate	out	[2:0]	**PIPE Interface Clock Rate**: indicates current PIPE interface clock rate: • 000: 62.5 MHz • 001: 125 MHz • 010: 250 MHz • 011: 500MHz • 100: 1 GHz • 111:101: reserved This signal can be left unconnected if not present on the PHY.
pl_txmargin (common to all lanes)	out	[2:0]	**Transmit Margin**: selects transmitter voltage levels.
pl_txswing (common to all lanes)	out	1	Transmitter Voltage Swing Level
pl_txdeemph (common to all lanes)	out	1	Transmitter Deemphasis at 5.0 GT/s
pl_blockaligncontrol (common to all lanes)	out	1	Block Alignment Control: used at 8.0 GT/s only. This signal can be left unconnected if this port does not exist on the PHY.
PIPE output signals (per lane)			
pl_txdetectrx	out	N_L	**Transmit Detect Receive**: prompts the PHY to start a receiver detection operation or to begin loopback. If the PHY has a single Tx/Rx Detect input, connect only PL_TXDETECTRX Bit 0 to the input.

Table 20: PIPE interface signals

Signal	I/O	Width	Description
pl_txdata	out	G_PIPE_IF_W*8*N_L	**Transmit Data**
pl_txdatak	out	G_PIPE_IF_W*N_L	Transmit Data Control
pl_txdatavalid	out	N_L	Used at 8.0 GT/s or above only
pl_txstartblock	out	N_L	Used at 8.0 GT/s or above only
pl_txsyncheader	out	2*N_L	Used at 8.0 GT/s or above only
pl_txelecidle	out	N_L	**Transmit Electrical Idle**: forces the transmit output to electrical idle.
pl_txcompliance	out	N_L	**Transmit Compliance**: forces the running disparity to negative in Compliance mode (negative COM character) and can also be used to turn off Lanes that are not initialized.
pl_rxpolarity	out	N_L	**Receive Polarity**: prompts the PHY layer to perform a polarity inversion on the receiver decoding block.
pl_rxstandby	out	N_L	**PHY RX Control**: This signal can be left unconnected if this port does not exist on the PHY.
pl_pclk_change_ack	out	N_L	**PHY Change Complete**: this signal is only used when PCLK is an input to the PHY, it must be left unconnected otherwise.
PIPE input signals (per lane)			
pl_phystatus	in	N_L	**PHY Status:** If PHY has a single PHY status output, the same value can be replicated on all PL_PHYSTATUS bits.
pl_rxstatus	in	3*N_L	**Receive Status**: encodes receive status and error codes for the receive data stream and receiver detection.
pl_rxdata	in	G_PIPE_IF_W*8*N_L	**Receive Data**.
pl_rxdatak	in	G_PIPE_IF_W*N_L	Receive Data Control
pl_rxdatavalid	in	N_L	Used at 8.0 GT/s or above only
pl_rxstartblock	in	N_L	Used at 8.0 GT/s or above only
pl_rxsyncheader	in	2*N_L	Used at 8.0 GT/s or above only
pl_rxelecidle	in	N_L	**Receive Electrical Idle**: indicates that electrical idle is detected on receiver lane.
pl_rxvalid	in	N_L	**Receive Valid**: indicates symbol lock and valid data on PL_RXDATA and PL_RXDATAK.
pl_pclk_change_ok	in	N_L	**PHY ready for PCLK Rate Change**: this signal is only used when PCLK is an input to the PHY; it must be tied to 0's if not present on the PHY.

Table 20: PIPE interface signals

Signal	I/O	Width	Description
PCLK PLL control signals			
pl_pll_rate	out	[2:0]	**PCLK PLL Rate Change Control**: this signal is only used when PCLK is an input to the PHY; it must be left unconnected otherwise. When there is a rate change, the Core changes the value of this signal when it is safe to change the PCLK frequency. The PLL generating the PCLK must then be programmed to change the PCLK frequency (see Figure 12). • 000: 62.5 MHz • 001: 125 MHz • 010: 250 MHz • 011: 500MHz • 100: 1 GHz • 111...101: reserved
pl_pll_ack	in	1	**PCLK PLL Rate Change Acknowledge**: this signal is only used when PCLK is an input to the PHY, it must be tied to 0 otherwise. When PL_PLL_RATE value has changed, the PLL generating the PCLK must be programmed to change the PCLK frequency. When this is complete and the PCLK is stable at the new frequency, then this signal must be asserted for one clock cycle (see Figure 12).

Table 20: PIPE interface signals

The following waveform shows the protocol used to change PCLK frequency when it is an input to the PHY. In this example, the Core changes the PIPE interface from 16-bit at 125MHz (2.5GT) to 32-bit at 250 MHz (8 GT):

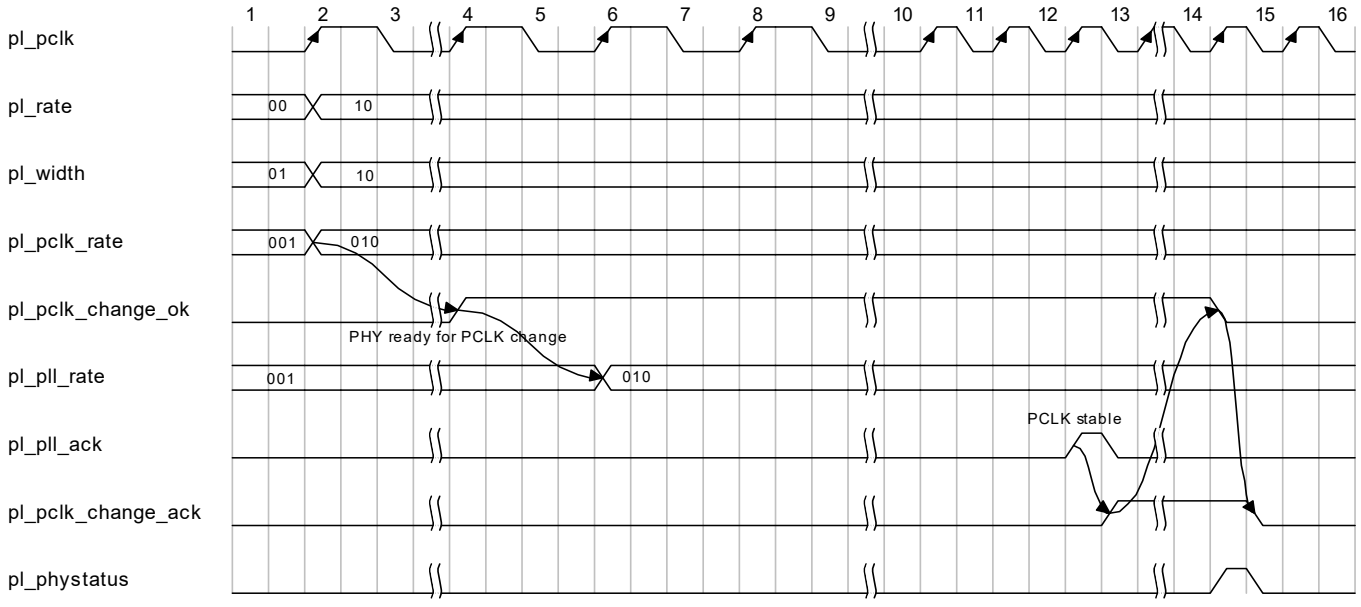


Figure 12: PCLK rate change sequence

5.2 Lane Margining at the Receiver

The Core supports lane margining at the receiver as defined in the *PCIe 4.0 Specification*. The lane margining process is entirely controlled by software and the Core only executes and responds to the commands it receives; therefore, there are no application interface signals or application-specific settings to control this process.

The Core communicates with the PHY using the PL_M2P_MSGBUS/PL_P2M_MSGBUS message bus interface, which is defined in the *PIPE Specification v4.4.1*. See *Chapter 9.5* of this specification for examples of Rx Margining sequences.

Note:

- The Core only issues Write Committed commands to the PHY.
- The Core accepts both Write Committed and Write Uncommitted commands from the PHY, but does not currently support Read commands.

Chapter 6 Equalization Interface

The Equalization interface is a set of signals that enables the Core's MAC layer to interact with the PHY in order to set up transmit and receive settings and to perform equalization when the Core operates at 8.0 GT/s.

Note: In the following table, the notation N_L is used to represent [G_NUM_LANES-1:0].

Signal	I/O	Width	Description
pl_macdataen	out	N_L	**MAC to PHY command/data enable**: When asserted this signal indicates that PL_MACDATA contains a valid command or valid data. This output can be left unconnected if the Core does not support 8.0 GT/s link rate.
pl_macdata	out	6*N_L	**MAC to PHY command/data**: The first data phase is a command code, the following phases (if present) contain data sent to the PHY. This output can be left unconnected if the Core does not support 8.0 GT/s link rate.
pl_phydataen	in	N_L	**PHY to MAC response/data enable**: When asserted, this signal indicates that PL_PHYDATA contains a valid response or valid data. If the PHY needs to send a response then it must wait until PL_MACDATA has been completely received and the PHY is ready (which may be any reasonable amount of time) before sending the response. This signal must be tied to 0 if the Core does not support 8.0 GT/s link rate.
pl_phydata	in	6*N_L	**PHY to MAC response/data**: The first data phase is a response code, the following phases (if present) contain response data sent to the Core. This signal must be tied to 0 if the Core does not support 8.0 GT/s link rate.

Table 21: Equalization interface signals

An independent Equalization interface is implemented for each lane in the Core. This means that different response data can be transmitted on each lane, and the PHY may send responses at different times. However, the Core always sends the same command on all lanes at the same time.

The waveform below shows typical behavior for this interface. In this example:

- the Core sends a command, with one data phase
- when it is ready, the PHY sends a response, with three data phases

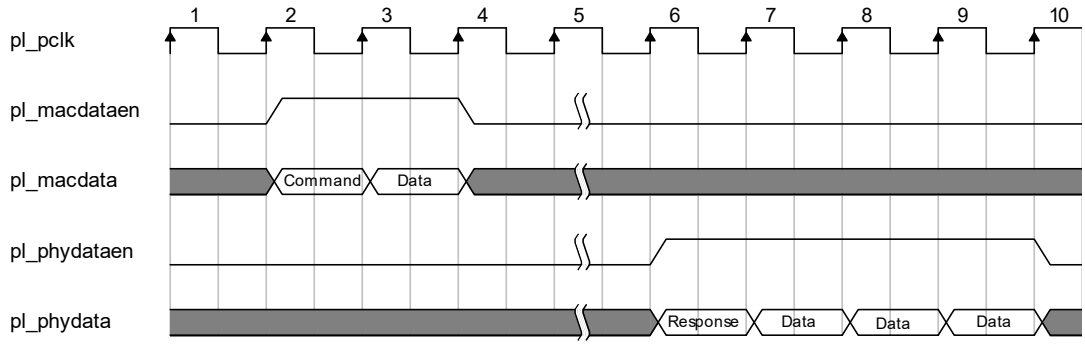


Figure 13: Typical equalization interface behavior

6.1 MAC to PHY Commands

The following table shows the commands data that the Core's MAC layer can send to the PHY:

Code	Command	Data	Possible PHY to MAC Response
0	Set Initial Preset	Tx preset RX preset hint	0: Local Preset to Coefficients Mapping
1	Request Local Preset	Tx preset	0: Local Preset to Coefficients Mapping OR 2: Local Coefficients Request Rejected
2	Request Local Coefficient	Tx C_{-1} coefficient Tx C_0 coefficient Tx C_{+1} coefficient	1: Local Coefficients Request Accepted OR 2: Local Coefficients Request Rejected
3	Remote Coefficients	Remote Tx C_{-1} Remote Tx C_0 Remote Tx C_{+1} Remote FS Remote LF	None
4	Perform Receiver Evaluation	None	5: Equalization complete; successful OR 6: Equalization complete; failed OR 3: New Remote Preset Request OR 4: New Remote Coefficient Request
5	Remote Request Rejected	None	None
Other	-	-	Must be ignored; no response.

Table 22: MAC to PHY commands

6.2 PHY to MAC Responses

The following table shows response data that the PHY can return to the Core's MAC layer:

Code	Description	Data
0	Local Preset to Coefficient Mapping	Local Tx C_{-1}, Local Tx C_0, Local Tx C_{+1}, Local FS, Local LF
1	Local Coefficients Request Accepted	None
2	Local Coefficient Request Rejected	None
3	New Remote Preset Request	Remote Tx preset
4	New Remote Coefficients Request	Remote Tx C_{-1}, Remote Tx C_0, Remote Tx C_{+1}
5	Equalization Complete, Successful	None
6	Equalization Complete, Failed	None
Other	Reserved, must not be used by PHY	-

Table 23: PHY to MAC commands

6.3 Equalization Interface Behavior

6.3.1 Initial Settings

After switching to 8.0 GT/s link rate and before exiting electrical idle, the Core provides the PHY with the local transmitter preset and the receiver preset hint. The PHY then responds with the set of coefficients that correspond to the transmitter preset, and its FS (Full Swing) and LF (Low Frequency) values.

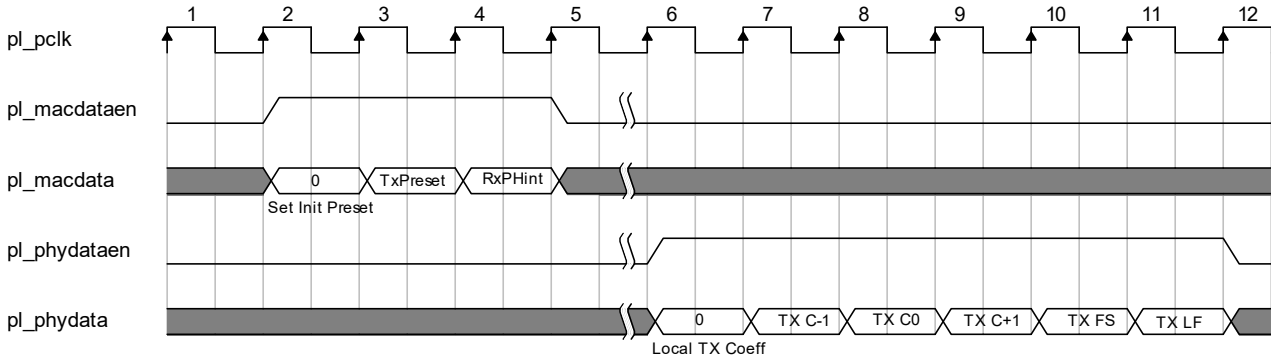


Figure 14: Set Initial Preset

Note:

- The Tx Preset value is 4 bits so `PL_MACDATA[5:4]` must be tied to 0 on the corresponding data phase.
- The Rx Preset Hint value is 3 bits so `PL_MACDATA[5:3]` must be tied to 0 on the corresponding data phase.

6.3.2 Local Transmitter Adjustment

The Core can adjust local transmitter settings when it receives a request to use a new preset or coefficients from its link partner during the equalization process. These requests can be accepted or rejected as described in the *PCI Express specification*, and each lane can accept or reject requests independently.

The waveform below illustrates a request to change the local Tx preset, which is accepted by the PHY:

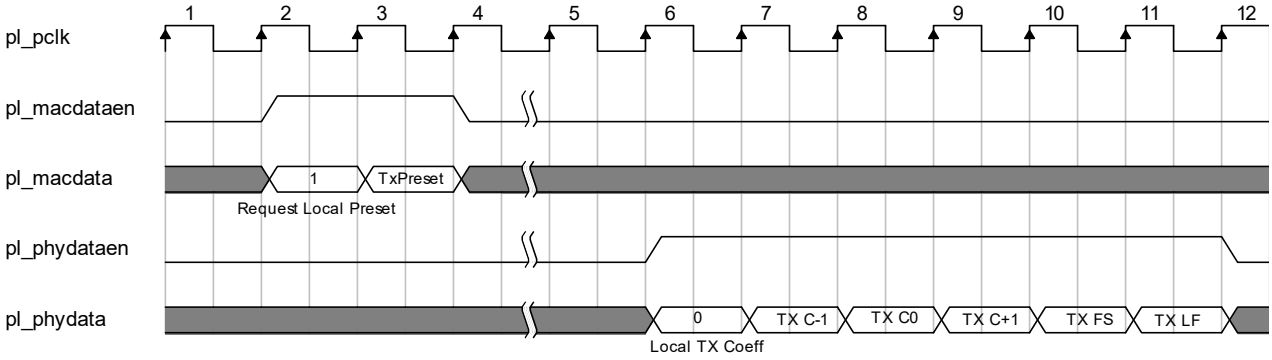


Figure 15: Set Local Preset Accepted

The waveform below illustrates a request to change the local coefficients:

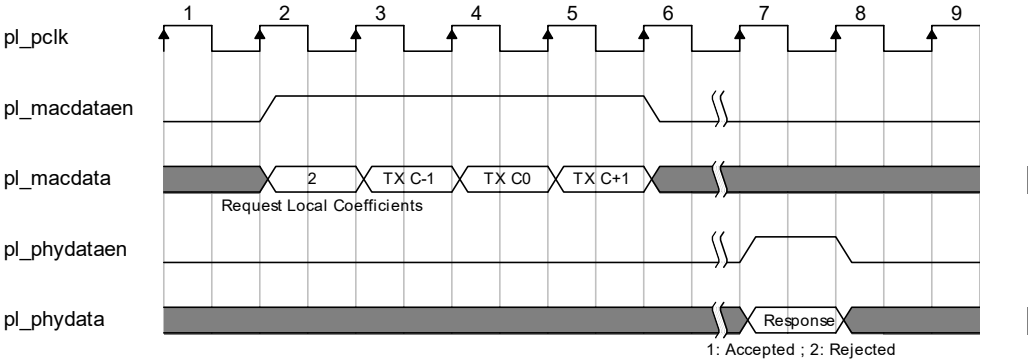


Figure 16: Set Local Coefficient

6.3.3 Remote Transmitter Adjustment

During the equalization process the Core attempts to fine-tune the remote transmitter settings, unless equalization phases 2/3 are disabled via the k_gen variable (see Section 4.2).

This tuning process is performed in one or more iterations of the following process:

1. The Core sends a Remote Coefficients command to the PHY with current remote transmitter settings.
2. The Core sends a Perform Receiver Evaluation command to the PHY.
3. The PHY performs internal tasks and sends a response to the Core.
4. If the PHY response is:
   - Equalization Complete: the Core stops fine-tuning processes for the lane that has received this response, and step 5 is not executed. If the status is “Failed”, then this error is reported in the Link Status 2 register (see 150 Register Content of the Configuration Space).
   - New Remote Preset Request or New Remote Coefficient Request: the Core sends new parameters to its link partner.
5. The Core then waits until the new parameters have been applied:
   - If the link partner correctly applied the settings and there is enough time left to perform a new fine-tuning cycle, then the Core sends a Remote Coefficient command with updated coefficients to the PHY and goes to step 2; otherwise the Core stops the fine-tuning process.
   - If the link partner rejects the new parameters or if TS1's acknowledging preset/coefficient change is not received in a 2ms delay then the Core sends a Remote Request Rejected to the PHY. Then, if there is

enough time left to perform a new fine-tuning cycle, the Core goes to step 2; otherwise the Core stops the fine-tuning process.

- If data can no longer be received correctly on the lane: the Core requests that the link partner use the last good parameters and sends a Remote Request Rejected command to the PHY. Then, if there is enough time left to perform a new fine-tuning cycle, the Core goes to step 1; otherwise the Core stops the fine-tuning process.

The Core allows a maximum of 20ms (200 µs in simulation) to perform the complete tuning process. For example, if a PHY requires 1ms for each evaluation, then it can perform up to 20 evaluations. The number of evaluations and time allowed per evaluation can be of any combination as long as all evaluations are executed within 20 ms.

The waveform below shows the Remote Tx Coefficients command:

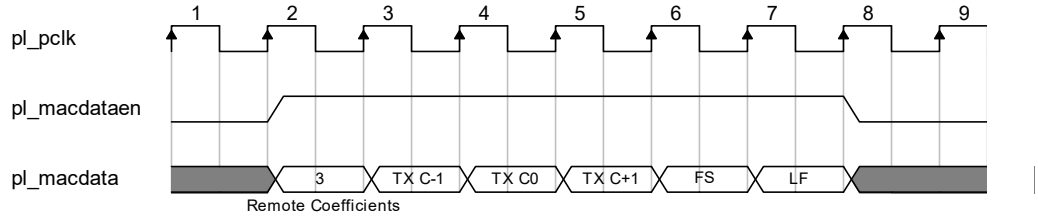


Figure 17: Remote Coefficients

The waveform below shows an evaluation request followed by a New Remote Preset Request command; the Core later reports that its link partner rejected the new parameters:

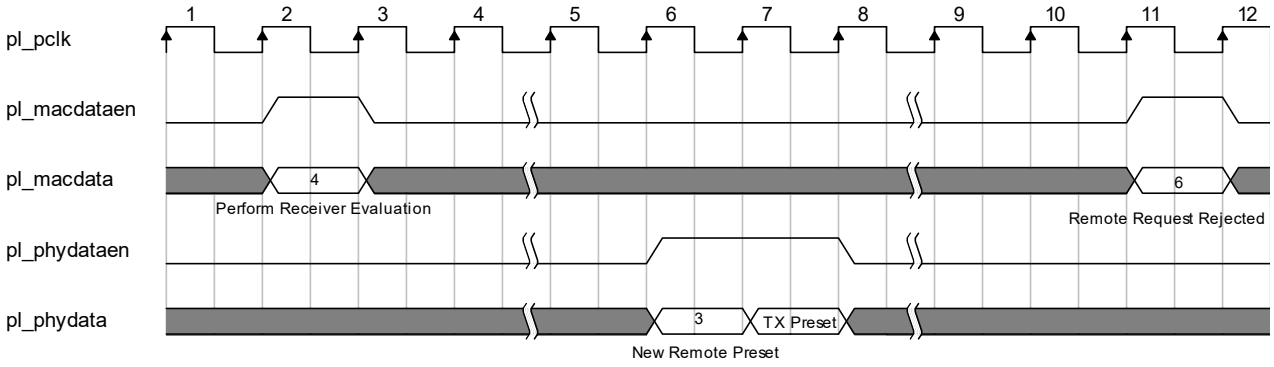


Figure 18: New Remote Preset Request

Note: The Tx Preset value is 4 bits so `PL_PHYDATA[5:4]` must be tied to 0 on the corresponding data phase.

The waveform below shows an evaluation request followed by a New Remote Coefficients Request command:

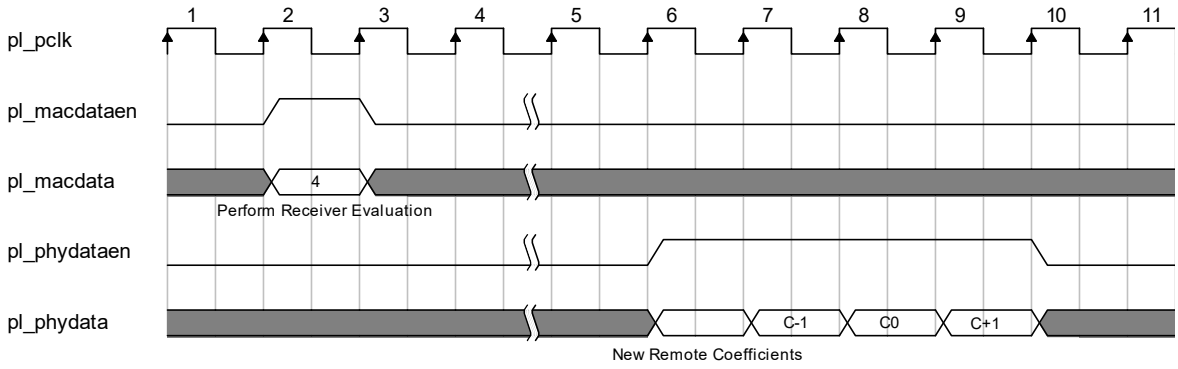


Figure 19: New Remote Coefficients Request

The waveform below shows an evaluation request followed by an Equalization Complete response:

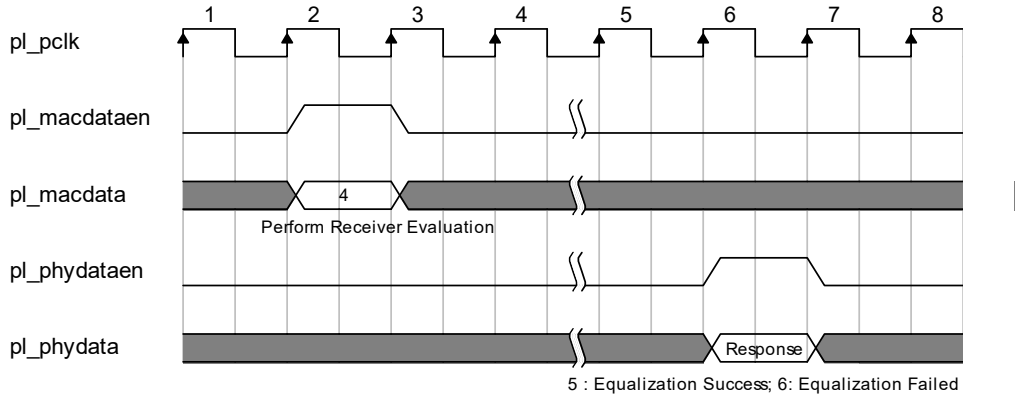


Figure 20: Equalization Complete response

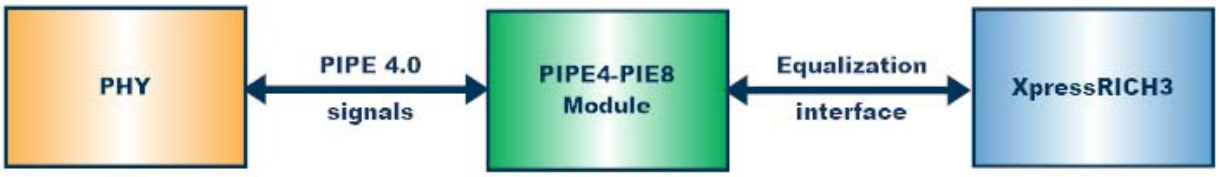
6.3.4 Equalization Problems

When Recovery.ReceiverLock is entered from Recovery.Equalization, if the Core is an upstream port, then it checks the equalization settings advertised by the link partner. If these settings do not match the settings that were accepted during equalization, then the Core reports an equalization problem.

When an equalization problem is detected, the Core automatically moves to Recovery and requests equalization using the Request Equalization bit in the TS2 ordered sets it sends in the Recovery.RcvrCfg state.

6.4 PIPE4-PIE8 Module

You can use the PIPE4-PIE8 module to interface any PIPE 4.0-compliant PHY with the XpressRICH3 equalization interface, as shown below:



6.4.1 XpressRICH3 Equalization Interface

The following table describes the signals used to interface the PIPE4-PIE8 module with the XpressRICH3 equalization interface:

Name	I/O	Width	Description
pl_pclk	in	1	PIPE clock (same as Core)
pl_rstn	in	1	Asynchronous reset (same as Core)
pl_srst	in	1	Synchronous reset (same as Core)
pl_rate	in	2	Rate information from Core
pl_ltssm	in	5	LTSSM state machine from Core
pl_equ_phase	in	2	Equalization phase from Core
pl_txdeemph	in	1	Gen1/2 de-emphasis from Core
pl_macdataen	in	N_L	MAC to PHY command/data enable from Core
pl_macdata	in	6*N_L	MAC to PHY command/data from Core
pl_phydataen	out	N_L	PHY to MAC response/data enable to Core
pl_phydata	out	6*N_L	PHY to MAC response/data to Core

Table 24: XpressRICH3 equalization interface

6.4.2 PHY Interface

The following table describes the signals used to interface the PIPE4-PIE8 module with a PIPE compliant PHY:

Name	I/O	Width	Description
pipe_phystatus	in	N_L	See *PIPE Interface* specification.
pipe_getlocalpresetcoefficients	out	N_L	These signals are used to query coefficients corresponding to a preset from the PHY. If this interface is not present on then PHY, then inputs can be tied to 0's and outputs can be left unconnected. If bit 4 is not present on the PHY, then it can be left unconnected.
pipe_localtxcoefficientsvalid	in	N_L	
pipe_localpresetindex	out	5*N_L	
pipe_localtxpresetcoefficients	in	18*N_L	
pipe_localfs	in	6*N_L	FS setting from PHY
pipe_locallf	in	6*N_L	LF setting from PHY
pipe_rxeqeval	out	N_L	RX Evaluation request
pipe_rxeqinprogress	out	N_L	RX Equalization in progress indicator
pipe_txdeemph	out	N_L	TX de-emphasis/TX coefficients to PHY
pipe_fs	out	6*N_L	FS value received from link partner
pipe_lf	out	6*N_L	LF value received from link partner
pipe_rxpresethint	out	3*N_L	RX Preset Hint value received from link partner
pipe_linkevaluationfeedbackfiguremerit	in	8*N_L	Evaluation result in 'figure of merit' format. Must be tied to 0's if this output is not available on the PHY.
pipe_linkevaluationfeedbackdirectionchange	in	6*N_L	Evaluation result in 'direction change' format. Must be tied to 0's if this output is not available on the PHY. Note that bits [3:2] (corresponding to C_0) are ignored.
pipe_invalidrequest	out	N_L	Indicates that link evaluation feedback resulted in coefficients that were either illegal or rejected by the link partner.

Table 25: PHY interface

6.4.3 Configuration Parameters

These parameters are hardwired or static values that configure the behavior of PIPE4-PIE8 module; they are common to all lanes.

Name	I/O	Width	Description
k_finetune_max_8gt	in	6	**Maximum number of fine-tuning iterations at 8GT**: • 0: Do not fine-tune • 1-63: Fine-tune coefficients using direction-change feedback, limited to the specified number of iterations. The number of iterations should be adjusted so that the remote transmitter adjustment does not continue for too long and that it completes within the 24ms time frame allowed by the *PCI Express Specification*. This setting must be 0 if the PHY does not support evaluation feedback in direction-change format.
k_finetune_err	in	2	Specifies behavior when an error is detected during the fine-tuning process. If PHY gives direction-change instructions that would lead to illegal coefficients then the module does one of the following: • 00: last good coefficients are kept and fine-tuning iterations continue • 01: last good coefficients are kept and fine-tuning iterations are stopped • 10: best preset is reapplied and fine-tuning iterations continue from there (# of fine tuning iterations is reset) • 11: best preset is reapplied and fine-tuning iterations are stopped
k_preset_to_use_8gt	in	11	This signal indicates which preset will be tested in the preset scanning phase at 8GT: • 0: Use preset #0 • 1: Use preset #1 ... • 10: Use preset #10 If all bits are 0 then no preset will be tested. Either no bits or just one bit must be set if the PHY does not support evaluation feedback in 'figure of merit' format.
k_phyparam_query	in	1	Query PHY to get parameters: • 0: Use internal FS/LF/Preset-to-coefficient values (these internal values need to be adjusted depending on the PHY) • 1: Query local FS/LF/Preset-to-coefficient values from the PHY through PIPE interface signals (see Section 6.4.4, below).
k_query_timeout	in	1	Implement timeout for Preset-to-coefficient query: • 0: No timeout (default) • 1: Implement a 256 clock cycles timeout Note that PIPE specification indicates that PHY must respond to a Preset-to-coefficient within 128ns.

Table 26: PIPE4-PIE8 Module Configuration Parameters

6.4.4 Local FS/LF/Preset-to-Coefficient Values

We recommend querying the PHY to get local FS/LF/Preset-to-coefficient values; however some required PIPE interface signals are optional and might not be available. If this is the case, it is possible to use set parameters in the PIPE4-PIE8 module with the desired values:

Name	Size	Description
PHY_LF_8GT	6 bits	PHY LF value at 8GT.
PHY_FS_8GT	6 bits	PHY FS value at 8GT
PRESET_0_8GT	18 bits	Coefficients corresponding to preset 0 - 10 at 8GT. • Bits 5 - 0: C-1 • Bits 11 - 6: C0 • Bits 17 - 12: C+1
...		
PRESET_10_8GT		

Table 27: Local FS/LF/Preset-to-coefficient values

If these values are used then they must match PHY parameters; check the technical documents for your PHY or contact your PHY vendor to get the appropriate parameters.

6.4.5 Remote Transmitter Adjustment

Remote Transmitter Adjustment is executed in equalization phase 2 for upstream devices (such as endpoints) or in phase 3 for downstream devices (such as rootports). During this phase the PIPE4-PIE8 module will request its link partner to change its TX preset/coefficients in order to find optimal settings.

Note: Remote Transmitter Adjustment is not executed by the Core if K_GEN[23]=0. In addition, equalization phases 2/3 are optional and can be skipped by downstream devices.

The PIPE4-PIE8 module handles the Remote Transmitter Adjustment process as follows:

1. If no preset is selected in K_PRESET_TO_USE, then skip to step 3. If only one preset is selected in K_PRESET_TO_USE then request that the link partner uses this preset; then skip to step 3. Otherwise proceed to step 2.
2. **Preset Scan:**
   For each preset specified in K_PRESET_TO_USE: request that the link partner uses this preset, runs RX evaluation and gets 'figure of merit' feedback from the PHY. When all presets have been tested, request that the link partner uses the preset that achieved the highest 'figure of merit' value.
3. If K_FINETUNE=1 proceed to step 4, otherwise skip to step 5.
4. Fine tuning:
   - If the maximum number of iterations specified by K_FINETUNE_MAX has been reached then skip to step 5. If not, then run RX evaluation and get 'direction change' feedback from the PHY.
   - If the PHY requests no change or C-1 and C+1 then skip to step 5.
   - If the PHY gives direction-change instructions that would lead to illegal coefficients then proceed as indicated by k_finetune_err (see Table 26).
   - Otherwise, recalculate coefficient values and request link partner to use new coefficients, then run step 4 again.
5. Adjustment is complete.

Note: Although k_... parameters are common to all lanes, this process is run on each lane independently. As a consequence, the adjustment result may be different on each lane, and adjustment does not necessarily complete on all lanes at the same time.

The following waveforms show a typical equalization sequence for a x2 endpoint when the following PIPE4-PIE8 configuration parameters are set:

- K_FINETUNE=1
- K_FINETUNE_MAX=4
- K_PRESET_TO_USE=00101000010b (meaning that presets #1, #6, and #8 will be tested)

On Lane 0: Preset #6 is determined to be the most efficient. Coefficients are fine-tuned twice, after which no more adjustments are requested by the PHY:

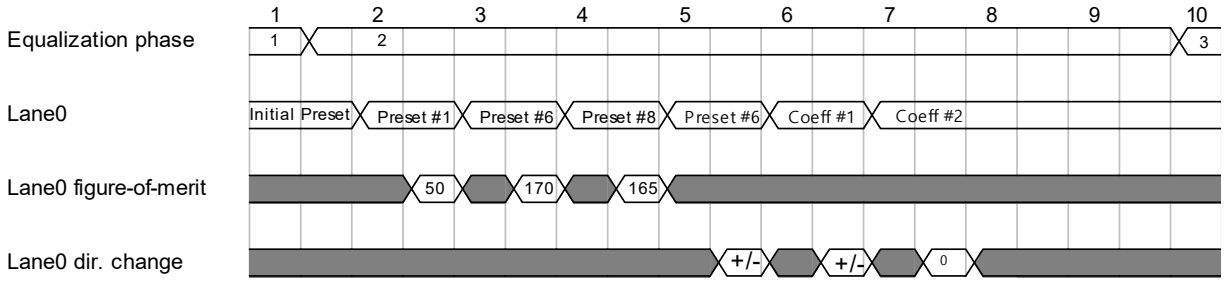


Figure 21: Remote Transmitter Adjustment on Lane 0

On Lane 1: Preset #8 is determined to be the most efficient. Coefficients are fine-tuned four times, after which the maximum number of fine-tuning iterations is reached:

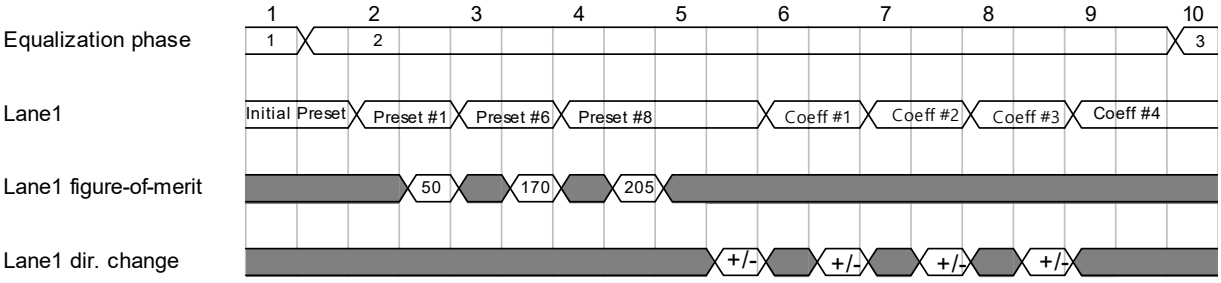


Figure 22: Remote Transmitter Adjustment on Lane 1

Chapter 7 Receive and Transmit Interfaces

7.1 Rx/Tx Latency and Cut-Through

The XpressRICH3 Core Receive and Transmit buffers can operate in two modes:

- Store and Forward
- Rx/Tx Cut-Through

Store and Forward is the default mode. Receive/transmit latency in a PCI Express Core is a function of TLP size; a received TLP must be completely stored in the Rx/Tx buffer before being forwarded to the application's interface or the PIPE interface. This latency is usually not a concern in Endpoint designs, but it is critical for Switches. Store and Forward is illustrated in the waveform below:

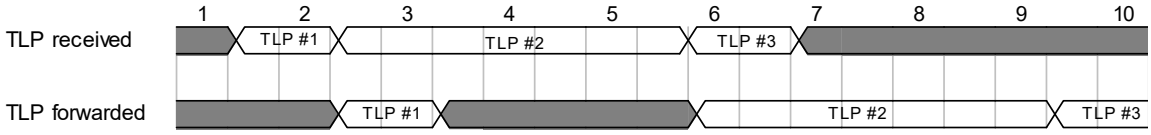


Figure 23: Store and Forward

Rx/Tx Cut-Through allows TLPs to be forwarded to their destination interface before they are completely stored in the Rx/Tx buffer. This enables significant latency reductions for large TLPs. You can enable Rx and Tx cut-through using `K_GEN[21]` and `K_GEN[22]`variables. See `TL_RX_VALID0` and `TL_TX_STREAM` also for information about how cut-throughs are handled in the Core. Rx/Tx Cut-Through is illustrated in the waveform below:

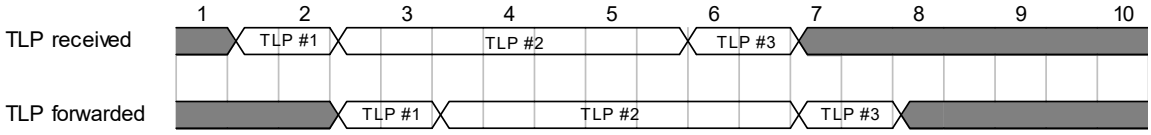


Figure 24: Rx/Tx Cut-Through

Note that the k_pipe value (see Chapter 4), which defines which optional pipeline levels are implemented in the Core, has a significant affect on receive and transmit latency.

The following table shows the latency measured when transmitting or receiving a TLP with a 16/256-byte payload. The latency figures indicate the number of clock cycles between the start of TLP on the PIPE interface and the start of TLP on the application interface:

Link speeds	TX latency without cut-through	TX latency with cut-through	RX latency without cut-through	RX latency with cut-through
x1 - 16 bytes TLP	12	12	20	20
x1 - 256 bytes TLP	18	14	83	27
x2 - 16 bytes TLP	12	12	20	20
x2 - 256 bytes TLP	18	14	52	23
x4 - 16 bytes TLP	12	12	20	20
x4 - 256 bytes TLP	18	15	21	24
x8 - 16 bytes TLP	12	12	18	18
x8 - 256 bytes TLP	18	18	26	20

Table 28: Rx/Tx latency and cut-through

Note that in this example, the PIPE interface is 32-bit and CDC is not enabled, so that all Core logic runs at 62.5MHz in Gen1, 125MHz in Gen2 and 250MHz in Gen3, and the low-latency k_pipe option is used.

7.2 Receive Interface

The Receive Interface allows user applications to receive TLPs from the PCI Express bus.

The Receive Interface can operate in three different modes depending on the type and complexity of the user application:

- Normal Mode
- Normal Mode with Non-Posted Buffer
- RX Stream Mode

7.2.1 Normal Mode

Normal mode is the default receive interface mode that is suitable for most applications with minimum complexity.

In this mode:

- the Receive buffer is completely implemented inside the Core and is large enough to store all received TLPs.
- All credit updates are handled internally.
- Only TLPs intended for the user application are output on the Receive interface, in the order they are received from the link.

The following diagram shows normal mode operation:

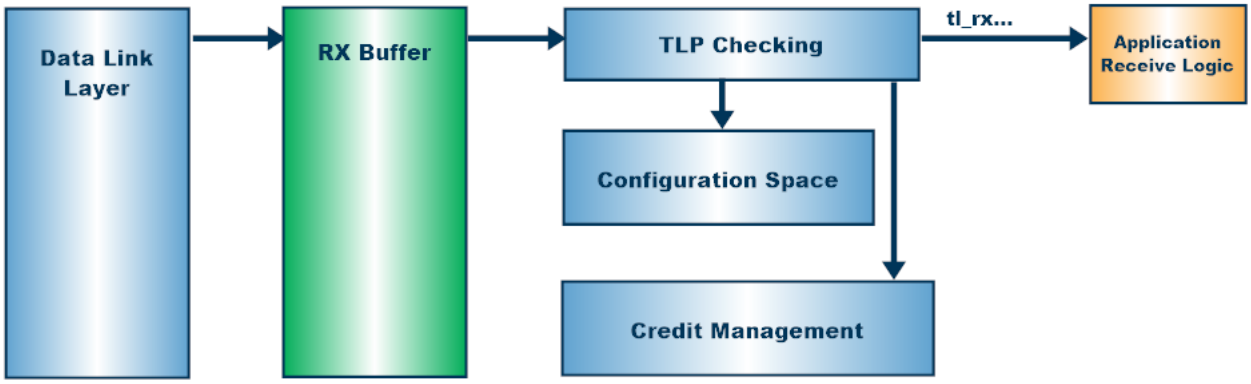


Figure 25: Normal Mode

7.2.2 Normal Mode with Non-Posted Buffer

This mode is similar to normal mode except that an additional buffer is implemented inside the Core that can be used to temporarily store non-posted TLPs. This enables the application to continue receiving posted and completion TLPs even when it is not ready to process non-posted TLPs.

The non-posted buffer must be large enough to store all received non-posted TLPs in order to avoid deadlocks.

The following diagram shows normal mode with non-posted buffer operation:

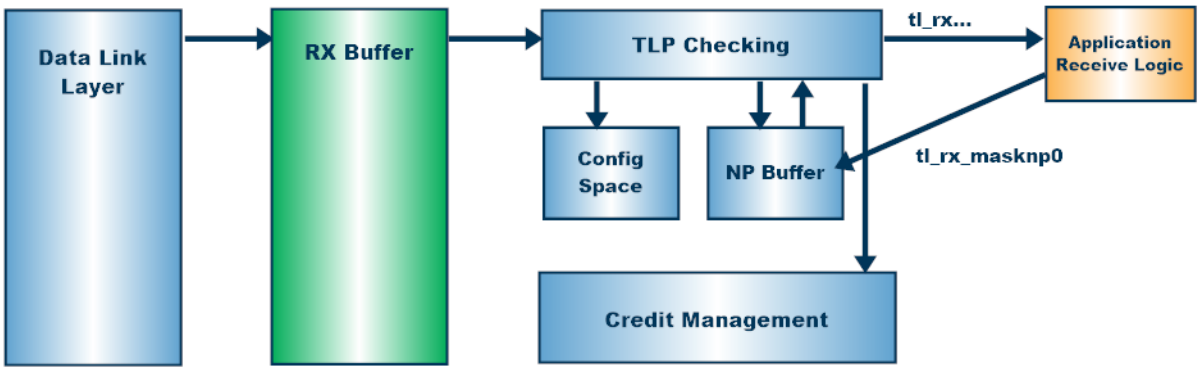


Figure 26: Normal Mode with Non-Posted Buffer

7.2.3 RX Stream Mode

In RX stream mode, the Core does not store received TLPs internally but outputs them to the user application immediately. This mode enables greater control on TLP ordering and processing and is particularly suitable for Bridge and Switch designs.

The Core contains only a very small receive buffer; the application must implement storage that is large enough to store all received TLPs. It must also handle all credit updates for the TLPs it receives using the `TL_RX_CRED.` signals.

The application can implement a single buffer to receive all TLPs or several buffers; one per TLP type, for example. In each case TLP ordering must be handled carefully.

A non-posted buffer cannot be used in RX stream mode.

The following diagram shows RX stream mode operation:

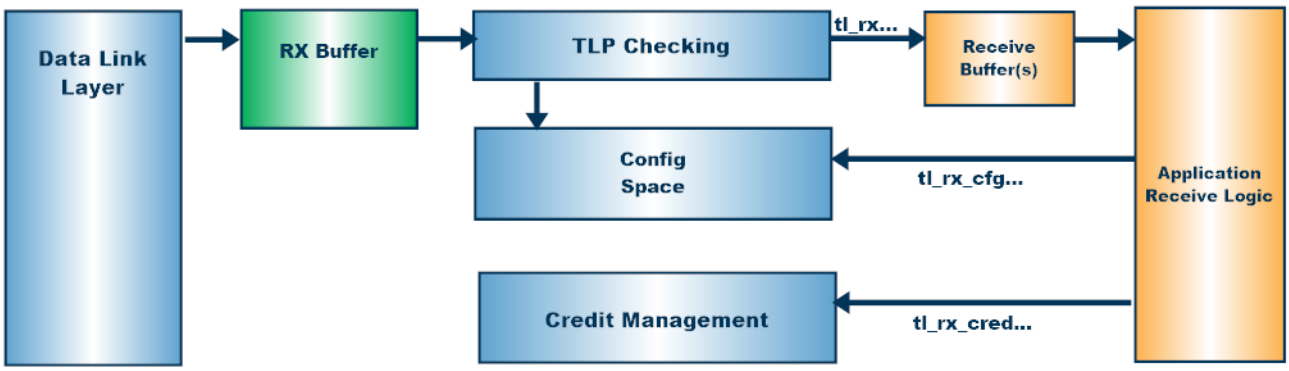


Figure 27: RX Stream Mode

In RX Stream mode, the Core implements only a minimal buffer, which means that the application must always be able to accept received TLPs. As a consequence:

- the application is not allowed to insert wait-states with `TL_RX_WAIT0`.
- RX cut-through must be enabled.
- The transaction layer clock must run at a minimum frequency in order to sustain throughput.

7.2.3.1 Clock Frequencies in RX Stream Mode

The transaction layer clock minimum frequency depends on link width and speed, as shown in the table below:

Link Width	Min tl_clk frequency at 2.5 GT/s (MHz)	Min tl_clk frequency at 5.0 GT/s (MHz)	Min tl_clk frequency at 8.0 GT/s (MHz)
x1	12.5	25	50
x2	25	50	100
x4	50	100	200
x8	100	200	400
x16	200	400	800

Table 29: Minimum tl_clk frequencies in RX Stream mode with 256-bit TL datapath

Note: The values in this table are applicable when the TL datapath is 256-bit; these values must be divided by 2 if the TL datapath is 512-bit.

These requirements are calculated to avoid overflow, based on the worst case scenario (a series of TLPs without data).

7.2.3.2 RX Stream Watchdog

Although it is not recommended, it may be necessary in some implementations to run TL_CLK at a frequency lower than the minimum frequency required in RX stream mode.

RX stream watchdog is an optional feature that stops incoming data flow when the Receive buffer is nearly full (this level is programmable using K_GEN[55:52]) and resumes normal operation when the buffer is sufficiently empty (this level is programmable using K_GEN[51:48]) This can avoid RX buffer overflow and, consequently, data corruption, however, this feature is not compliant with the PCIe specification.

The RX stream watchdog is enabled by setting K_GEN[35]=1, but it also requires G_RXBUF_SIZE>=10

When the watchdog is active, then the first TLP is NAK'd and the following TLPs are silently discarded (no ACK or NAK sent). Depending on the core pipeline options, several writes to the RX buffer can occur before data flow is stopped. The application may receive one or more TLP(s) with TL_TX_ERR0[1]=1 and must discard them silently. No TLPs are lost, however, as the link partner eventually replays from the last ACK'd TLP and data processing resumes normally.

7.2.4 TLP Processing

The following table shows how TLPs are processed in each of the different modes:

TLP Type	Normal Mode	RX Stream Mode
Erroneous TLP	Core discards TLP.	Core discards TLP.
Unsupported TLP (NP)	Core discards TLP & sends CPL UR.	TLP is sent to application, which must either send it to configuration space*; or discard it and send CPL UR*.
Unsupported TLP (P/CPL)	Core discards TLP.	Core discards TLP.
ACS Violation (NP)	Core discards TLP & sends CPL CA.	TLP is sent to application, which must either send it to configuration space*; or discard it and send CPL CA*.
ACS Violation (P/CPL)	Core discards TLP.	Core discards TLP.
Unexpected CPL	Core discards TLP.	Core discards TLP.
Config Type0	Core consumes TLP & sends CPL/ CPLD.	TLP is sent to application, which must send it to configuration space*.
Msg (P) handled by the Core itself	Core consumes TLP.	Core consumes TLP.
Other TLPs	TLP is sent to application; processing is application-specific.	TLP is sent to application; processing is application-specific*.

Figure 28: TLP Processing

The Core updates credits for TLPs it consumes/discards internally; the application must update credits for TLPs it receives*.

*If the application receives a nullified or malformed TLP then it must discard it without any processing (no credit update).

7.2.5 Receive Interface Signals

The following table describes all the signals that can be used for the Receive interface. Note that some signals can only be used in RX Stream Mode; these signals are not available in normal mode.

<table>
<tr><th>Signal</th><th>I/O</th><th>Description</th></tr>
<tr><td>tl_rx_sop0</td><td>Out</td><td>Receive Start of TLP: This signal is asserted when the first data phase of a TLP is available on TL_RX_DATA0.</td></tr>
<tr><td>tl_rx_eop0</td><td>Out</td><td>Receive End of TLP: This signal is asserted on the last data phase of the TLP transfer.</td></tr>
<tr><td>tl_rx_data0[255:0] (with a 256-bit TL datapath) or [511:0] (with a 512-bit TL datapath)</td><td>Out</td><td>Receive Data: This signal contains the TLP End-End prefixes (if present), header and payload, and optionally ECRC. Note that the Core optionally checks ECRC, but does not strip ECRC from the TLP before passing it to the application. The End-End prefixes and TLP header are always present on the first data phase.
Example of DWORD ordering on the first data phase with a 256-bit datapath:
<table>
<tr><th>Bits</th><th>TLP with 3DW header and 4DW Data</th><th>TLP with 4DW header, 1 DW Data and ECRC</th><th>TLP with 2 End-End prefixes, and 4 DW header</th></tr>
<tr><td>255:224</td><td>Header DW 0</td><td>Header DW 0</td><td>EE Prefix 0</td></tr>
<tr><td>223:192</td><td>Header DW 1</td><td>Header DW 1</td><td>EE Prefix 1</td></tr>
<tr><td>191:160</td><td>Header DW 2</td><td>Header DW 2</td><td>Header DW 0</td></tr>
<tr><td>159:128</td><td>Data 0</td><td>Header DW 3</td><td>Header DW 1</td></tr>
<tr><td>127:96</td><td>Data 1</td><td>Data 0</td><td>Header DW 2</td></tr>
<tr><td>95:64</td><td>Data 2</td><td>ECRC</td><td>Header DW 3</td></tr>
<tr><td>63:32</td><td>Data 3</td><td>–</td><td>–</td></tr>
<tr><td>31:0</td><td>–</td><td>–</td><td>–</td></tr>
</table></td></tr>
<tr><td>tl_rx_valid0[7:0] (with a 256-bit TL datapath) or [15:0] (with a 512-bit TL datapath)</td><td>Out</td><td>• Receive Data Valid: This signal indicates which Dwords in TL_RX_DATA0 are valid. Bit n indicates that TL_RX_DATA0[n x 32 + 31:n x 32] is valid.
• This signal is 0’s when the Core is not ready to transfer data (this cannot happen when a TLP is being received, except when receive cut-through is enabled and throughput adjustments are necessary).
• This signal is 1's when a TLP is being received, except on the last data phase where it indicates which Dwords are valid (note that MSB bit is always 1 when a TLP is being received). The last valid DWORD of a TLP is either the last DWORD of a header, the last DWORD of data payload, or ECRC.</td></tr>
</table>

Table 30: Receive Interface Signals

<table>
<tr><th>Signal</th><th>I/O</th><th>Description</th></tr>
<tr><td>tl_rx_prot0</td><td>Out</td><td>Receive data protection bits: When data protection is implemented, this signal contains the data protection bits for the data received on TL_RX_DATA0 on the same clock cycle. Note that parity is not be checked for 32-bit words that are not valid as indicated by TL_RX_VALID0 on the same clock cycle. See Chapter 8 for more information.
• If G_DATA_PROT=0: This signal is not used
• If G_DATA_PROT=1:
• bit 0: parity for TL_RX_DATA0 [31:0]
• bit 3:1: unused
• bit 4: parity for TL_RX_DATA0 [63:32]
• bit 7:5: unused
• bit 28: parity for TL_RX_DATA0[255:224]
• bit 31:29: unused
• If G_DATA_PROT=4:
• bit 0: parity for TL_RX_DATA0 [7:0]
• bit 1: parity for TL_RX_DATA0 [15:8],
• ...
• bit 31: parity for TL_RX_DATA0[255:248]</td></tr>
<tr><td>tl_rx_bardec0[30:0]</td><td>Out</td><td>Receive BAR Decoding:
This signal indicates which function is targeted when a unicast TLP is received and provides decoding information. Decoding information depends whether the targeted function is Type 0 or Type 1, and on the type of received TLP. This signal is valid for the entire time a TLP is present on the TL_RX_... interface.

If TLP is routed by address (MemWr/MemRd/IO):
<table>
<tr><th>Bit</th><th>Target Function is Type 0</th><th>Target Function is Type 1</th></tr>
<tr><td>0</td><td colspan="2">BAR 0</td></tr>
<tr><td>1</td><td colspan="2">BAR 1 (reserved when BAR0 is 64-bit)</td></tr>
<tr><td>2</td><td>BAR 2</td><td>reserved</td></tr>
<tr><td>3</td><td>BAR 3
(reserved when BAR2 is 64-bit)</td><td>reserved</td></tr>
<tr><td>4</td><td>BAR 4</td><td>reserved</td></tr>
<tr><td>5</td><td>BAR 5
(reserved when BAR4 is 64-bit)</td><td>I/O window</td></tr>
<tr><td>6</td><td>Expansion ROM</td><td>Memory window</td></tr>
<tr><td>7</td><td>reserved</td><td>Prefetchable memory window</td></tr>
<tr><td>10:8</td><td colspan="2">PF#: If a PF is targeted then these bits indicate the PF#; if a VF is targeted then these bits indicate the PF it is associated with.</td></tr>
<tr><td>20:11</td><td>FID (used only if VFs are implemented, see Chapter 12)</td><td>reserved</td></tr>
<tr><td>30:21</td><td colspan="2">VF#: If 0, then a physical function is targeted.
Reserved if no virtual functions are implemented.</td></tr>
</table></td></tr>
</table>

Table 30: Receive Interface Signals

<table>
<tr><th>Signal</th><th>I/O</th><th>Description</th></tr>
<tr><td>tl_rx_bardec0[30:0]
(continued)</td><td>Out</td><td>If TLP is routed by ID (Msg/Cpl/Cfg):
<table>
<tr><th>Bit</th><th>Target Function is Type 0</th><th>Target Function is Type 1</th></tr>
<tr><td>0</td><td colspan="2">reserved</td></tr>
<tr><td>1</td><td colspan="2">reserved</td></tr>
<tr><td>2</td><td>reserved</td><td>Bus# = Primary Bus#</td></tr>
<tr><td>3</td><td>reserved</td><td>Bus# = Secondary Bus#</td></tr>
<tr><td>4</td><td>reserved</td><td>Upstream:
Primary Bus# <= Bus# <= Subordinate Bus#
Downstream:
Bus# <Secondary Bus# or Bus# > Sub. Bus#</td></tr>
<tr><td>5</td><td colspan="2">reserved</td></tr>
<tr><td>6</td><td colspan="2">reserved</td></tr>
<tr><td>7</td><td colspan="2">reserved</td></tr>
<tr><td>10:8</td><td colspan="2">PF#: If a PF is targeted then these bits indicate the PF#; if a VF is targeted then these bits indicate the PF it is associated with.</td></tr>
<tr><td>20:11</td><td>FID (used only if VFs are implemented, see Chapter 12)</td><td>reserved</td></tr>
<tr><td>30:21</td><td colspan="2">VF#: If 0, then a physical function is targeted.
Reserved if no virtual functions are implemented.</td></tr>
</table></td></tr>
<tr><td>tl_rx_mchit0[15:0]</td><td>Out</td><td>Multicast Function Hit Information: This signal indicates which physical functions are the recipient of a Multicast TLP and what the Multicast Group is when TL_RX_MCHIT[15] is asserted. It is valid for the entire time a TLP is present on the TL_RX_... interface.
• Bit 0: Function 0 is targeted
• ...
• Bit 7: Function 7 is targeted
• Bits 13:8: Multicast Group
• Bit 14: Multicast TLP is blocked by function #0 (Rootport & switch only)
• Bit 15: Multicast hit
Note:
• Bits 14:0 are not applicable when TL_RX_BARDEC[15] is not asserted
• Any number of function can be targeted simultaneously, and it is also possible for no function to be targeted at all.</td></tr>
</table>

Table 30: Receive Interface Signals

Signal	I/O	Description

tl_rx_err0[7:0]	Out	• **Received Error**:
		• Bit 0: ECRC Error: the calculated ECRC does not match the ECRC present in the TLP, so the application should take any appropriate action. This error is always reported in the last dataphase of the TLP (when TL_RX_EOP0 is asserted).
		• Bit 1: Invalid TLP: This error occur can happen if:
		  • a TLP is nullified or has been marked as incorrect in the receive buffer (RX cut-through mode only)
		  • the TLP payload size does not match information in the header (all modes)
		The application must discard this TLP, and in RX stream mode it must not update credits for this TLP. This error is always reported in the last dataphase of the TLP (when TL_RX_EOP0 is asserted).
		• Bit 2: Receive Buffer Uncorrectable Read Error: this bit is asserted when an uncorrectable error has been detected by the memory's ECC logic when reading data from the Receive buffer. The error is reported on the same clock cycle as the affected data. See Section 2.6 for more information.
		• Bit 3: Direct to Config (RX stream mode only): this bit is asserted in order to inform the application that the TLP must be directed to the TL_RX_CFG... interface to be processed by the Core. This information is reported for the duration of the TLP.
		• Bit 4: Unsupported TLP (RX stream mode only): this bit is asserted when a TLP is an unsupported Non-Posted request. In this case, the application must either:
		  • discard the TLP and send a CPL UR, or
		  • direct the TLP to tl_rx_cfg... so that the Core can discard it and send a CPL UR.
		This information is reported for the duration of the TLP.
		• Bit 5: ACS Violation (RX stream mode only): this bit is asserted when a TLP is an Non-Posted request and an ACS violation has been detected. In this case, the application must either:
		  • discard the TLP and send a CPL CA, or
		  • direct the TLP to tl_rx_cfg... so that the Core can discard it and send a CPL CA.
		This information is reported for the duration of the TLP.
		• Bits 7:6: reserved

tl_rx_masknp0	In	**Mask Received Non-Posted TLPs**: This signal is asserted by the user application before the end of a TLP in order to indicate that it will not be able to accept any further non-posted TLPs while this signal is asserted. This enables the Core to present posted TLPs and completions. The Core takes this signal into account either when no TLP is transmitted or on the last data phase of the currently transmitted TLP. Note that most non-posted TLPs are only one data-phase so the application might need to throttle transfers in order to have time to detect the TLP type and to check if TL_RX_MASKNP0 needs to be asserted. **Note:** This signal must be tied to 0 when Rx reordering is not implemented (G_NPBUF_SIZE = 0).
tl_rx_wait0	In	**Receive Wait**: This signal is used in normal mode by the user application, which asserts it to indicate that it is not ready to accept data. This signal can be asserted at any time while a TLP is being received; no data is thus transferred and the values of TL_RX_DATA0, TL_RX_VALID0, TL_RX_SOP0, and TL_RX_EOP0 do not change until this signal is deasserted. In RX stream mode, this signal cannot be used and must by tied to 0.
tl_rx_cfgdata0[255:0]	In	**Configuration TLP Data**: This signal is used in RX stream mode. It contains the TLP End-End prefixes and header to direct to the Core configuration space when TL_RX_ERR[3] is set. It has the same layout as TL_RX_DATA0. **Note**: In normal mode, this signal is not used and must be tied to 0's.

Table 30: Receive Interface Signals

Signal	I/O	Description
tl_rx_cfgreq0[17:0]	In	**Configuration TLP Request**: This signal is used in RX stream mode to request a TLP to be directed to the Core configuration space when TL_RX_ERR[3] is set. • Bit 0: Request • Bit 1: TLP is unsupported (copied from TL_RX_ERR[4]) • Bit 2: ECRC error was detected in TLP (copied from TL_RX_ERR[0]) • Bit 3: TLP is nullified (copied from TL_RX_ERR[1]) • Bit 4: ACS violation was detected with TLP (copied from TL_RX_ERR[5]) • Bits 7:5: PF#: If a VF is targeted then this indicates the PF it is associated with Bits 17:8: FID (used only if VFs are implemented, see Chapter 12)
tl_rx_cfgack0	Out	**Configuration TLP Acknowledge** (RX stream mode only): This output pulses for one clock cycle to indicate that the TLP was accepted.
tl_rx_creddsize0[26:0]	In	**Number of Data Credits to Release:** This signal is used in RX stream mode to indicate how many data credits must be released by the Core. It is only taken into account when one or more bits of TL_RX_CREDUPDATE0 are asserted. • Bits 8:0: Non-Posted data credits, • Bits 17:9: Posted data credits, • Bits 26:18: Completion data credits **Note**: In normal mode, this signal is not used and must be tied to 0's.
tl_rx_credupdate0[2:0]	In	**Update Credits**: This signal is used in RX stream mode to request that the Core release credits for received TLPs when a TLP is consumed by the user application. Bit(s) must be asserted for one clock cycle to release credits for the specified type. • Bit 0: NP • Bit 1: P, • Bit 2: CPL Several bits can be asserted at the same time making it possible to update NP/P/CPL independently. **Note**: In normal mode, this signal is not used and must be tied to 0's.
tl_rx_credhsize0[14:0]	In	**Number of Header Credits to Release**: This signal is used in RX stream mode to indicate how many header credits must be released by the Core. It is only taken into account when one or more bits of TL_RX_CREDUPDATE0 are asserted. • Bits 4:0: Non-Posted header credits, • Bits 9:5: Posted header credits, • Bits 14:10: Completion header credits **Note**: In normal mode, this signal is not used and must be tied to 0's.

Table 30: Receive Interface Signals

7.2.6 Normal and RX Stream Mode Waveforms

The waveform below illustrates a TLP received in normal mode on a 256-bit datapath with a 4 DWORD header and 15 dwords of payload that targets BAR0 of function 2. The user application throttles the transfer with `TL_RX_WAIT0`:

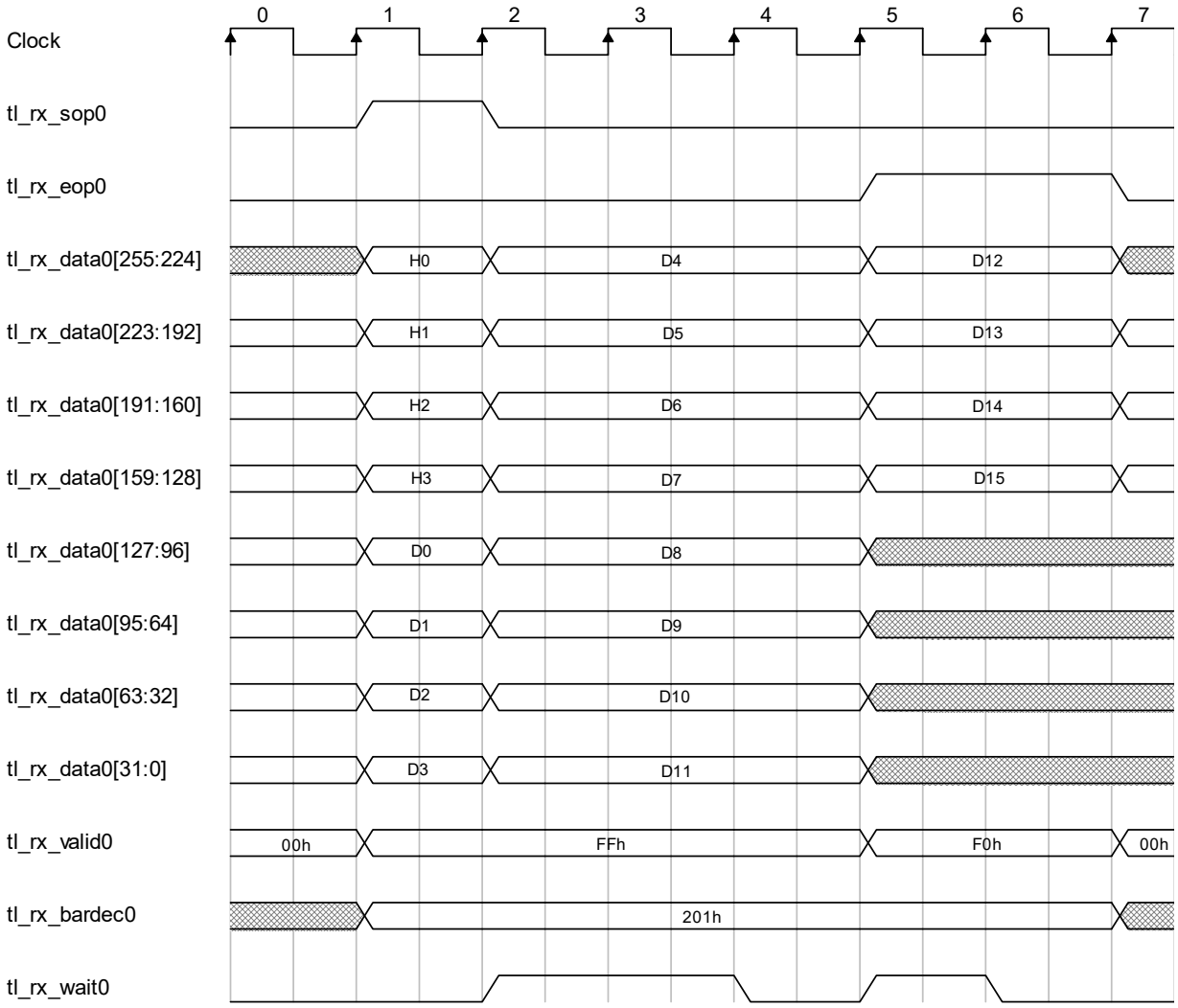


Figure 29: TLP received in normal mode

The following waveform illustrates a non-posted TLP received by the application in normal mode on a 256-bit datapath. The application waits for one clock cycle; then accepts the TLP and asserts TL_RX_MASKNP0 to indicate that it will not accept another non-posted TLP. The Core then outputs a completion TLP, and presents another non-posted TLP only after the application has deasserted TL_RX_MASKNP0:

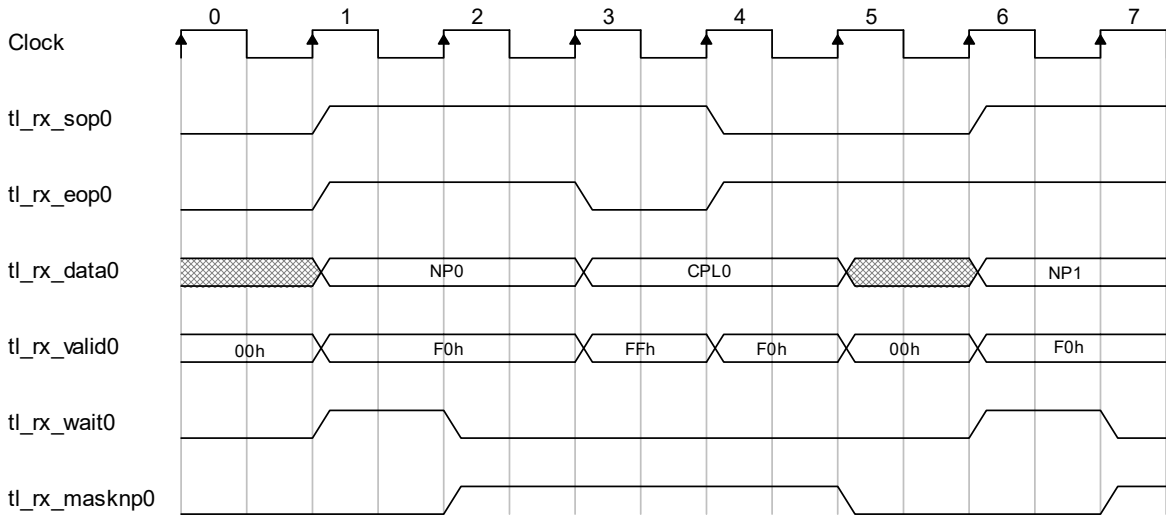


Figure 30: Non-posted TLP received by application (normal mode)

The following waveform illustrates credit update in RX stream mode. The posted TLP is received by the application; when the application consumes the TLP (i.e. it is completely read from the receive buffer) then it releases one posted TLP with two data credits.

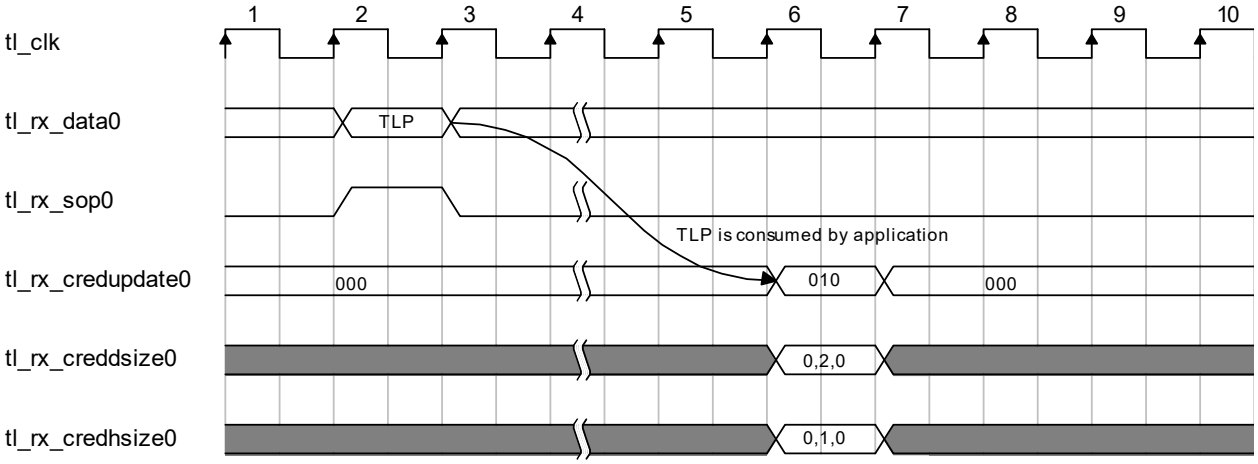


Figure 31: Credit update in RX Stream Mode

The following waveform illustrates a TLP forwarded to the configuration space in RX stream mode.

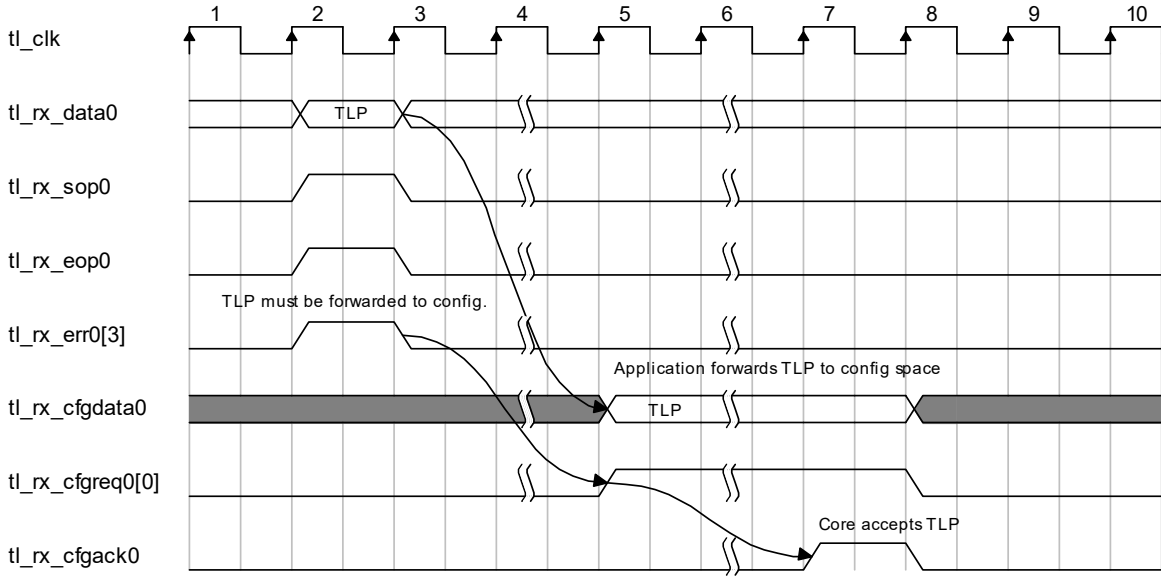


Figure 32: TLP forwarded to config space (RX Stream mode)

The following waveform illustrates how an application releases credits in the RX stream:

- On clock cycle #2, the application releases 6 Completion Header credits + 24 Completion Data credits
- On clock cycle #4, the application releases 1 Posted Header credit + 8 Posted Data credits
- On clock cycle #7, the application simultaneously releases 1 Non-Posted Header credit + 1 Non-Posted Data credit and 1 Completion Header credit.

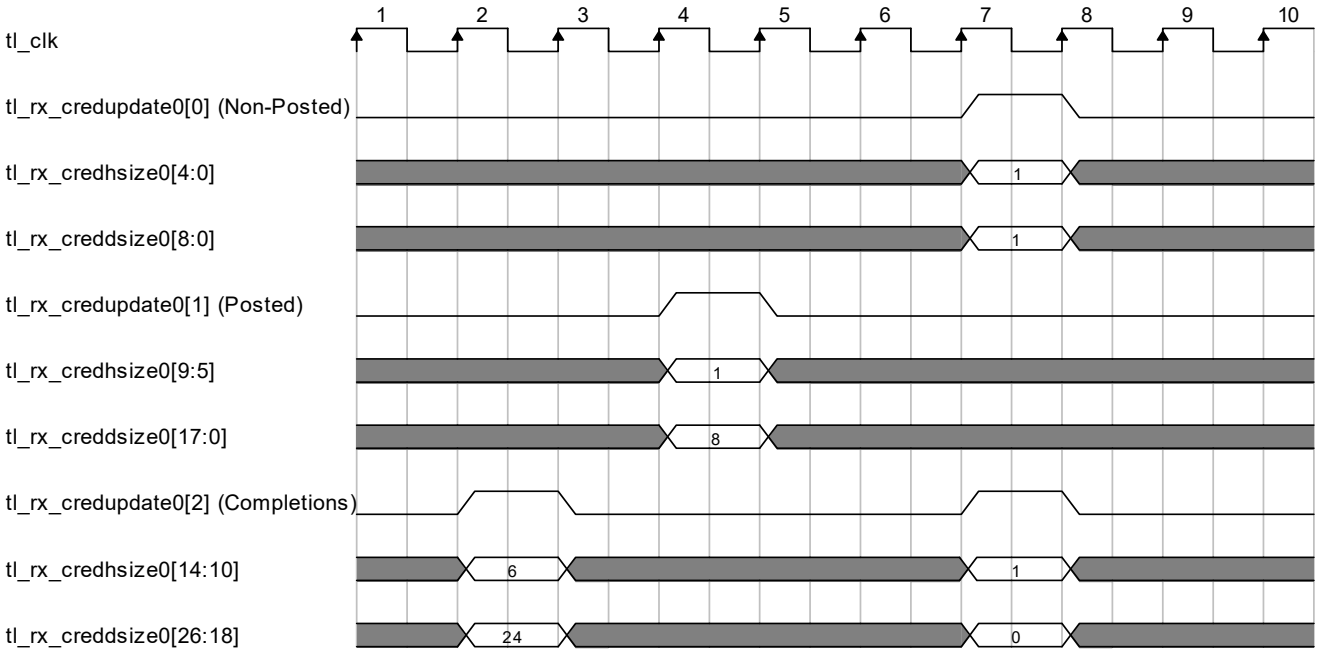


Figure 33: Application releases multiple credits in RX Stream mode

7.3 Transmit Interface

The Transmit interface allows the user application to send TLPs to the PCI Express bus.

Signal	I/O	Description
tl_tx_sop0	In	**Transmit Start of TLP**: This signal is asserted when the first data phase of a TLP is available on TL_TX_DATA0. The application must not wait until TL_TX_WAIT0 is de-asserted before starting a transfer (see Figure 34).
tl_tx_eop0	In	**Transmit End of TLP**: This signal is asserted on the last data phase of the TLP transfer.
tl_tx_data0[255:0] (with a 256-bit TL datapath) or [511:0] (with a 512-bit TL datapath)	In	**Transmit Data**: This signal contains the TLP End-End prefixes (if present), header and payload, and optionally ECRC (see Section 7.3.1). The End-End prefixes and TLP header are always present on the first data phase.
tl_tx_valid0[7:0] (with a 256-bit TL datapath) or [15:0] (with a 512-bit TL datapath)	In	**Transmit Data Valid**: This signal indicates which Dwords in TL_TX_DATA0 are valid. Bit *n* indicates that TL_TX_DATA0[n x 32 + 31:n x 32] is valid. • This signal must be 0's when the user application is not ready to transfer data (this must not happen when a TLP is being transmitted if TL_TX_STREAM0 is asserted). • This signal is 1's when data is being transmitted, except on the last data phase where it indicates which Dwords are valid.

tl_tx_prot0[31:0]	In	Transmit data protection bits: When data protection is implemented, this signal contains the data protection bits for the data transmitted on TL_TX_DATA0 on the same clock cycle. Note that parity is not provided for 32-bit words that are not valid as indicated by TL_TX_VALID0 on the same clock cycle. See Chapter 8 for more information.
		• If G_DATA_PROT=0: This signal is not used
		• If G_DATA_PROT=1:
		  • bit 0: parity for TL_TX_DATA0 [31:0]
		  • bit 3:1: unused
		  • bit 4: parity for TL_TX_DATA0 [63:32]
		  • bit 7:5: unused
		  • bit 28: parity for TL_TX_DATA0[255:224]
		  • bit 31:29: unused
		• If G_DATA_PROT=4:
		  • bit 0: parity for TL_TX_DATA0 [7:0]
		  • bit 1: parity for TL_TX_DATA0 [15:8],
		  • ...
		  • bit 31: parity for TL_TX_DATA0[255:248]

tl_tx_stream0	In	**Transmit Stream Mode**: This signal is asserted by the user application when TL_TX_SOP0 is asserted in order to indicate that it will not throttle the transfer. This enables the PCIe Core to bypass the transmit interface (when possible) and reduce transmit latency. Note that Tx streaming is automatically disabled if: • transmit cut-through is not enabled (k_gen[22]=0) • there is not enough space in transmit buffer for a complete TLP • transmit interface throughput is higher than the link throughput

Table 31: Transmit Interface Signals

Signal	I/O	Description

tl_tx_err0[7:0]	In	**Transmit Nullify**: This signal can be asserted on the last data-phase (at the same time as TL_TX_EOP0) in order to indicate that TLP contains an error:
		• Bit 0: Nullify/Discard TLP:
		  • If TX streaming is enabled for this TLP then the TLP is transmitted and nullified.
		  • If TX streaming is not enabled, then this TLP is removed from TX buffer and thus not transmitted.
		Note: In 2.5/5.0 GT/s mode, you can signal an end of TLP and set this bit earlier than expected. However, in 8.0 GT/s mode, the TLP length must be as specified in the TLP header.
		• TLP must be nullified (switch only). In 2.5/5.0 GT/s mode, you can signal end of TLP and set this bit earlier than expected, however in 8.0 GT/s mode, TLP length must be as indicated in the TLP header.
		• Bit 1: Generate LCRC error
		• Bit 2: Invert ECRC generated by the Core
		• Bit 3: Force ECRC generation in Automatic Mode (K_GEN[31]=0) (see Section 7.3.1)
		• Bits 7:4: reserved

tl_tx_wait0	Out	**Transmit Wait**: This signal is asserted by the Core to indicate that it is not ready to accept data. This signal can be asserted at any time while a TLP is being received; no data is thus transferred. If a transfer is in progress, then the values of TL_TX_SOP0, TL_TX_EOP0, TL_TX_VALID0, and TL_TX_DATA0 cannot change until TL_TX_WAIT0 is de-asserted. See Figure 37 for an illustration of this situation. If a transfer is not in progress, then the application can change tTL_TX_SOP0, TL_TX_EOP0, TL_TX_VALID0, and TL_TX_DATA0 while TL_TX_WAIT0 is asserted in order to start a new transfer. See Figure 34 for an illustration of this situation. The Core can assert TL_TX_WAIT0 when: • the data link is not up • it is preparing to enter ASPM L0s/L1 • it is in legacy D1/D2/D3 low-power mode • the Transmit buffer is full • it is checking credits or when there is not enough credits available to transmit the current TLP • it is busy transmitting a TLP from an internal source • there are not enough credits to transmit any TLP of any type between TLP transfers

tl_tx_credits0[95:0]	Out	**Transmit Credits**: This signal informs the application of the number of available flow control credits.
		• **tl_tx_credits0[31:0]: Posted credits**:
		  • [14:0]: Available data credits (0 if infinite)
		  • [15]: Data credits are infinite
		  • [26:16]: Available header credits (0 if infinite)
		  • [27]: Header credits are infinite
		  • [30:28]: reserved
		  • [31]: There are enough credits to transmit a TLP with payload of maximum size (for Posted & Completion this is based on maximum payload size; for Non-Posted this is 1 data credit, or 2 if device supports requesting CAS 128bit TLPs)
		• **tl_tx_credits0[63:32]: Non-Posted credits**: same as for Posted credits
		• **tl_tx_credits0[95:64]: Completion credits**: same as for Posted credits
		When the application transmits a TLP, the corresponding credits are decremented from tl_tx_credits two clock cycles after the header of this TLP has been accepted. If this TLP is nullified by the application, then the corresponding credits are reinstated two clock cycles after the last dataphase of this TLP.

Table 31: Transmit Interface Signals

The following waveform illustrates a situation where the Core asserts TL_TX_WAIT0 between TLPs because there are not enough credits to transfer a TLP.

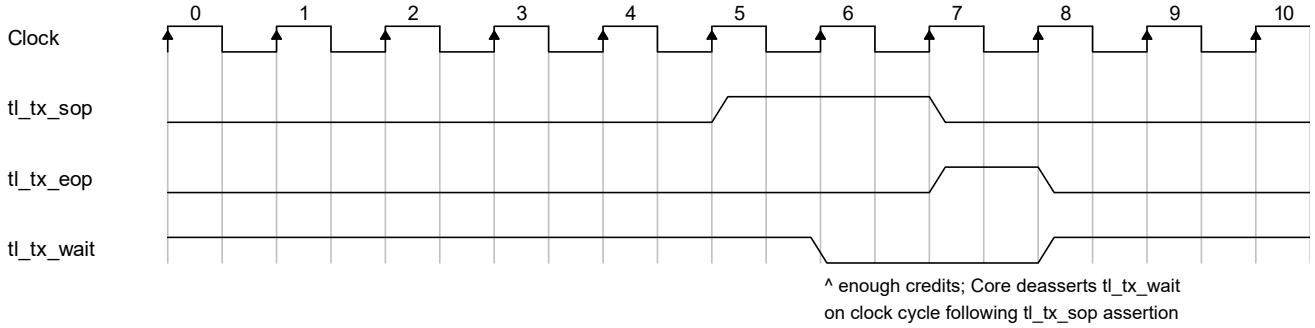


Figure 34: tl_tx_wait asserted between TLPs

The following waveform illustrates a situation where the Core keeps TL_TX_WAIT0 asserted until there are enough credits available to transmit a TLP.

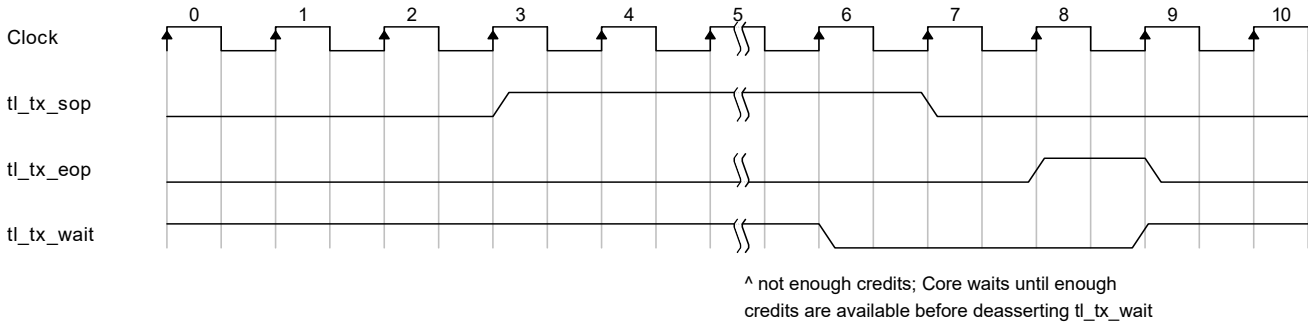


Figure 35: tl_tx_wait kept asserted

The following waveform illustrates a situation where the Core asserts tl_tx_wait for one clock cycle after single-dataphase TLPs (4 single-dataphase TLPs are sent by the application).

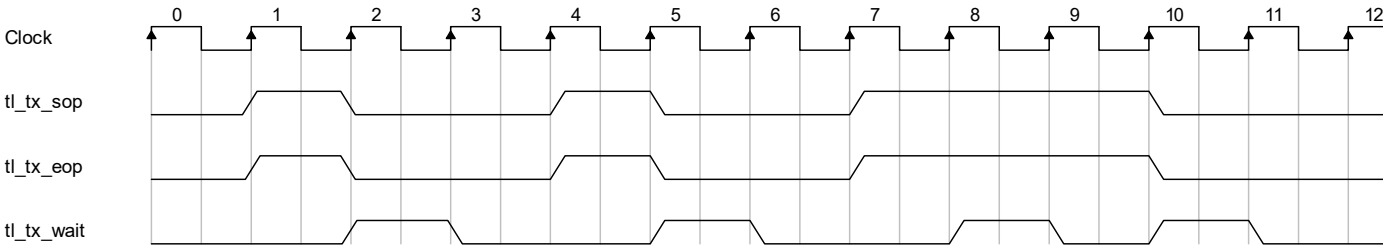


Figure 36: tl_tx_wait asserted for one clock cycle

The following waveform illustrates a TLP transmitted with a 3 DWORD header and 27 Dwords of payload on a 256-bit datapath. Both the user application and the Core throttle the transfer with TL_RX_VALID0 and TL_TX_WAIT0

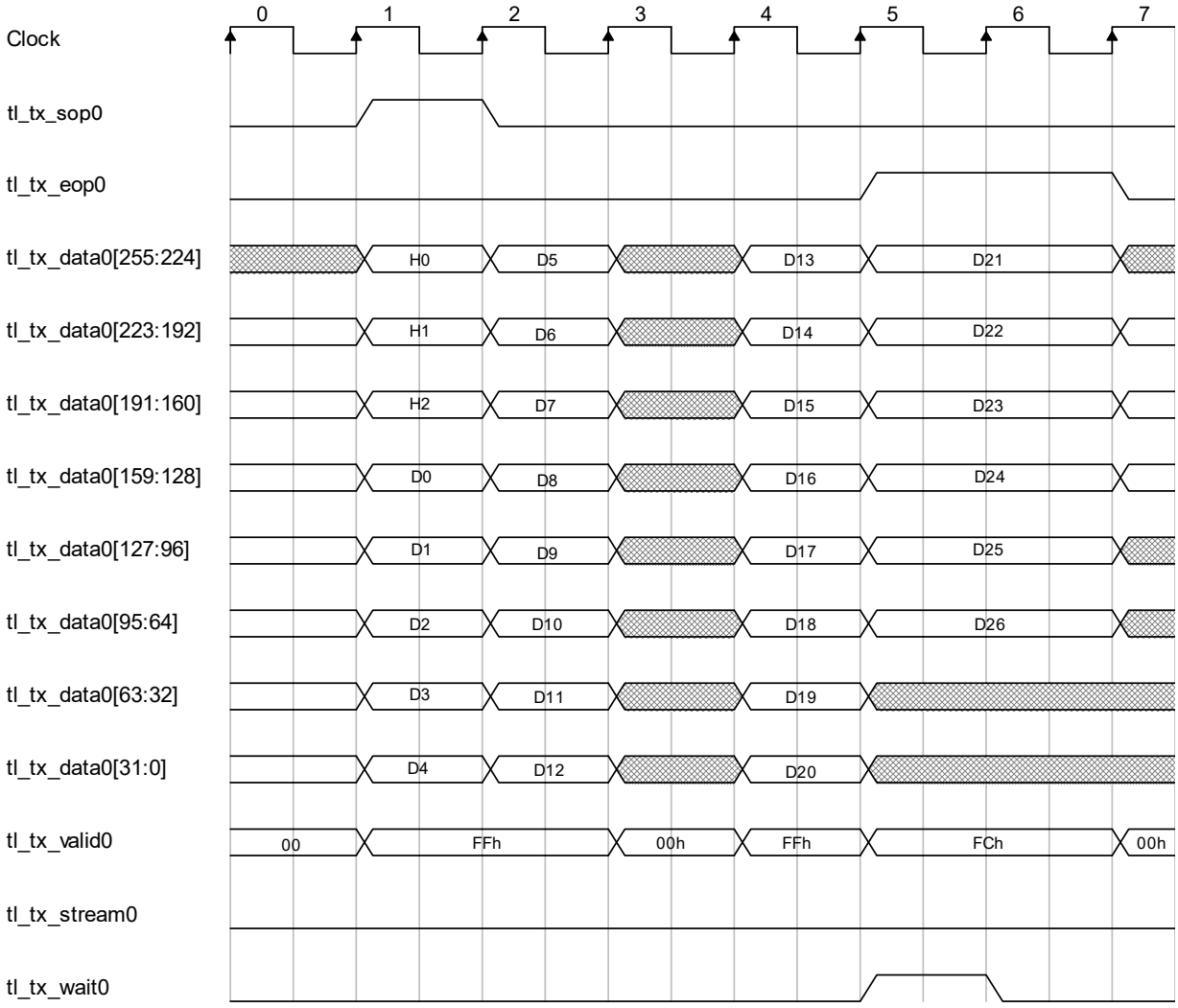


Figure 37: tl_tx_wait throttles transfer

7.3.1 ECRC Generation

The Core can append ECRC to transmitted packets when ECRC generation is enabled.

There are two ECRC generation modes:

- **Automatic mode (k_gen[31]=0)**:
  - If Core is a switch port then it does not modify any transmitted packet.
  - If Core is not a switch port then it automatically appends an ECRC to every transmitted packet that has TD=0 in its TLP header. If a packet has TD=1 in its TLP header, then it must already contain an ECRC generated by the application, so the Core does not modify it.

ECRC regeneration:

In a switch, during a multicast operation, a TLP's address can be modified by the switching logic due to the MC Overlay mechanism. If this TLP contains the optional ECRC, the unmodified ECRC will almost certainly indicate an error. In this case, the Core, as an egress switch port in Automatic Mode (k_gen[31]=0), supports ECRC regeneration. When the switching logic modify a TLP that contains an ECRC (TD=1), then the ECRC is dropped by the switching logic and the signal tl_tx_err0[3] is set to force the egress port to regenerate an ECRC.

- **TD-controlled mode (k_gen[31]=1)**:
  - If TD=0 in the TLP header then the Core does not modify the packet.
  - If TD=1 in the TLP header then the Core appends an ECRC (packet must not already contain an ECRC).

The following diagrams illustrate example TLPs for both automatic mode and TD-controlled mode for a 256-bit datapath:

- **Automatic Mode**
  - TLP transmitted without application-generated ECRC:

tl_tx_data0	Header0(TD=0)	Header0	Header0	Data0	-	-	-	-
tl_tx_valid	1	1	1	1	0	0	0	0
Transmitted TLP	Header0(TD=1)	Header0	Header0	Data0	ECRC			

  - TLP transmitted with application-generated ECRC:

tl_tx_data0	Header0(TD=1)	Header0	Header0	Data0	ECRC	-	-	-
tl_tx_valid	1	1	1	1	1	0	0	0
Transmitted TLP	Header0(TD=1)	Header0	Header0	Data0	ECRC			

- **TD-based Mode**
  - TLP transmitted without ECRC generation request:

tl_tx_data0	Header0(TD=0)	Header0	Header0	Data0	-	-	-	-
tl_tx_valid	1	1	1	1	0	0	0	0
Transmitted TLP	Header0(TD=0)	Header0	Header0	Data0				

  - TLP transmitted with ECRC generation request:

tl_tx_data0	Header0(TD=1)	Header0	Header0	Data0	-	-	-	-
tl_tx_valid	1	1	1	1	0	0	0	0
Transmitted TLP	Header0(TD=1)	Header0	Header0	Data0	ECRC			

7.4 Multicast

This section describes how the Multicast feature is handled by the Core and application logic.

7.4.1 Multicast for Endpoints

The diagram below illustrates multicast processing when the Core is an Endpoint:

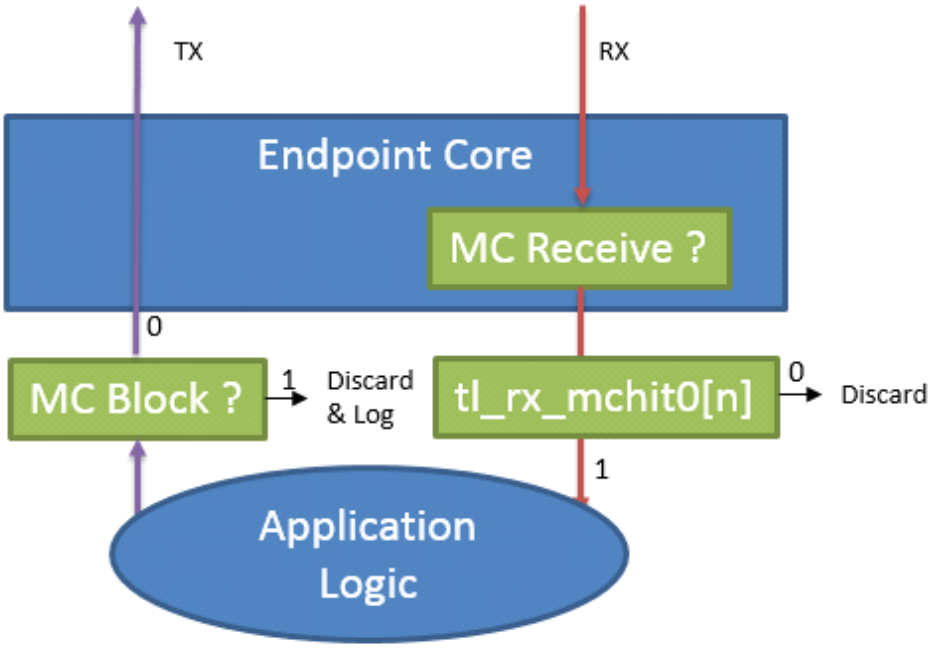


Figure 38: Multicast when Core is an Endpoint

When a multicast TLP is received, the Core extracts the multicast group of this TLP and checks the corresponding MC_Receive register's bit for all implemented functions: this information is provided by the signal `TL_RX_MCHIT0` when the TLP is presented on the Transmit/Receive interface. The application logic must check this signal to identify which functions, if any, should receive the TLP.

When an application transmits a multicast TLP, it must check the MC_Block_All/MC_Block_Untranslated register bits for its multicast group:

- if the TLP is blocked, then the application must report this error using `TL_REPORT_STATUS0` and discard the TLP.
- if the TLP is not blocked, the application sends the TLP to the Core's Transmit/Receive interface.

7.4.2 Multicast for Rootports/Switches

The diagram below illustrates multicast processing when the Core is a Rootport or Switch

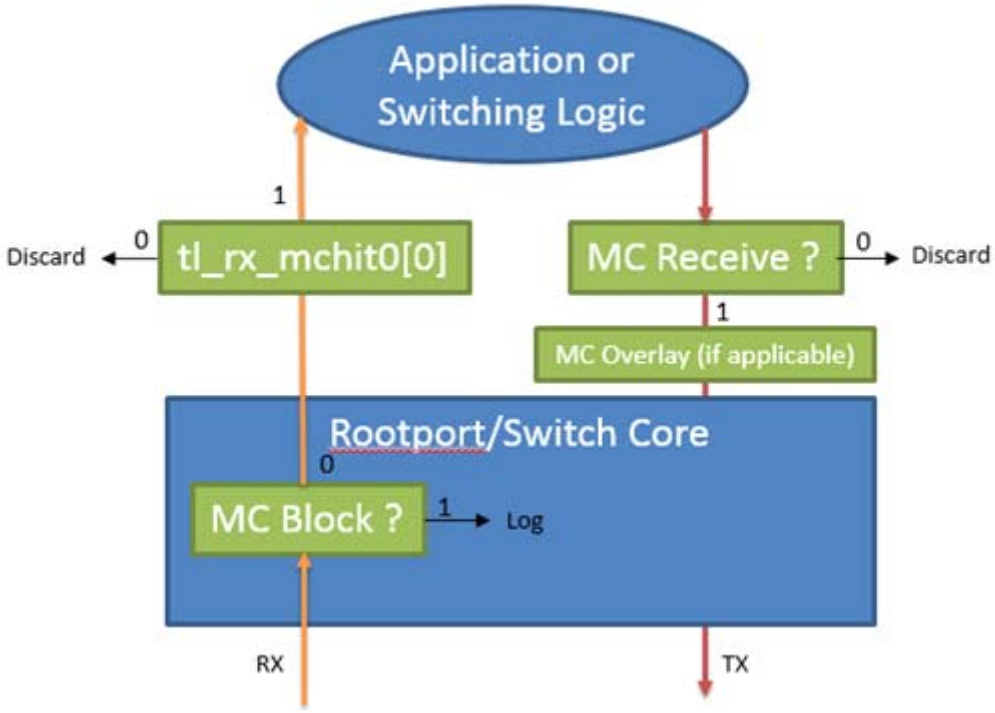


Figure 39: Multicast when Core is an Endpoint

When a multicast TLP is received, the Core extracts the multicast group of this TLP and checks the corresponding MC_Block_All/MC_Block_Untranslate register bits.

If the TLP is blocked, this error is logged and reported by the Core.

Whether the TLP is blocked or not, however, the Core still passes it to the application's Rx/Tx interface. The application logic checks `TL_RX_MCHIT0[15]` and either discards the TLP (if `TL_RX_MCHIT0[14]`= 1) or processes it.

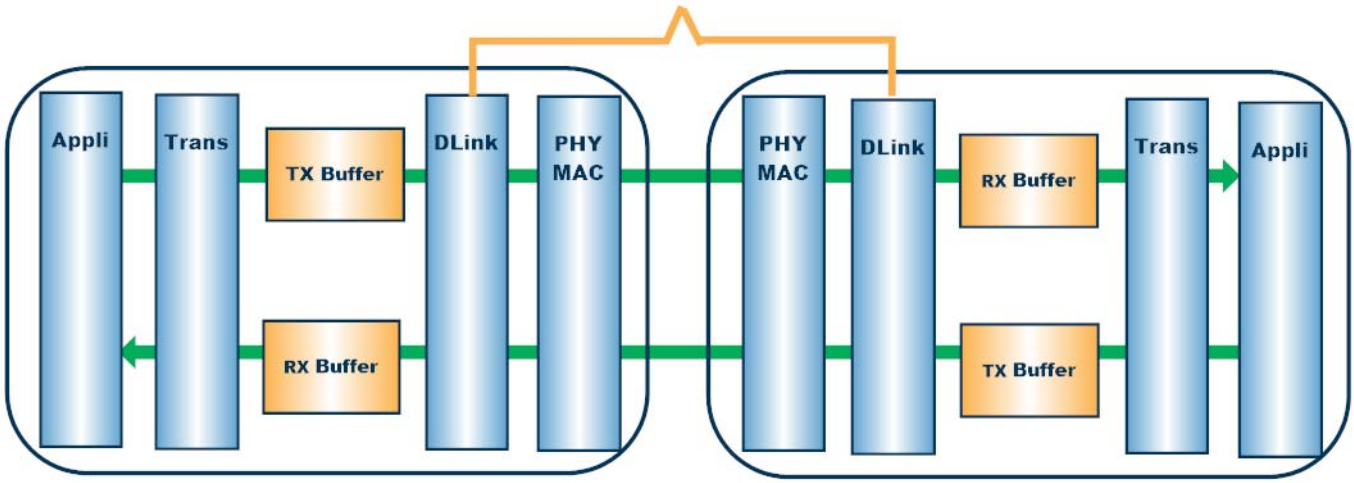
When an application transmits a multicast TLP, it must check its multicast group and transmit it only if the corresponding bit in the MC_Receive register is set. If it is not set, then the TLP must be discarded.

The application then checks the MC_Overlay register and performs an overlay operation on the TLP, if applicable, before sending it to the Core's Transmit/Receive interface.

Chapter 8 Data Protection

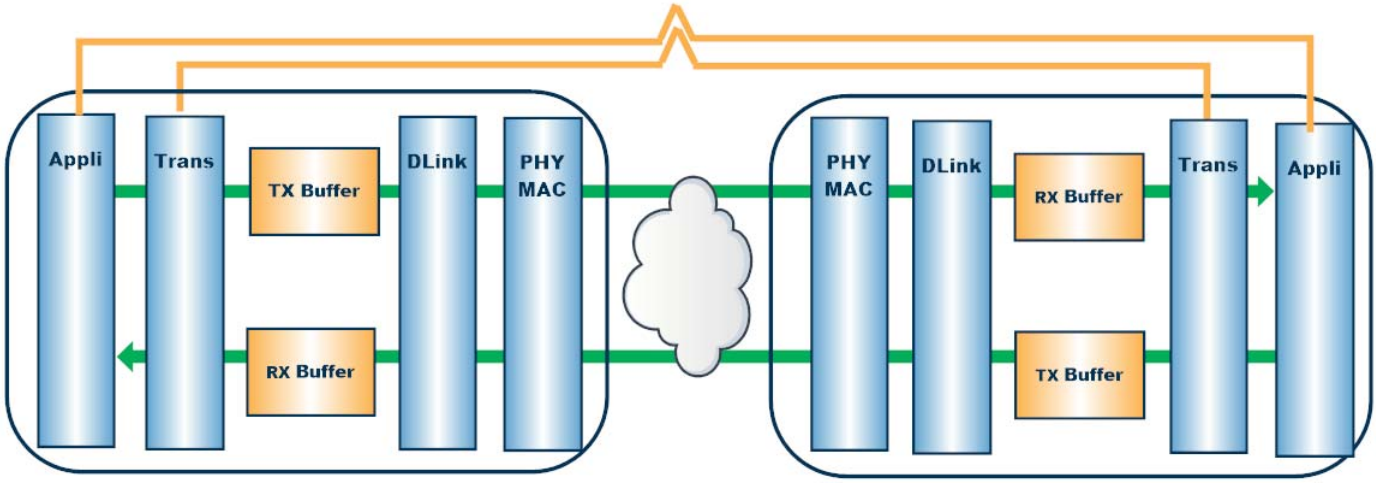
8.1 LCRC

LCRC is a standard PCIe feature that protects the contents of a TLP across a single PCIe link. It is generated/checked in the data link layer and is not available to the transaction layer and the application.



8.2 ECRC

ECRC is an optional PCIe feature that is available when advanced error reporting (AER) is implemented. It protects the contents of a TLP from its source to its ultimate receiver. It is typically generated/checked by the transaction layer. However, with XpressRICH3, the application can optionally generate/check ECRC itself in order to ensure data integrity across all data paths.



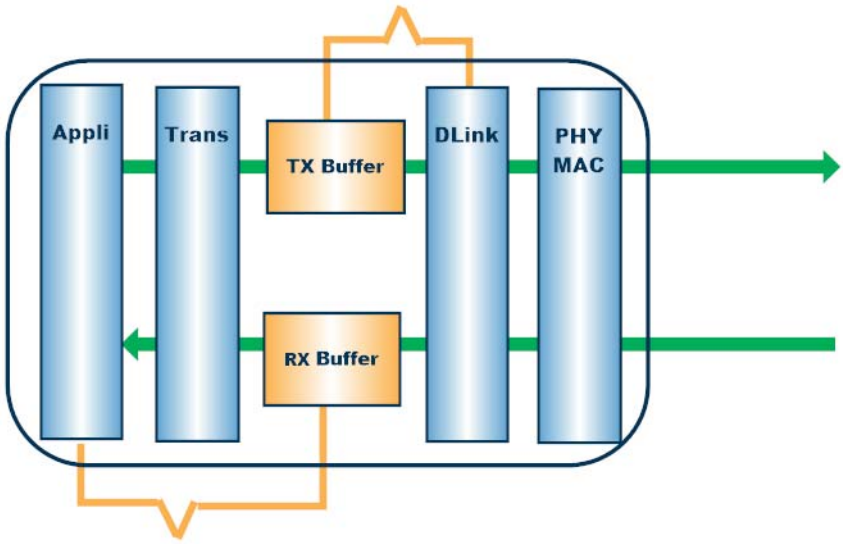
However, note that in order to be effective, ECRC must be supported by all the devices between the source and the ultimate receiver, and enabled by the system.

On XpressRICH3:

- The received ECRC, if present, is output on TL_RX_DATA0 after the payload.
- The Core can optional check ECRC, if enabled to do so. Any ECRC error is reported on TL_RX_ERR[0].
- The application can optionally check a received ECRC and/or generate and transmit ECRC immediately after payload on TL_TX_DATA0.

8.3 ECC

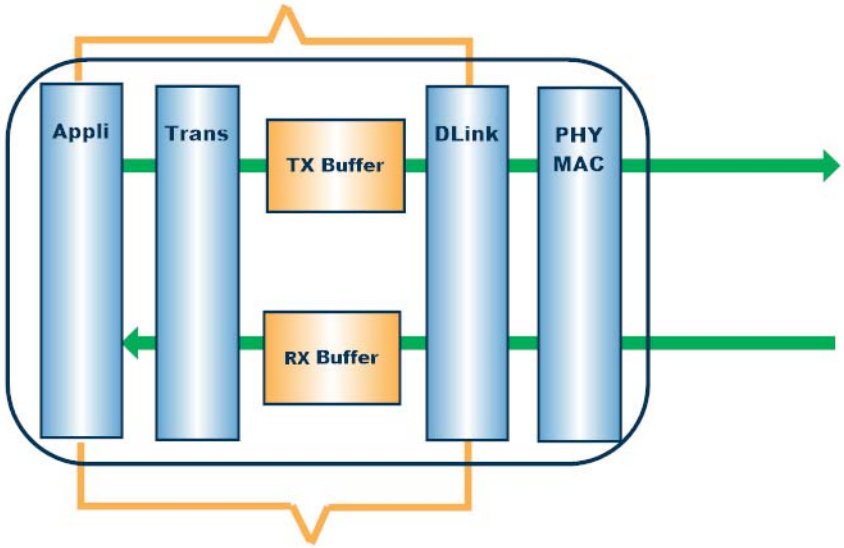
ECC is an optional mechanism implemented in some memories that can detect and/or correct some types of data errors that occurred in memory. When ECC is implemented, uncorrectable errors are reported on the RDDERR output of the transmit and receive buffers.



- On the Receive side, an uncorrectable memory error is reported on TL_RX_ERR[2] but does not modify Core behavior: it is the application's responsibility to take any appropriate action according to its error handling policy.
- On the Transmit side, an uncorrectable memory error is detected by the Core and signaled on TEST_OUT[447], additional processing is described in Section 8.5.

8.4 Parity

Parity data protection is an optional feature that adds parity bits to data to protect it between the data link layer and the application's logic.



- On the **Receive** datapath: the Data link layer adds parity to the data when LCRC is checked. Parity bits follow the same path as data through the data link layer, RX buffer and transaction layer. Then parity is output on TL_RX_PROT0 at the same time as data on TL_RX_DATA0 so that the application can check it. The Core does not check parity errors; the application must check for these and take any appropriate action according to its error handling policy.

- On the **Transmit** datapath: the application must generate parity and provide it to the Core on TL_TX_PROT0 at the same time as data on TL_TX_DATA0. Parity follows the same path as data through the transaction layer, TX buffer and data link layer. The Core checks parity in the data link layer and reports errors on TEST_OUT[446]. Additional processing is described in Section 8.5.

XpressRICH3 uses even parity to generate/check parity which means that parity is a simple XOR between data bits:

1-bit parity per 32-bit data (G_DATA_PROT=1):

```
parity = d[31] ^ d[30] ^ d[29] ^ … d[2] ^ d[1] ^ d[0];
```

4-bit parity per 32-bit data (G_DATA_PROT=4):

```
parity[0] = d[7]  ^ d[6]  ^ d[5]  ^ d[4]  ^ d[3]  ^ d[2]  ^ d[1]  ^ d[0];
parity[1] = d[15] ^ d[14] ^ d[13] ^ d[12] ^ d[11] ^ d[10] ^ d[9]  ^ d[8];
parity[2] = d[23] ^ d[22] ^ d[21] ^ d[20] ^ d[19] ^ d[18] ^ d[17] ^ d[16];
parity[3] = d[31] ^ d[30] ^ d[29] ^ d[28] ^ d[27] ^ d[26] ^ d[25] ^ d[24];
```

8.5 TX Error Management

When the Core detects an ECC or parity error while transmitting a TLP, it reports the information to the application. It can also take specific actions to contain the error, depending on its operating mode. The following table shows the different operating modes that can be set using the Core variables K_GEN[33] and K_GEN[34]:

k_gen[34]	k_gen[33]	Mode
0	0	Reporting-Only
0	1	Error Nullification
1	1	Error Flush

Table 32: Tx Error Management Operating Modes

8.5.1 Reporting-only Mode

In Reporting-Only mode, when the Core detects an ECC or parity error while transmitting a TLP, it informs the application that a transmit error occurred, but keeps transmitting TLPs (including the TLP on which the error was detected) without any change.

The TL_TX_PROTERR0[31] signal is pulsed for one or more clock cycles to report the error, while TL_TX_PROTERR0[30:0]] reports essential TLP information at the same time.

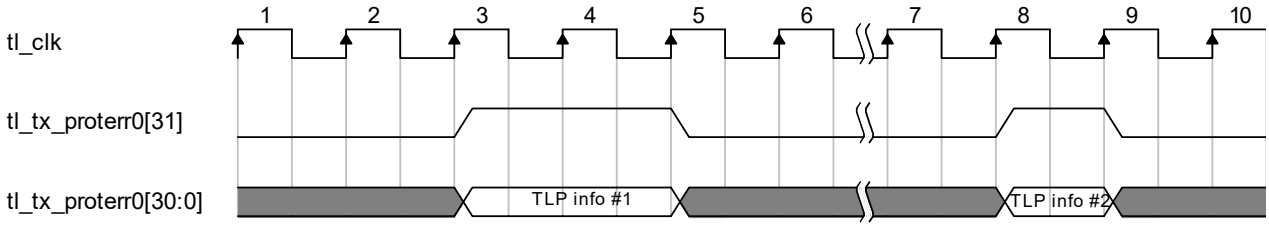


Figure 40: Error reporting using tl_tx_proterr0

Note: If several consecutive errors occur then TL_TX_PROTERR0 might not accurately report them all, because of clock domain crossing interaction.

8.5.2 Error Nullification Mode

In Error Nullification mode, when the Core detects an ECC or parity error while transmitting a TLP, it informs the application that a transmit error has occurred and the TLP on which the error is detected is nullified. Subsequent TLPs are transmitted normally, however.

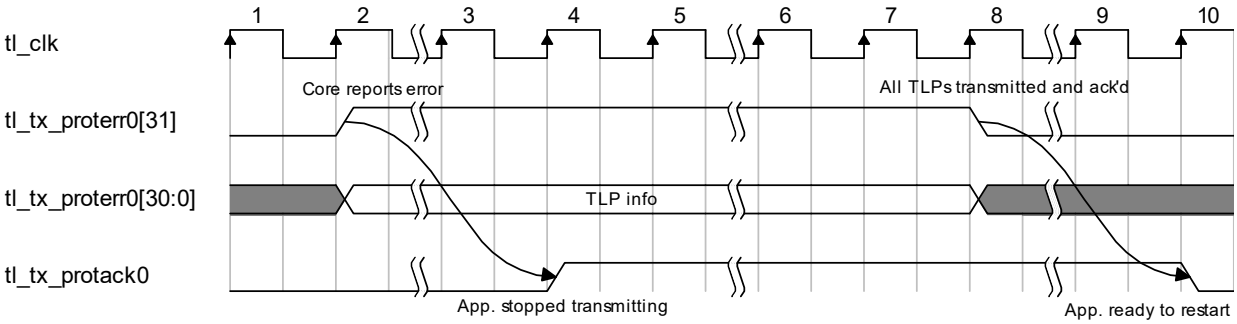


Figure 41: Error Nullification

Credits for the erroneous TLP are subtracted by the Transaction Layer before the TLP is nullified, so the following mechanism is used to ensure that no credits are lost:

1. The Core asserts `TL_TX_PROTERR0[31]` and nullifies the packet that contained the error, but transmits subsequent TLPs normally (unless an error is detected in them, in which case they are also nullified). It also disables Internal TLP generation.
2. The application stops transmitting as soon as possible and asserts `TL_TX_PROTACK0` when it has done so (note that the Core does not stall the `TL_TX_...` interface).
3. The Core waits until the Tx buffer is empty, all transmitted TLPs have been acknowledged, and `TL_TX_PROTACK0` is asserted.
4. The Core then resets Tx credits to their proper value, deasserts `TL_TX_PROTERR0[31]`, and enables internal TLP generation.
5. The application de-asserts `TL_TX_PROTACK0`and can resume TLP transmission.

8.5.3 Error Flush Mode

In Error Flush mode, when the Core detects an ECC or parity error while transmitting a TLP, it informs the application that a transmit error occurred and the TLP on which an error is detected, as well as all subsequent TLPs, are flushed. Internally-generated TLPs (such as Cpl for Cfg access, messages, interrupts, etc.) are not flushed, however, and are transmitted normally as there is no recovery method.

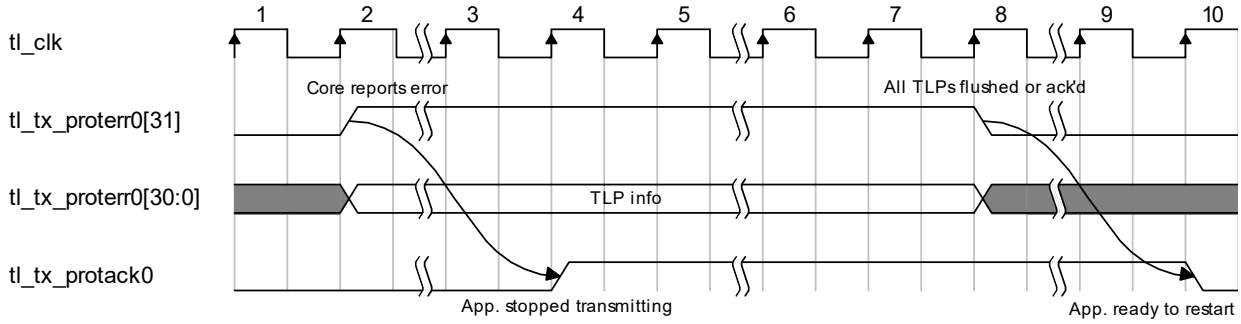


Figure 42: Error Flush

The following mechanism is used to ensure that the Tx datapath is completely flushed and that no credits are lost:

1. The Core asserts `TL_TX_PROTERR0[31]` and nullifies the packet that contained the error, as well as all subsequent TLPs (except internally generated TLPs). It also disables Internal TLP generation.
2. The application stops transmitting as soon as possible and asserts `TL_TX_PROTACK0` when it has done so (note that the Core does not stall the `TL_TX_...` interface).
3. The Core waits until the Tx buffer is empty, all transmitted TLPs have been acknowledged, and `TL_TX_PROTACK0` is asserted.
4. The Core then resets Tx credits to their proper value, deasserts `TL_TX_PROTERR0[31]`, and enables internal TLP generation.
5. The application de-asserts `TL_TX_PROTACK0` and can resume TLP transmission.

8.5.4 Tx Error Management Signals

Signal	I/O	Description
tl_tx_proterr0[31:0]	Out	**Data protection error indication**: • Bit [31]: TLP error status indicator • Bits [30:29]: TLP header fmt[1:0]; valid when TLP error status is set. • Bits [28:24]: TLP header type; valid when TLP error status is set. • Bits [23:8]: TLP completer ID (for completions) or requester ID (all other TLP types); valid when TLP error status is set. • Bits [7:0]: TLP header tag; valid when TLP error status is set. The full TLP header is not provided so it is recommended that the application transmits TLPs with unique tags for each type of TLP, in order to be able to identify them easily. Note that if several consecutive errors occur then TL_TX_PROTERR0 might not accurately report them all, because of clock domain crossing interaction.
tl_tx_protack0	In	**Data protection acknowledge**: This input is used in Error Nullification and Error Flush modes; otherwise it must be tied to 0. • It is asserted when TL_TX_PROTERR0[31] is asserted, when the application has stopped transmitting. • It is de-asserted when TL_TX_PROTERR0[31] is de-asserted, when the application is ready to resume transmitting.

Table 33: TX Error Management Signals

Note: TEST_OUT[447:446] reports additional information on the nature of any errors detected.

Chapter 9 Physical Function Interrupts

The XpressRICH3 Core supports all three interrupt mechanisms defined by PCI Express: INTx, MSI & MSI-X. This chapter describes how the Core handles interrupts for Physical functions. Information on interrupts for virtual functions can be found in Chapter 12.

9.1 Interrupt Interface

The following table describes Core interrupt signals for physical functions. All signals are synchronous to `TL_CLK`.

Signal	I/O	Description
tl_int_status0 (also for functions 1 - 7)	In	**Interrupt Status**: This signal must be asserted when the physical function requests interrupts, and de-asserted when the application detects that all interrupts have been cleared. If MSI and MSI-X are not implemented, and interrupts are not handled externally (see Section 9.4) then a legacy interrupt message is sent on the rising and falling edges of this signal.
tl_int_msireq0 (also for functions 1 - 7)	In	**MSI Request**: This signal is asserted by the application for one clock cycle to request an interrupt.The application must wait for `TL_INT_MSIACK0` to be asserted before requesting another MSI.No message is transmitted if MSI is not enabled, or if the MSI message number is masked.
tl_int_msinum0[4:0] (also for functions 1 - 7)	In	**Interrupt MSI Message Number**: This signal indicates the MSI message number to use. If multiple MSI messages are not used or MSI is not enabled, then this signal is ignored and must be set to 0.
tl_int_msiack0 (also for functions 1 - 7)	Out	**MSI Acknowledge**: This signal is asserted by the Core for one clock cycle to acknowledge an interrupt request (`TL_INT_MSIREQ0` is asserted), even if no message has been sent.
tl_int_pinstate[3:0]	Out	**Interrupt Lines State**: This signal is used in Rootport and bridge/switch downstream ports to report the state of virtual interrupt pins: • Bit 0: INTA# • Bit 1: INTB# • Bit 2: INTC# • Bit 3: INTD#
tl_int_pincontrol[3:0]	In	**Interrupt Lines Control**: This signal is used in bridge/switch upstream ports to set the state of virtual interrupt pins: • Bit 0: INTA# • Bit 1: INTB# • Bit 2: INTC# • Bit 3: INTD#

Table 34: Interrupt signals

9.2 Interrupt Handling

The following table show how the Core handles interrupts, depending on the interrupt mechanism implemented (INTx, MSI & MSI-X).

When the application requests an interrupt, the XpressRICH3 performs one of the following actions:

MSI enabled	MSI-X implemented and enabled	Core Action
no	no	• If the PF Interrupt Status (TL_INT_STATUS) signal is asserted or deasserted, the Core sends an INTx assert/deassert message • If the MSI Request: (TL_INT_MSIREQ) signal is asserted or deasserted, the Core acknowledges the request but does not send any message.

yes	no	• If the PF Interrupt Status (TL_INT_STATUS) signal is asserted or deasserted, then the Core does not perform any action.
		• If the MSI Request: (TL_INT_MSIREQ)signal is asserted:
		  • If the requested message is implemented and not masked, the Core sends an MSI with the requested message number.
		  • If the requested message is not implemented or is masked, the Core acknowledges the request but does not send any message.

no	yes	Although not necessary, the application can assert the MSI Request: (TL_INT_MSIREQ); if it does, the Core acknowledges the request but does not send any message. The application must transmit an MSI-X via the Transmit interface.

Table 35: Interrupt Handling

Note: MSI and MSI-X should never be enabled simultaneously; this is a configuration error.

The following waveform shows an interrupt requested by the physical function. TL_INT_STATUS0 indicates when an interrupt has been requested; it is deasserted when the application detects that the interrupt has been cleared.

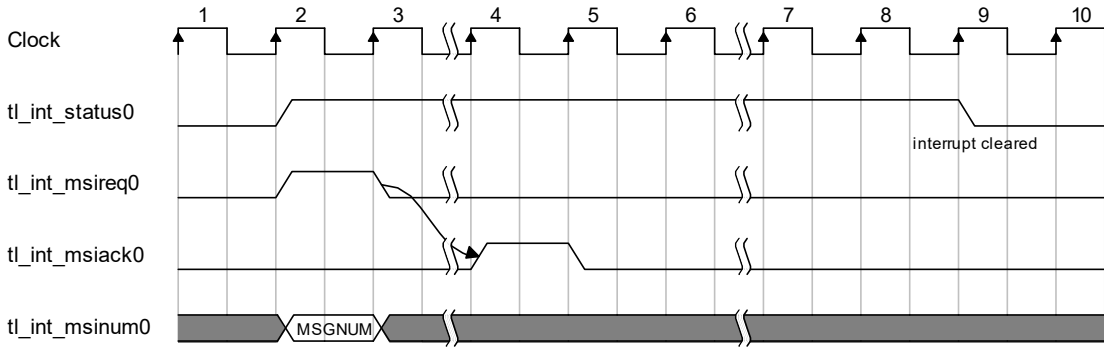


Figure 43: Single interrupt requested by physical function

The following waveform shows several concurrent interrupts requested by the physical function. TL_INT_STATUS0 indicates that an interrupt has been requested; it is deasserted when the application detects that the interrupt has been cleared.

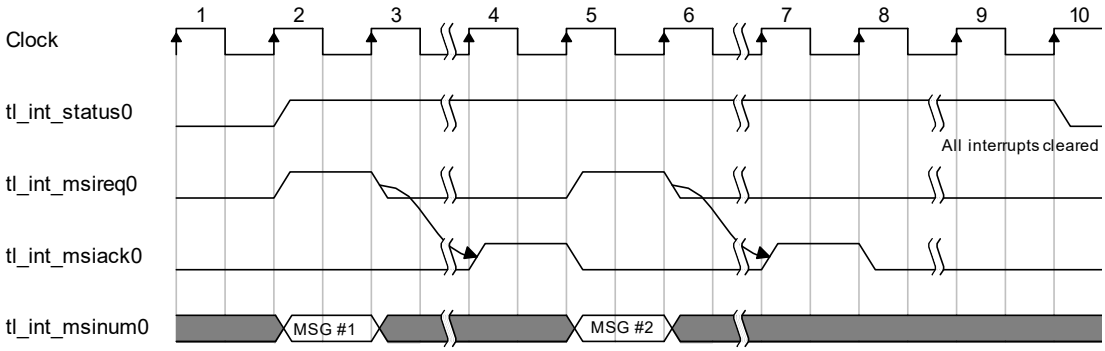


Figure 44: Consecutive interrupts requested by physical function

9.3 Per-Vector Masking

Per-vector masking is an optional feature set by the Core variable K_PCICONF, (see Section 4.2) that adds Mask Bits and Pending Bits registers to MSI capability:

- The Mask Bits register is programmed by the host and its value can be accessed by the application using TL_CFG_REGS0..7.
- Pending bits must be implemented and maintained by the application, which can report their value on TL_REPORT_STATE0..7.

Note: It is the application's responsibility to resend pending MSI when the MSI message number is unmasked.

9.4 External Interrupt Mode

External interrupt mode enables the application to send INTx / MSI messages rather than relying on the Core to do so. This special mode is enabled for physical functions using K_GEN[32]. When it is enabled, the application no longer uses TL_INT_MSI... interface signals to request MSI, but sends INTx/MSI messages using the Core TX interface instead.

In this mode, the TL_INT_STATUS signal must still be asserted when the application has pending INTx interrupts.

When this mode is enabled, it is also possible to implement MSI and/or MSI-X capabilities externally (see K_PCICONF bits 131 and 159) so that the application has full control of interrupt processing.

Chapter 10 Power Management

10.1 Legacy and Native Power Management

10.1.1 Signals

The following table describes legacy and native power management signals. All signals are synchronous to the TL_CLK. Signals ending with a zero are implemented per function.

Signal	I/O	Description
tl_pm_event0 (also for functions 1 - 7)	in	**Power Management Event**: (Endpoint only) This signal initiates a Power Management Event Message (PM_PME) that is sent to the Root Complex to request an exit from L1/L2. If the Link is in low-power state L2, a Beacon (or Wake#) is generated in order to re-initialize the Link before the Core generates the message. This signal is positive-edge sensitive. If TL_CLK is turned off then TL_PM_EVENT should be asserted until TL_CLK is restarted or the transaction layer is reset. This signal must be hardwired to zero for downstream components (Rootports). **Note**: If the CDC is not implemented and PL_PCLK is turned off, then this signal has no effect and cannot be used to wake-up the device.

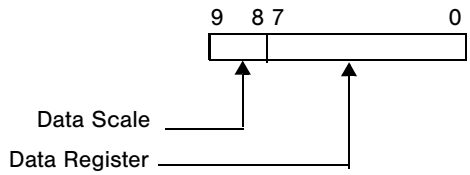
tl_pm_data0[9:0]	in	
		Figure 45: Power Management Data
		Power Management Data: This bus indicates power consumption of the component and can only be implemented if all three bits of AUX_power (part of the Power Management Capabilities structure) are set to 0. It includes the following bits:
		• PM_DATA[7:0]: Data Register: This register is used to maintain a value associated with the power consumed by the component.
		• PM_DATA[9:8]: Data Scale: This register is used to maintain the scale used to find the power consumed by a particular component and can include the following values:
		  • 00: unknown
		  • 01: 0.1 x
		  • 10: 0.01 x
		  • 11: 0.001 x
		For example, the two registers might have the following values:
		• pm_data[7:0]: 1110010 = 114
		• pm_data[9:8]: 10, which encodes a factor of .01
		To find the maximum power consumed by this component, multiply Data Value by Data Scale (114 X .01 = 1.14). 1.14 Watts is the maximum power permitted to this component in the power state selected by the data_select field.

Table 36: Power management signals

Signal	I/O	Description
tl_pm_l2_control	in	**L2 Control**: This signal controls L2 state entry and its behavior varies depending upon the type of component: • **Downstream Port**: The application can assert this signal to request L2 state entry. This signal must remain asserted until TL_PM_L2_STATUS is asserted, indicating that the link partner is ready to enter L2. If the CDC is implemented, then de-asserting this signal makes the Downstream Port exit the L2 state and wakes the Link; otherwise it has no effect. • **Upstream Port**: The application must assert this signal following a TL_PM_L2_STATUS assertion, to indicate that it is ready for L2 state entry. This signal must remain asserted for as long TL_PM_L2_STATUS is asserted.
tl_pm_l2_status	out	**L2 Status**: This signal indicates L2 state entry status and its behavior varies depending upon the type of component. • **Downstream Port**: The Core asserts this signal after TL_PM_L2_CONTROL is asserted, when it receives acknowledgment that the link partner is ready to enter L2. This signal is deasserted once LTSSM has entered L2 state. • **Upstream Port**: This signal is asserted when the Upstream Port receives a Link turn off message from the Downstream Port, as shown below. It is de-asserted after LTSSM has entered the L2 state, after TL_PM_L2_CONTROL is asserted.
tl_pm_auxpwr	in	**Auxiliary Power Detected**: This signal is asserted when auxiliary power is detected.
pl_wake_oen	out	**WAKE# Signal Open-Drain Output Control**: This active-low signal controls the WAKE# pin open-drain output. It is asynchronous when PL_CLK is not available. This output is used to drive the optional WAKE# pin if it is present: • For upstream devices: this pin drives the wake-up mechanism in the L2 state. • For downstream devices: this pin is used for the OBFF mechanism, if implemented. When unused this output can be left open.
pl_wake_in	in	**WAKE# Signal Input**: This signal reads the state of the WAKE# pin. This input is used to read the state of optional WAKE# pin if it is present: • For upstream devices: this pin is used for the OBFF mechanism, if implemented. • For downstream devices: this pin is used for t the wake-up mechanism in the L2 state. When unused this input must be tied to 1.

Table 36: Power management signals

10.1.2 ASPM L0s

ASPM L0s is a mandatory PCI Express native power management mode that allows a device to very quickly suspend or resume transmission during periods of inactivity. ASPM L0s does not require any application action.

It can be entered by any type of device and does not require any negotiation with the link partner.

1. The Core automatically enters ASPM L0s after a period of inactivity (defined by `K_PEXCONF[209:205]`).
2. The Core automatically exits ASPM L0s when necessary.

Note: ASPM L0s must be enabled in the PCI Express Link Control register to allow entry.

The number of fast training sequences (NFTS) in `K_PEXCONF` must be set according to PHY vendor recommendations to ensure proper exit from this state.

10.1.3 ASPM L1

ASPM L1 is an optional PCI Express native power management mode that enables the link to be put into low-power mode with the ability to restart quickly during periods of inactivity. ASPM L1 does not require any application action. It involves negotiation between the link partners and can only be initiated by an upstream port (such as an endpoint).

1. The upstream port (endpoint) automatically requests ASPM L1 entry after a period of inactivity (defined by `K_PEXCONF[214:210]`).
2. The downstream port (rootport) can either:
   - reject the entry request; in this case, the link stays in full operational state and the upstream port waits for at least 10µs before requesting ASPM L1 entry again.
   - accept the entry request; in this case, both devices enter ASPM L1.
3. Each link partner automatically exits low-power when there are packets to transmit.

Note: ASPM L1 must be enabled in the PCI Express Link Control register.

The ASPM L1 entry delay must be longer than the ASPM L0s entry delay because ASPM L1 is a deeper power-saving state that takes more time to enter and exit.

10.1.4 L1

L1 is a legacy power state that is automatically entered by an upstream port (such as an endpoint) when its legacy power state is not D0. L1 entry does not require any action from the application; however, the application can request L1 exit.

1. The upstream port (endpoint) automatically enters L1 when the legacy low-power state is set to a non-D0 state by the host.
2. The downstream port (rootport) enters L1 when entry is requested by its link partner and there is no packet to be transmitted.
3. The downstream port (rootport) automatically exits L1 as soon as a packet needs to be transmitted. The upstream port (endpoint) exits L1 only when directed to do so by its link partner; however, the application can send a power management event PME# using `TL_PM_EVENT` to ask the host to change its legacy power state.

Note: Legacy power management control and status registers must be properly configured in order to allow power management event transmission. The legacy power management register is located in configuration register address 0FCh.

10.1.5 L2

L2 is a legacy power state where the PCIe link is completely turned off and power may be removed. L2 entry must be initiated by a downstream port (such as a rootport).

1. The downstream port (rootport) enters the L2 state and disables the link as soon as its link partner indicates that it's ready (the application asserts `TL_PM_L2_CONTROL`).
2. The upstream port (endpoint) enters the L2 state when requested and when the application indicates with `TL_PM_L2_STATUS` that it is ready.
3. The downstream port (rootport) can exit L2 either when the application no longer asserts `TL_PM_L2_CONTROL`, or when a beacon/power management event is received from its link partner. The upstream port (endpoint) exits L2 only when directed to do so by its link partner; however the application can send a power management event PME# using `TL_PM_EVENT` to ask the host to restore the link.

The example below shows an L2 entry seen from the downstream port side, followed by a wake-up initiated by the downstream port:

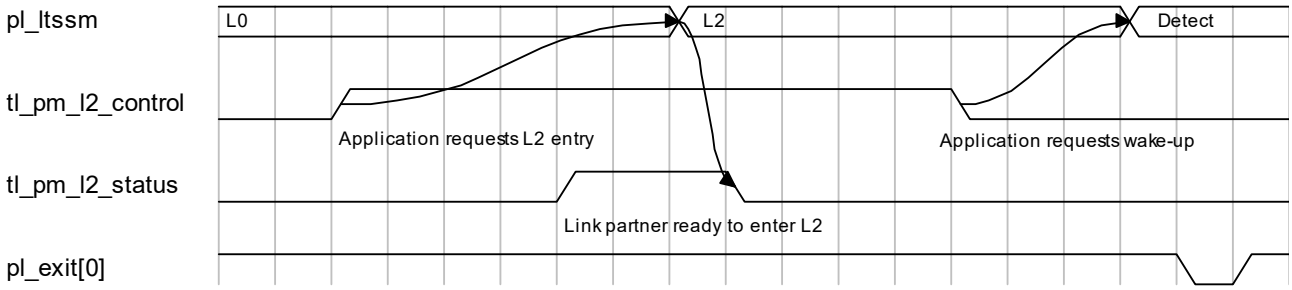


Figure 46: L2 Entry from Downstream Port

The example below shows an L2 entry seen from the upstream port side, followed by a wake-up initiated by the upstream port:

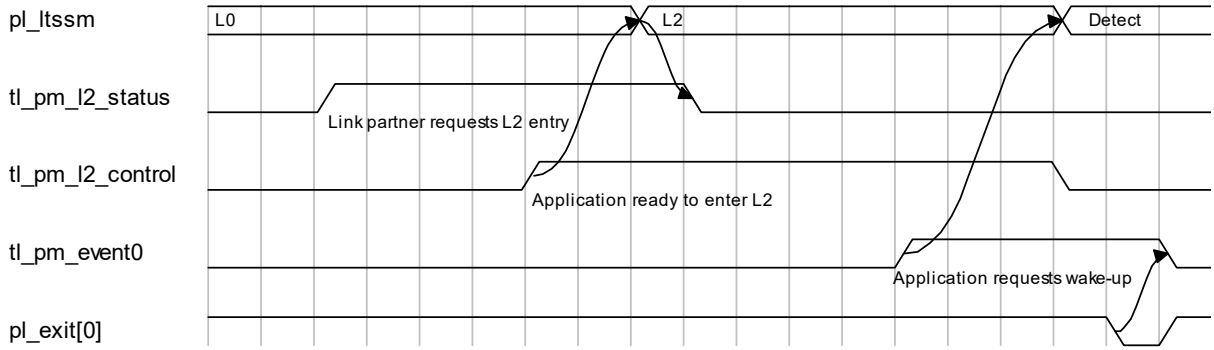


Figure 47: L2 Entry from Upstream Port

Note: Legacy power management control and status registers must be properly configured in order to allow power management event transmission. The legacy power management register is located in configuration register address 0FCh.

10.2 Power Management with CLKREQ#

CLKREQ# is an optional side-band pin present in some form factors, which enables power saving. It is used by the Clock Power Management and L1 PM substates with CLKREQ# features. Note that only one of these power saving techniques can be enabled at any one time.

10.2.1 Interface Signals

The following table describes the interface signals used with CLKREQ#:

Signal	I/O	Description
pl_clkreq_oen	out	**CLKREQ# enable**: Open-drain control for CLKREQ# output: • When 0: CLKREQ# drives 0 • When 1: CLKREQ# is in high-impedance This output is used when either Clock Power Management or L1 PM Substates are implemented.
pl_clkreq_in	in	**CLKREQ# input**: this signal must be connected to the CLKREQ# line. It is used when L1 PM substates are implemented.
tl_pm_refclk_rem	in	**Allow Slot REFCLK Removal**: this signal is used by the application to indicate when the PCI Express reference clock can be safely removed (when applicable). • 0: the application does not allow the reference clock to be removed. • 1: the application allows the reference clock to be removed. This signal is not used and must be tied to 0 when Clock Power Management and L1 PM substates with CLKREQ# are not implemented.

tl_pm_l1ss_status[7:0]	out	**L1 Sub-States Status**:
		• bits 2:0: current state:
		  • 000: Inactive: default, means that either LTSSM is not in the L1 state or the Core is not enabled to enter a L1 sub-state
		  • 001: L1.1
		  • 010: L1.2 entry
		  • 011: L1.2 idle
		  • 100: L1.2 exit
		  • 101: L1.0
		  • 110: Entry: the Core has left Inactive state and is preparing to enter L1.0
		  • 111: Exit: the Core has left L1.0 and is preparing to revert to Inactive state
		• bits 7:3: reserved

tl_pm_l1ss_entreq	out	**L1 Sub-States Entry/Exit Request**: the Core asserts this signal to indicate that the PHY should prepare for possible L1 sub-state entry, and de-asserts it to indicate that the PHY should prepare for L1 exit. If L1 sub-states are not supported then this signal can be left unconnected.
tl_pm_l1ss_entack [G_NUM_LANES]	in	**L1 Sub-States Entry Acknowledgment**: PHY reports with this per-lane signal when L1 sub-states entry/exit preparation is complete: • each bit of this signal must be asserted following a `TL_PM_L1SS_ENTREQ` assertion, when the corresponding lane is prepared for L1 sub-state entry. • each bit of this signal must be de-asserted following a `TL_PM_L1SS_ENTREQ` de-assertion, when the corresponding lane is ready for L1 exit. If L1 sub-states are not supported then this signal can be tied to 0; if L1 sub-states are supported but the PHY does not need to perform any entry/exit preparation then this signal must be tied to `TL_PM_L1SS_ENTREQ`.

Table 37: CLKREQ# management signals

10.2.2 Clock Power Management

Note: Clock Power Management is a deprecated feature available only in some form factors such as PCIe mini cards; it is highly recommended that new designs implement L1 PM sub-states instead.

CLKREQ# functionality is implemented via the Link Capability and Link Control registers in the Configuration Space. When multiple functions are enabled, each function must be CLKREQ# capable.

When CLKREQ# is used to manage power consumption in the L2 state, the MAC transitions from the P0 to the P2 low power state to stop the Reference Clock, then deasserts CLKREQ#.

If TL_PM_REFCLK_REM is set to 1 when entering the L1 state, then the MAC transitions from P0 to P2 directly, instead of finishing in P1. When transitioning out of the L1 or L2 states, CLKREQ# is asserted to re-start the Reference Clock.

The following waveform illustrates an L1 entry where the application does not allow clock removal (TL_PM_REFCLK_REM=0), followed by an L1 entry where the application does allow clock removal (TL_PM_REFCLK_REM=1):

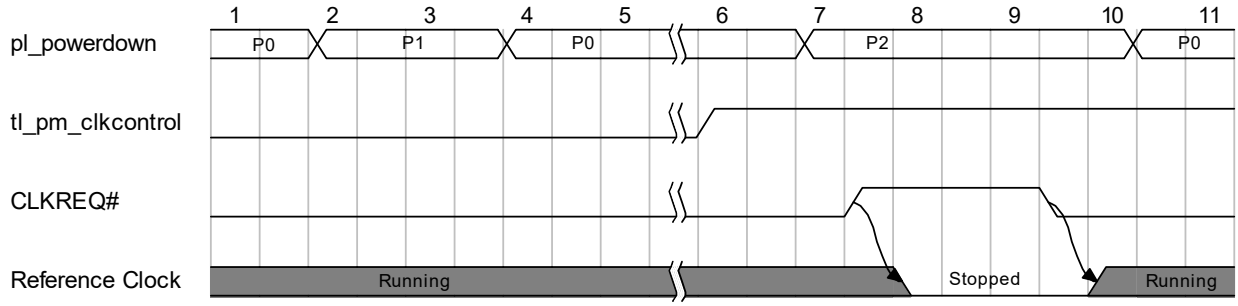


Figure 48: L1 entry with & without clock removal

Note: While specifications define Clock Power Management for x1 mobile form factors only, there is no limitation in the XpressRICH3 Core itself, so CLKREQ# can be implemented for any lane configuration.

The CDC must be implemented when using Clock Power Management, as the PIPE clock (PL_PCLK) can be stopped when the PHY is in the P2 state. The Transaction Layer clock (TL_CLK) must continue to run, however, even if at a very low frequency, so that any traffic on the Transaction Layer Transmit interface will cause the Core to assert CLKREQ# and exit low-power.

10.2.3 L1 PM Substates with CLKREQ#

L1 PM sub-states is an addition to the PCI Express Specification that enables deep power savings in the L1 and ASPM L1 states.

10.2.3.1 Enabling L1 PM Sub-States

L1 PM sub-states are configured via the L1 PM sub-states capability and the T_POWEROFF parameter (K_PEXCONF[263:261], see Table 18). The configuration of these sub-states also requires that Latency Tolerance Reporting is implemented and properly configured. Values for TL_REPORT_LATENCY must always be valid as they are used to check if L1 sub-state entry is possible.

In order to verify that the Core is configured properly, you can check TEST_OUT[93:90] to verify which sub-states are enabled and can be entered from L1.

10.2.3.2 Sub-States Entry/Exit

Whenever the Core enters the L1 state, it checks if all conditions are met to enter a substate.

If no suitable sub-state is enabled or the application does not allow REFCLK removal (TL_PM_REFCLK_REM=0), then the Core enters L1 and does not take any further action. This is illustrated in the waveform below:

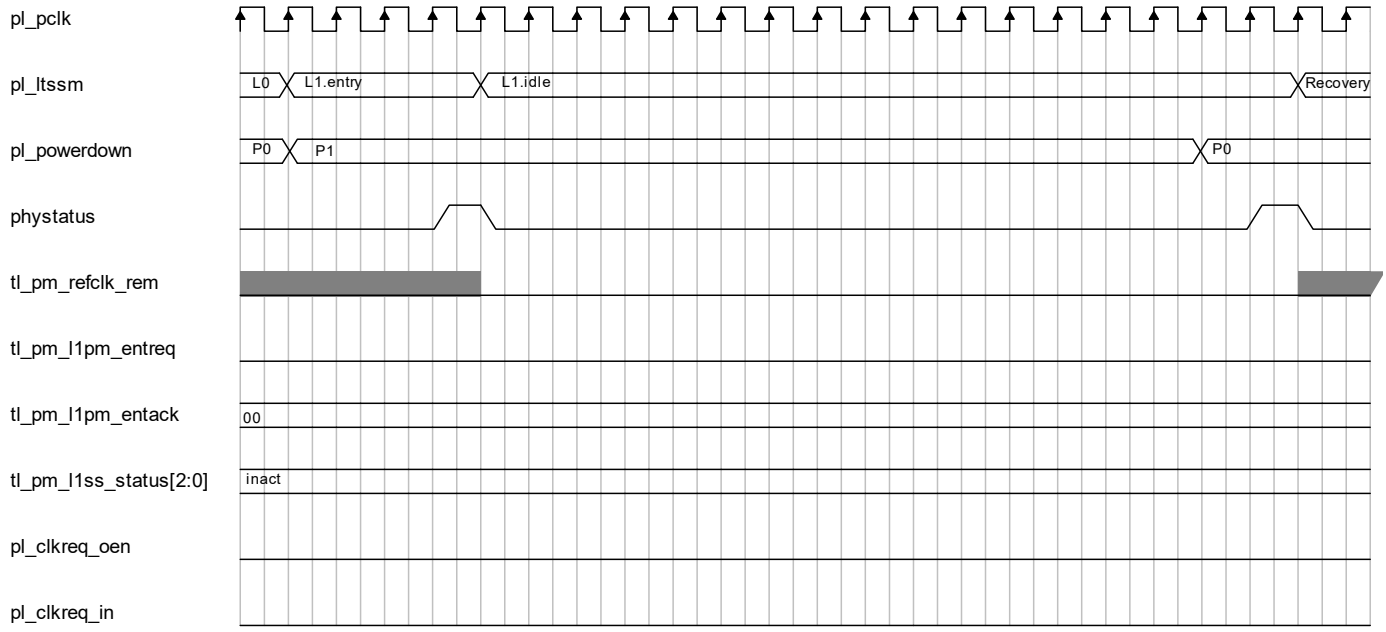


Table 38: L1 Entry without Sub-States

If a suitable sub-state is enabled and the application allows REFCLK removal (TL_PM_REFCLK_REM=1) then the Core:

1. Enters the Entry state, where it requests the PHY to prepare for possible L1 sub-state entry.
2. When all PHY lanes have acknowledged this request (the PHY can turn off PL_PCLK at this point), the Core moves to L1.0 and deasserts CLKREQ#.
3. If CLKREQ# is sampled deasserted, the Core can then enter the L1.1 or L1.2 sub-states.

To exit L1, the Core:

1. First reverts to L1.0, then enters the Exit state where it requests the PHY to prepare for L1 exit.
2. When all PHY lanes have acknowledged this request, and the PHY has turned PL_PCLK back on, the Core exits L1.

The waveform below shows an entry in L1.1 state:

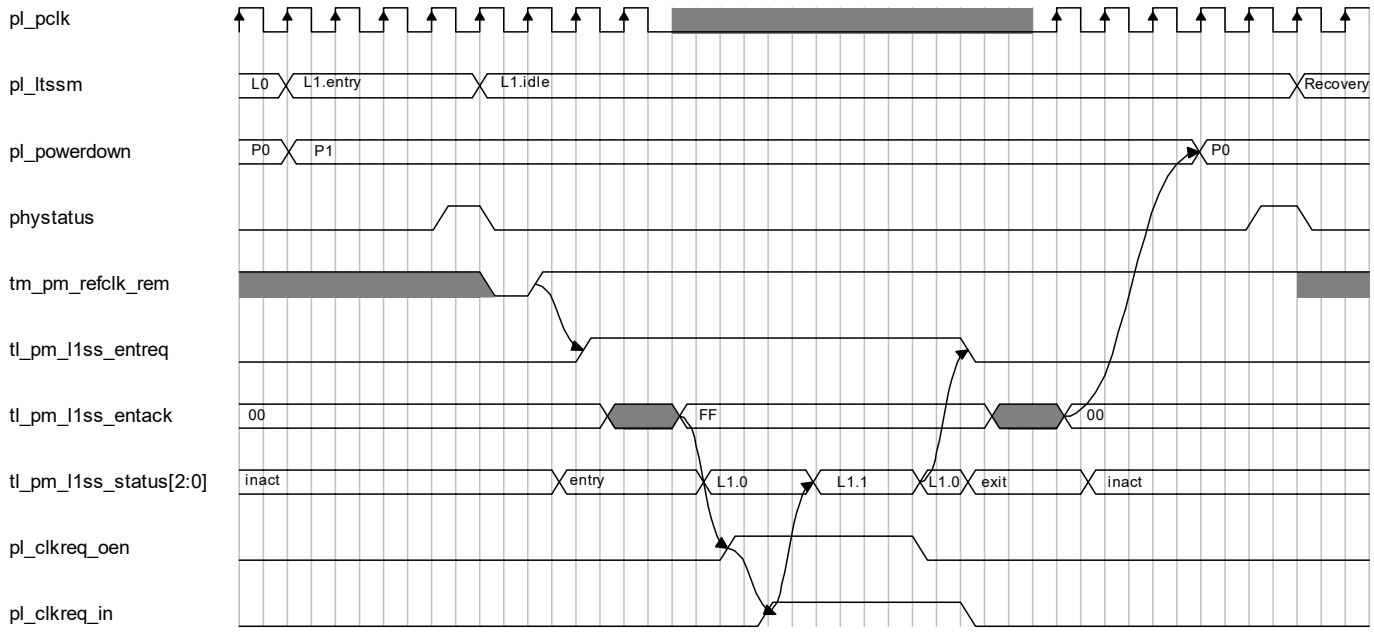


Table 39: L1 Entry with Sub-States

The waveform below shown an entry in L1.0 state, that is not followed by an entry into a sub-state. This can happen if the Core needs to exit L1 before sub-state entry occurs, or if the link partner does not allow sub-state entry:

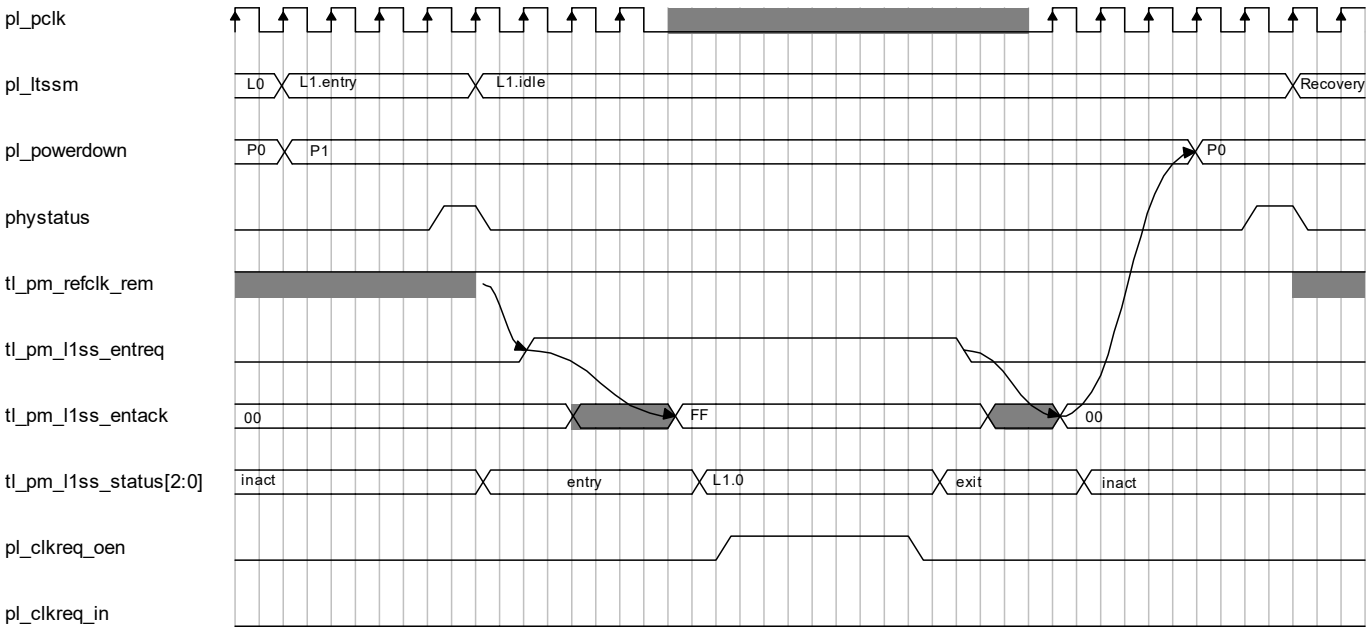


Table 40: L1.0 Entry without Sub-States Entry

Note: The CDC must be implemented when using L1 PM sub-states as the PIPE clock (PL_PCLK) can be stopped when the Core is in the L1.x sub-states. The Transaction Layer clock (TL_CLK) must continue to run, however, even if at a very low frequency, so that any traffic on the Transaction Layer Transmit interface will cause the Core to assert CLKREQ# and exit low-power.

10.3 Optimized Buffer Flush-Fill

Optimized buffer flush-fill (OBFF) is an optional mechanism that enables a host to send its power state information to all devices so that they can align their activity with the host state (for example, by refraining from sending packets when the host is idle so that low-power periods are not interrupted).

The following table describes OBFF signals. All signals are synchronous to the TL_CLK.

Signal	I/O	Description
tl_pm_obffcontrol [3:0]	in	**OBFF Control** (downstream ports only): If OBFF (Optimized Buffer Flash Fill) is enabled, changing the value of this signal causes the Core to signal an OBFF status change to its link partner. Allowed values are: • 1111: CPU Active (default) • 0001: OBFF • 0000: Idle Either WAKE# or message signaling can be used, depending on which signaling method is enabled. If OBFF is not enabled then changing the value of this signal has no effect.
tl_pm_obffstatus [3:0]	out	OBFF Status (upstream ports only): If OBFF is enabled, this signals reports the OBFF status received from the Core's link partner. Possible values are: • 1111: CPU Active • 0001: OBFF • 0000: Idle Either WAKE# or message signaling can be used, depending on which signaling method is enabled. If OBFF is not enabled then the value of this signal is always 'CPU Active'.

Table 41: OBFF signals

10.4 Autonomous Bandwidth Change

The application can direct the Core to change link speed and/or width in order to adjust link bandwidth to its throughput requirements. Although this is not considered a low-power feature, tuning link bandwidth can save significant power.

The following table describes the signal used to manage autonomous link width and speed change:

Signal	I/O	Description
tl_pm_bwchange[7:0]	in	**Bandwidth Change Request:** This signal is used to trigger link width change and indicate which link widths are supported: • Bit 0: Link width change to x2 allowed. • Bit 1: Link width change to x4 allowed. • Bit 2: Link width change to x8 allowed. • Bit 3: Link width change to x16 allowed. • Bit 4: Request link width change: a pulse in LTSSM L0/L0s state requests link width change. • Bits [6:5]: Maximum link speed: 00=2.5GT/s, 01=5.0GT/s, 10=8.0 GT/s • Bit 7: Request link speed change: a pulse in LTSSM L0/L0s state requests link speed change. **Note:** There is no bit to allow x1 because this link width is always supported.

Table 42: Autonomous link width and speed change signal

10.4.1 Autonomous Link Width Change

The application can request a link width change while LTSSM is in either the L0 or L0s state using the signal TL_PM_BWCHANGE. Once this signal is asserted the Core automatically enters the Recovery state to change link width; the change is complete when the LTSSM reverts back to the L0 state.

Note: Link width change is not allowed and ignored if the Hardware Autonomous Width Disable bit is set in the Link Control register (configuration register address 090h bit 9).

Link width up-configuration (changing the link width to a higher link width than in initial link training) is only possible if both link partners support up-configuration. This can be checked with TEST_OUT[94].

Changes to both the Link speed and width should not be requested concurrently.

10.4.2 Autonomous Link Speed Change

The application can request a link speed change while LTSSM is in either the L0 or L0s state using the signal TL_PM_BWCHANGE[7].

Once this signal is asserted the Core automatically enters the Recovery state to change link speed; the change is complete when the LTSSM reverts back to the L0 state.

Note: Link speed change is not allowed and is ignored if the Hardware Autonomous Speed Disable bit is set in the Link Control 2 register.

Changes to both the Link speed and width should not be requested concurrently.

The application should not request a speed change to a speed for which equalization was not performed successfully (this can be checked using the Equalization Phase 3 Successful Configuration bit). If this happens, then the Core will perform a speed change to the closest lowest possible speed.

On a downstream port, it is also possible to change link speed using the APB Configuration interface by following this procedure:

1. Change the Target Link Speed value of the Link Control 2 register (Configuration register address 0B0h bits 3:0) to the desired value.
2. Set Retrain Link bit of Link Control register (configuration register address 090h bit 5) to 1.
3. The Core automatically enters Recovery state to change the link speed; change is complete when LTSSM reverts back to the L0 state.

10.5 Lane Turn-Off

In order to conserve power, the Core turns-off unused lanes by asserting `PL_TXELECIDLE` and `PL_TXCOMPLIANCE` simultaneously, as described in the *PIPE Specification*. This occurs when:

- Lanes disabled by `K_GEN[27:24]` are disabled in Detect.Quiet, immediately after reset.
- Lanes for which no receiver is detected are disabled on Detect.Active->Polling.Active transition.

The following waveform illustrates this behavior for a x8 Core that has lanes 7:4 disabled by `K_GEN[27:24]`, and no receiver detected on lanes 3:2. As a result, lanes 7:2 are turned-off and only lanes 1:0 are trained:

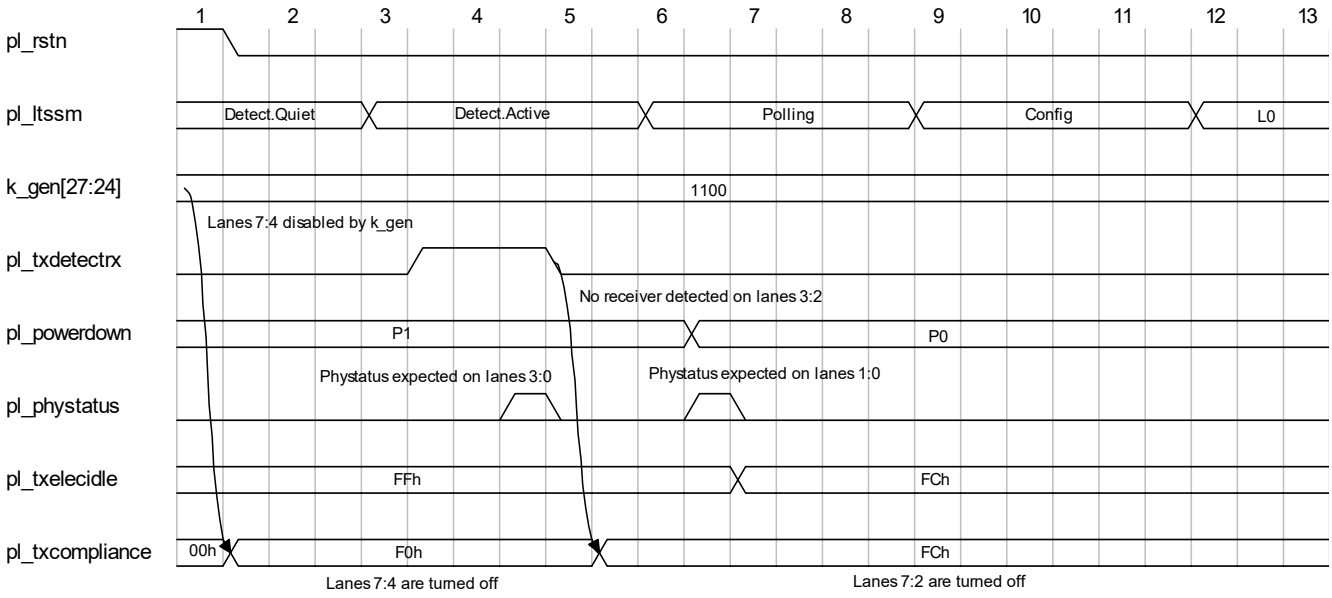


Figure 49: Lane Turn-Off

Chapter 11 Configuration Space Interface

The Configuration Space interface is used by the application to access configuration space registers.

11.1 APB Configuration Interface

This interface is based on the AMBA APB protocol. It enables the application to read any configuration register from a physical function and optionally to write to any writeable register. A separate interface is implemented for each physical function.

Note: You cannot access the Virtual Functions Configurations Register using this interface (see Section 11.2).

Signal	I/O	Description
tl_cfg_paddr0[11:0] (also for functions 1 - 7)	In	**APB Configuration Address**: This signal indicates which configuration register to access.
tl_cfg_penable0 (also for functions 1 - 7)	In	**APB Configuration Enable**: This signal must be asserted in order to start a read/write access operation, and de-asserted when TL_CFG_PREADY0 is asserted, unless another access operation immediately follows it.
tl_cfg_pwrite0 (also for functions 1 - 7)	In	**APB Configuration Write**: This signal is asserted when TL_CFG_PENABLE0 is asserted in order to indicate a write access operation. If this signal is not asserted, the access operation is a read access.
tl_cfg_pwdata0[31:0] (also for functions 1 - 7)	In	**APB Configuration Write Data**: This signal indicates the value to be written to the configuration register.
tl_cfg_pstrb0[3:0] (also for functions 1 - 7)	In	**APB Configuration Write Byte Valid**: This signal is not part of the APB specification. You can use this signal to select which bytes of TL_CFG_PWDATA0 must be written to the configuration space: • Bit 0: write bits [7:0] • Bit 1: write bits [15:8] • Bit 2: write bits [23:16] • Bit 3: write bits [31:24] This signal must not be 0h during a write access; it is not applicable during a read access.
tl_cfg_prdata0[31:0] (also for functions 1 - 7)	Out	**APB Configuration Read Data**: This signal returns the value read from the configuration register. This signal is valid during read access operations when TL_CFG_PREADY0 is asserted.
tl_cfg_pready0 (also for functions 1 - 7)	Out	**APB Configuration Ready**: This signal is asserted for one clock cycle in order to indicate that the read/write access operation is complete. Note that a read/write access operation can take one or more cycles to complete, depending on the activity inside the Core.

Table 43: APB Configuration signals

Note: Writing to read-only registers has no effect and does not cause an error.

The following waveform illustrates a write access operation for the register at address 350h, immediately followed by a read access operation for the register located at address 020h:

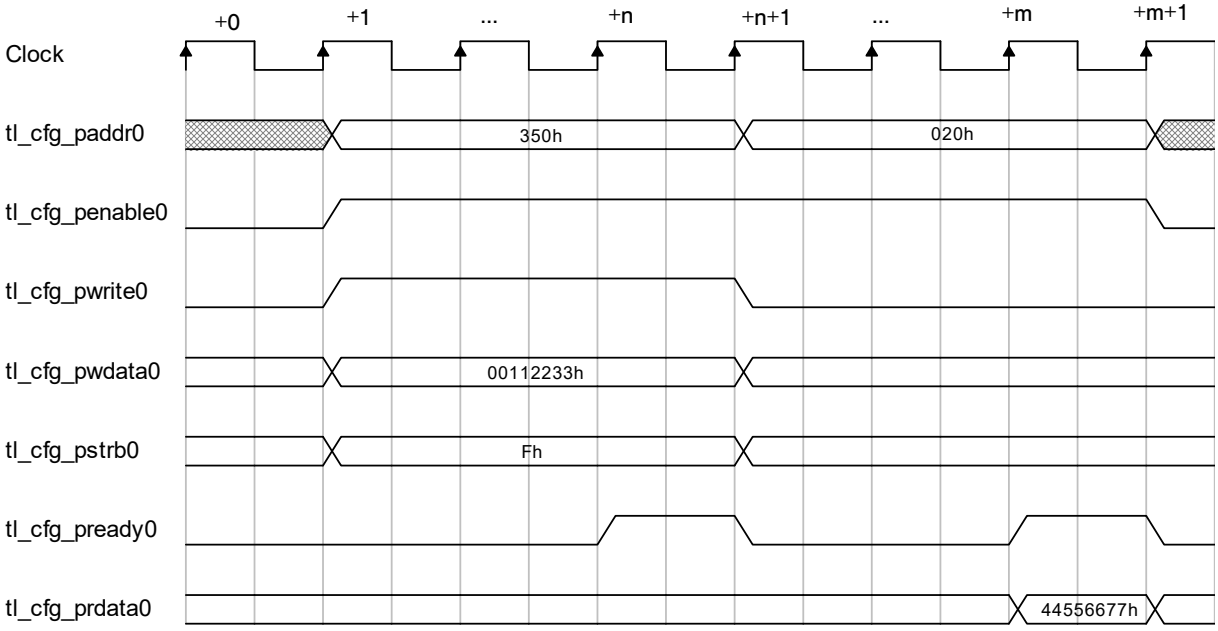


Figure 50: APB Configuration Access Example

11.2 Direct Configuration Access

Key configuration registers are directly available to the application using the following signals. These signals directly reflect the value of the corresponding configuration registers. Signals with name ending with “0” are per-function.

Signal	I/O	Description
tl_cfg_regs0[8191:0] (also for functions 1 - 7)	Out	**Configuration Registers Direct Access**: This signal provides the value of configuration registers from offset 000h to 3FFh. For example, the value of the Maximum Payload Size field of the Device Control Register, which is located in bits 7:5 at offset 088h, can be accessed as shown below: `max_payload = tl_cfg_regs0 ['h088*8+7 : 'h088*8+5];` See 150 Register Content of the Configuration Space for the location of all the registers implemented in the Core.
tl_cfg_vfregs [G_MAX_VF*64:0]	Out	**VF Configuration Registers Direct Access**: This signal provides the value of key per-VF configuration registers. There are 64 bits for each VF, which can be accessed using `TL_CFG_VFREGS[FID*64+63:FID*64]` The bits for each VF are described below: • Bit 0: Master enable • Bit 1: FLR request • Bit 2: MSI enable (0 if MSI capability is not implemented internally) • Bits 5:3: Number of MSI messages enabled (0's if MSI capability is not implemented internally) • Bit 6: MSI-X enable (0 if MSI-X capability is not implemented internally) • Bit 7: MSI-X function mask (0 if MSI-X capability is not implemented internally) • Bits 31:8: reserved • Bits 63:32: MSI mask bits (0's if MSI capability is not implemented internally) **Note:** The bit at index G_MAX_VF*64 is not used.
tl_cfg_busdev[12:0]	Out	**Bus and Device Number**: Reports captured/programmed bus and device numbers: • Bits 12:5: Bus# • Bits 4:0: Device#

Table 44: Direct Access Configuration Registers

11.3 External Registers Interface

This interface enables application-specific configuration registers and capabilities to be implemented in physical function and/or virtual function application-specific register ranges.

This interface also makes it possible to implement registers in a memory, which can lead to significant logic savings, especially when a large number of VFs are implemented. An application can, for example, implement VFs' MSI/MSI-X capability structures in an external memory, thus gaining flexibility as well as saving a lot of register logic.

There are two configuration space ranges available to the application:

- **PCI Configuration Space offset range 040h...07Fh**:
  If one or more capabilities is added to this space, then `K_PCICONF[111:104]` (for PFs) and/or `K_SRIOV[63:56]` (for VFs) must indicate the offset of the first capability structure.

- **PCI Express Extended Configuration Space offset range 800h...FFFh**:
  If one or more extended capabilities is added to this space, then the first capability must be implemented at offset 800h, and `K_PCICONF[103]` (for PFs) and/or `K_SRIOV[47]` (for VFs) must be set to 1.

Signal	I/O	Description
tl_cfg_xaddr[11:0]	Out	**External Register Address**: This signal indicates which configuration register to access. This signal is valid and its value does not change when `TL_XFG_XENABLE` is asserted.
tl_cfg_xfid[9:0]	Out	**External Register FID**: This signal indicates the FID of the targeted function (see Chapter 12). This signal is valid and its value does not change when `TL_XFG_XENABLE` is asserted.
tl_cfg_xenable	Out	**External Register Access Enable**: This signal is asserted when there is a read or write to an application-specific register address. This signal remains asserted until the application replies with `TL_CFG_XREADY`. This signal is deasserted for at least one clock cycle between two separate accesses.
tl_cfg_xwrite	Out	**External Register Write**: This signal is asserted to indicate that the external register access is a write operation. If it is not asserted, then the access is a read. This signal is valid and its value does not change when `TL_XFG_XENABLE` is asserted.
tl_cfg_xwdata[31:0]	Out	**External Register Write Data**: This signal indicates the value to be written to the configuration register. This signal is valid and its value does not change when `TL_XFG_XENABLE` is asserted.
tl_cfg_xstrb[3:0]	Out	**External Register Write Enable**: This signal indicates which bytes of `TL_CFG_XWDATA` must be written when `TL_XFG_XENABLE` is asserted. This signal is valid and its value does not change when `TL_XFG_XENABLE` is asserted. • Bit 0: write bits [7:0] • Bit 1: write bits [15:8] • Bit 2: write bits [23:16] • Bit 3: write bits [31:24]
tl_cfg_xrdata[31:0]	In	**External Register Read Data**: The application returns the value read from a register on this signal at the same time as it asserts `TL_CFG_XREADY`. If access is a write (`TL_CFG_XSTRB`/=0h) then this signal is n/a; if access is a read but the register at the offset indicated by `TL_CFG_XADDR` and/or the function indicated by `TL_CFG_XFID` are not implemented, then the application must return 0's on this signal.
tl_cfg_xready	In	**External Register Access Ready**: The application asserts this signal for one clock cycle after `TL_XFG_XENABLE` in order to report that the read/write has been completed. While there is no `TL_XFG_XENABLE` assertion to `TL_CFG_XREADY` assertion delay requirement, the application should keep this delay to a minimum (the guideline is <20 clock cycles) and predictable, as a large delay can stall the receive datapath.

Table 45: Configuration Expansion Interface Signals

The following waveform illustrates a write access to register 080h of PF #3 (FID=203h), followed by a read of the same register:

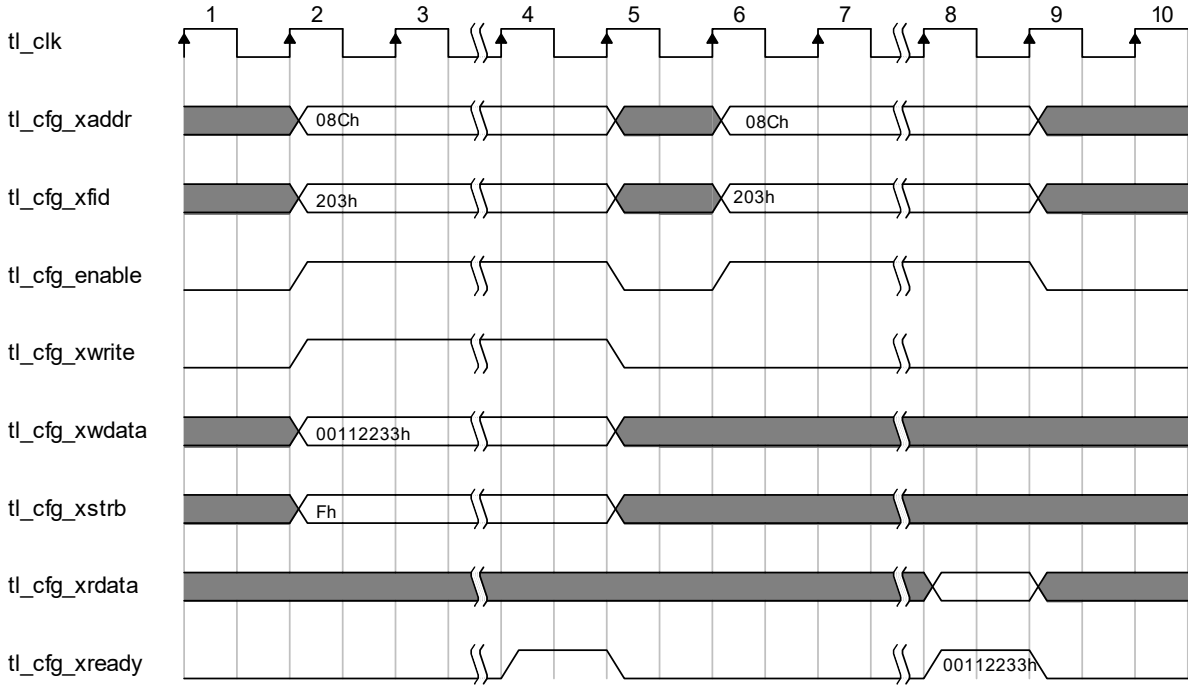


Figure 51: Write Access to Physical Function

The following waveform illustrates a write access to register 090h of VF FID=077h, which is not implemented, then a read access to the same register. As this register/VF is not implemented, the application must ignore the write and return 0s for the read:

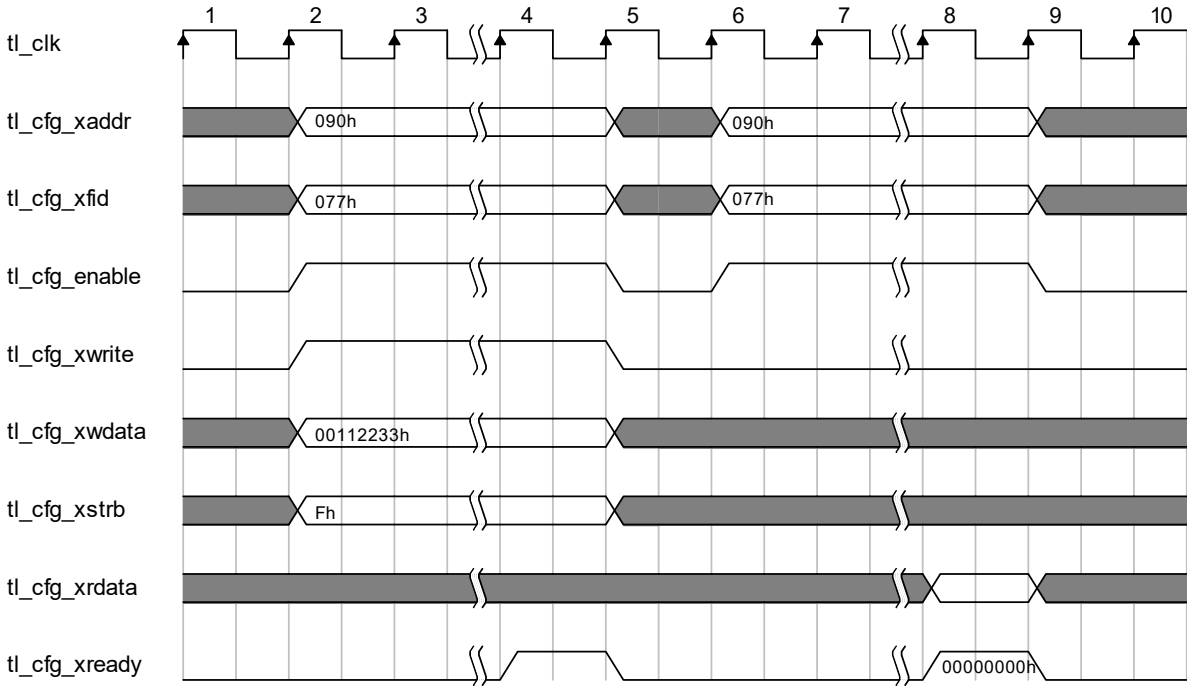


Figure 52: Write Access to Unimplemented Virtual Function

Chapter 12 Implementing SR-IOV

The XpressRICH3 implements Single Root I/O Virtualization (SR-IOV) as follows:

- The Core can implement up to 512 virtual functions, which can be freely split among the implemented physical functions.
- All virtual functions in the same physical function share the same configuration (as defined by K_PCICONF & K_SRIOV), however they each have a separate configuration space.
- In a multi-function device, physical functions are not all required to implement the same number of virtual functions.
- Virtual function migration is not supported, however the application can change the number of VFs assigned to each PF during reset.

12.1 Function ID

Each implemented PF/VF is assigned a unique function ID (FID), whose purpose is to provide a unique identifier for each VF, no matter which PF they are associated with.

- FID[9] is 1 if the function is a PF; 0 if the function is a VF.
- FID[8:0] is 0..G_NUM_FUNC-1 (maximum 7) if the function is a PF, or 0..G_MAX_VF-1 (maximum 511) if the function is a VF

For example:

- FID=207h indicates PF# 7.
- FID=010h indicates the 16th VF#.

Note: If no virtual functions are implemented, then FID[9:3] is always 1000000b.

12.2 VF Assignment to PFs

The parameter G_MAX_VF indicates the maximum number of VFs that are supported by the Core. However, the number of VFs associated to each PF is defined by the K_SRIOV0..7 VF base and VF top fields (bits 275:256), and this assignment can be changed during reset.

VF mapping to PFs must respect these rules:

- VF base/VF top must be in increasing order from PF#0 to PF#7.
- There must be no “hole” in the mapping; however, there can “left-over” VFs not assigned to any PF.

The following example illustrates a possible mapping for a design with 4 PFs and 200 VFs:

PF#	VF Base	VF Top	Number of VFs	VF FID
0	0	8	8	0 - 7
1	8	58	50	8 - 57
2	58	58	0	No VFs implemented
3	58	158	100	58 - 157

Table 46: VF/PF Assignment Mapping Example

Note that in this example VFs with FID=158 - 199 are not associated to any PF.

12.3 Function Number Assignment

A typical PCI Express function is assigned a unique ID that is composed of the Bus number (8 bits), the Device number (5 bits), and the Function number (3 bits).

If Hierarchy is ARI capable then the ID is instead only composed of the Bus number (8 bits) and the Function number (8 bits); and VFs are mapped in up to two additional bus numbers:

Bus #N	Bus #N+1	Bus #N+2
0 - 7: PFs 0 - 7	0 - 255: VF FID 248 - 503	0 - 7: VF FID 504 - 511
8 - 255: VF FID 0 - 247		8 - 255: not used

Table 47: Function Number Assignment when Hierarchy is ARI Capable

When transmitting a TLP, the application can compute Requester or Completer IDs using the formula:

- {BUS/DEV/NUM} = N*256 + VF FID + 8

If Hierarchy is not ARI capable , then VFs cannot be mapped in the base bus#, so are mapped in next one or two bus numbers:

Bus #N	Bus #N+1	Bus #N+2
0 - 7: PFs 0 - 7	0 - 255: VF FID 0 - 255	0 - 7: VF FID 256 - 511
8 - 255: reserved		

Table 48: Function Number Assignment when Hierarchy is not ARI Capable

When transmitting a TLP, the application can compute Requester or Completer IDs using the formula:

- {BUS/DEV/NUM} = N*256 + VF FID + 256

Note:

- N is the base Bus# obtained from `TL_CFG_BUSDEV[12:5]`
- The application must check the SR-IOV “ARI Capable Hierarchy” bit to find out if Hierarchy is ARI capable and properly calculate Requester/Completer IDs. This bit is located in bit 4 at offset 148h in the configuration space of the lowest-number PF that supports VFs (`TL_CFG_REGS[148H*8+4]`).

12.4 Virtual Functions Configuration Space

Virtual functions have a ‘light-weight’ configuration space where only a limited subset of registers are implemented, with the following limitations:

Type 0 configuration header:

- Vendor ID & device ID are set to FFFFFFFFh.
- Status and command registers are implemented but only the master-enable and RWC status register bits.
- Class code and revision ID are the same as for physical functions (configured with `K_PCICONF`).
- Base address/Expansion ROM are not implemented
- Subsystem vendor ID & device ID are configured with `K_SRIOV`.
- Interrupt line and interrupt pin are not implemented.

Capabilities:

- PCI Express capability is implemented as specified in the SR-IOV specification, with the same settings as physical functions (configured with K_PEXCONF).
- MSI capability is implemented with settings common to all VFs (configured with K_PEXCONF.
- MSI-X capability is optionally implemented with the same settings as physical function (configured with K_PCICONF).
- Power management capability is not implemented.
- Application-specific capabilities can be implemented, and configured with K_SRIOV.

Extended capabilities

- No extended capability is implemented
- Application-specific capabilities can be implemented, and configured with K_SRIOV.

12.5 Virtual Function Interrupt Handling

12.5.1 VF Interrupt Interface

The following table describes virtual function interrupt signals. All signals are synchronous to TL_CLK.

Signal	I/O	Description
tl_int_vfmsireq	in	**Virtual Function MSI Request**: This signal is asserted by the application for one clock cycle to request an MSI. The application must wait for TL_INT_VFMSIACK to be asserted before requesting another MSI. No message is transmitted if MSI is not enabled, or if the MSI message number is masked.
tl_int_vfmsinum[4:0]	in	**Virtual Function MSI Message Number**: This signal indicates the MSI message number to use. The application should not request a message number that is not supported and enabled.
tl_int_vfmsifid[9:0]	in	**FID of Requesting VF**: This signal indicates the FID of the virtual function that is requesting this MSI. Note that bit 9 of this signal is always 0 as it is used for VFs only.
tl_int_vfmsiack	out	**Virtual Function MSI Acknowledge**: This signal is asserted by the Core for one clock cycle to acknowledge an MSI request (TL_INT_VFMSIREQ is asserted), even if no message has been sent.

Table 49: VF Interrupt Interface

The following waveform shows an MSI request from VF FID=050h with message number 0, immediately followed by an MSI request from VF FID=0F5h with message number 7:

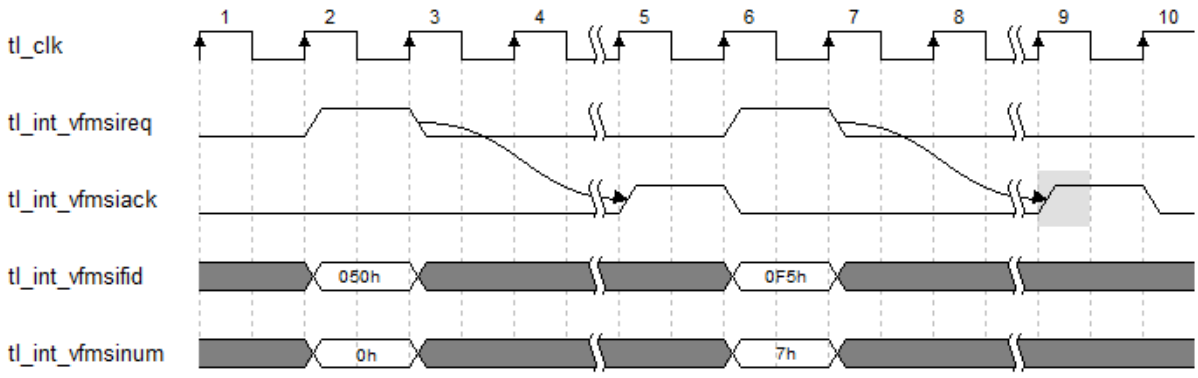


Figure 53: MSI requests

12.5.2 Per-Vector Masking

Per-vector masking is a mandatory feature for both PFs and VFs when SR-IOV is supported. Virtual functions' MSI capability Mask Bits and Pending Bits registers are controlled as follows:

- The Mask Bits register is programmed by the host and its value can be accessed by the application using TL_CFG_VFREGS.
- Pending bits must be implemented and maintained by the application, which must report their value on TL_REPORT_VFSTATE.

It is the application's responsibility to resend pending MSI when the MSI message number is unmasked.

12.5.3 External Interrupt Mode

External interrupt mode enables the application to send MSI messages rather than relying on the Core to do so. This special mode is enabled for VFs with K_GEN[38]. When it is enabled, the application no longer uses TL_INT_VFMSI... interface signals to request MSI, but sends MSI messages using the Core's TX interface instead.

Note: You can also implement MSI and/or MSI-X capabilities externally (see K_PEXCONF bits 182:181) in this mode, so that the application has full control of interrupt processing.

Chapter 13 Reporting Interface

13.1 Reporting Interface Signals

The Reporting interface is used to report request status and errors to the Core.

Signal	I/O	Description
tl_report_state0[63:0] (also for functions 1-7)	in	**Function State Reporting**: All the bits in this signal are level-sensitive and used to report the state of the physical function: • Bit 0: Completion pending • Bit 1: Physical function configuration not ready: devices that need time to load configuration settings must assert this signal until all physical function configuration settings are properly set. The Core replies to all physical function configuration requests with Configuration Retry Status completions while this signal is asserted. • Bit 2: Virtual function configuration not ready: devices that need time to load configuration settings must assert this signal until all virtual function configuration settings are properly set. The Core replies to all virtual function configuration requests with Configuration Retry Status completions while this signal is asserted. • Bit 3: Page Request Interface running: the application is issuing page requests or waiting for page request completions. • Bit 4: Margining ready: indicates that the margining feature is ready to accept margining commands. This bit is only available for function #0. For other functions, this bit is not available and should be tied to 0. • Bit 5: Margining software ready: indicates that the required software has performed the required initialization. This bit is only available for function #0. For other functions, this bit is not available and should be tied to 0.This bit should also be tied to 0 if no margining software is required. • Bit 6: Trigger FLR from application (must be asserted for one clock cycle). • Bit 7: FLR acknowledge (must be asserted for one clock cycle). • Bit 8: CCIX ESM Calibration complete: this bit must be set by the application after the ESM calibration step during L1 LTSSM and cleared after reset. This bit applies to Function #0 only and must be set to 0 for all other functions. • Bits 31:89: reserved • Bits 63:32: MSI pending bits: indicates the value of the pending bits register for MSI capability. These bits should be tied to 0 if MSI capability is not implemented internally or if MSI per-vector masking is not supported for this PF.
tl_report_vfstate [G_MAX_VF*64:0]]	in	**Per-VFs State Reporting**: All the bits in this signal are level-sensitive and used to report the state of virtual functions. There are 64 bits for each VF, which can be accessed using `TL_REPORT_VFSTATE[ FID*64+63:FID*64]` The bits for each VF are described below: • Bit 0: Completion pending • Bits 5:1: reserved • Bit 6: Trigger FLR from application (must be asserted for one clock cycle) • Bit 7: FLR acknowledge (must be asserted for one clock cycle) • Bits 31:8: reserved • Bits 63:32: MSI pending bits: indicates the value of the pending bits register for MSI capability. These bits should be tied to 0 if MSI capability is not implemented internally for this VF. **Note:** Bits corresponding to unimplemented VFs must be tied to 0. The bit at index G_MAX_VF*64 is not used and must also be tied to 0.

Table 50: Reporting signals

Signal	I/O	Description
tl_report_error[44:0]	in	**Function Error Reporting**: This signal is used to report errors detected by the physical function and related virtual functions. Bits 31:0 indicate the reason for the error. Each bit in this field is pulsed for one clock cycle to report an error. When one of these bits is asserted, bits [44:32] indicates which PF/VF reports the error. • Bit 0: reserved • Bit 1: Completion timeout occurred and a recovery mechanism is implemented.The header is logged if K_PEXCONF[192]=1, otherwise a zeroed header is recorded. • Bit 2: Completion timeout occurred and no recovery mechanism is implemented. The header is logged if K_PEXCONF[192]=1, otherwise a zeroed header is recorded. • Bit 3: Unexpected completion received by the application but no ECRC error occurred. • Bit 4: Malformed completion received (a completion with CRS status is received for a non-configuration request, or completion with more than 1 DWORD payload is received for an IO or configuration request) but no ECRC error occurred. • Bit 5: Aborted non-posted request but no ECRC error occurred. • Bit 6: Unsupported non-posted request but no ECRC error occurred and the TLP is not error-poisoned. • Bit 7: Unsupported posted request but no ECRC error occurred and the TLP is not error-poisoned. **Note:** If AER is enabled and the application needs to log the header then it can do so using the APB Configuration interface. • Bit 8: reserved • Bit 9: Page Request Interface response failure. • Bit 10: Page Request Interface unexpected page request group index error. • Bit 11: reserved • Bit 12: Atomic Egress blocked: application has blocked transmission of an atomic operation TLP. • Bit 14:13: reserved • Bit 15: End-End TLP Prefix blocked • Bit 16: ACS violation • Bit 17: Uncorrectable Internal Error • Bit 18: Corrected Internal Error • Bit 19: MC Block TLP Error: transmission of a multicast TLP has been blocked by the application because of the MC_Block_All or MC_Block_Untranslated mechanism (Endpoint devices only). • Bit 20: Poisoned TLP Egress Blocked Error: application has blocked transmission of a poisoned TLP, in conjunction with Downstream Port Containment. • Bit 21: Switch downstream port received a poisoned Config TLP: switching logic indicates that a poisoned config TLP was targeting the switch downstream port. • Bits 31:22: reserved • Bits 34:32: PF# of reporting function. If a VF is reporting then this must indicate the associated PF. • Bits 44:35: FID of reporting function (see Chapter 12).
tl_report_header[255:0]	In	**Report Header**: Header of the transaction for which the application is reporting the error signaled by TL_REPORT_ERROR. The format is the same as TL_RX_DATA0. If End-End TLP Prefixes are present then they must appear before the header. A header is logged for each error as indicated in TL_REPORT_ERROR description; so in all of these cases, the application must provide the header of the TLP which caused the error. A header is logged in the following cases: • TL_REPORT_ERROR bits 1 or 2 set and K_PEXCONF[192]=1. • TL_REPORT_ERROR bits 3, 4, 5, 6, 7, 12, 15, 16, 17, 19, 20 or 21 are set. In all of these cases, the application must provide a header. However, if no header is available or applicable, then the application should report a value of all 1s.

Table 50: Reporting signals

Signal	I/O	Description
tl_report_timer[3:0]	Out	**Timer**: The Core outputs a pulse signal at a fixed frequency, regardless of link speed or application clock speed, which can be used to drive the counters in the application. • Bit 0: 1us pulse • Bit 1: 1ms pulse • Bits 3:2: reserved
tl_report_event[7:0]	Out	**Report Event**: • Bit 0: Power Management Event interrupt (Rootport only): This signal is asserted when a power management event occurs and remains asserted until the interrupt source register has been cleared. Note that PME Interrupt Enable must be set in the PCIe Root Control Register to enable an interrupt, and that the PME Status bit must be cleared in the PCIe Status Control Register after each interrupt. • Bit 1: AER Error (Rootport only): This signal is asserted when an AER error occurs and remains asserted until the interrupt source register has been cleared. • Bit 2: System Error (Rootport only): This signal is asserted for one clock cycle when a system error, as defined by the *PCI Express Specifications*, occurs. • Bit 3: Hotplug Event interrupt (Rootport and switch downstream only): This signal is asserted when a hotplug event occurs and remains asserted until the interrupt source register has been cleared. Note that Hotplug Interrupt Enable must be set in the PCIe Slot Control Register to enable an interrupt. • Bit 4: Link equalization request (Rootport and switch downstream): This signal is asserted when a request to re-execute equalization is received from the upstream device, and remains asserted until the interrupt source register has been cleared. **Note:** The Link Equalization Request Interrupt Enable bit must be set in Link Control 3 register to enable this interrupt. • Bit 5: DPC interrupt (Rootport and switch downstream only): This signal is asserted when a DPC interrupt is triggered and remains asserted until the corresponding interrupt source bit is cleared. • Bit 6: Link Bandwidth Management event: this signal is asserted when link speed or width has been changed for a non-autonomous reason. • Bit 7: Link Autonomous Bandwidth event: this signal is asserted when link speed or width has been changed autonomously.

tl_report_hotplug[7:0]	In	**Hot Plug Status**: This signal reports the status of on-board hotplug components. It is only applicable to downstream ports that implement slot registers in PCI Express.
		• Bit 0: Attention Button Pressed: This signal is asserted for one or more clock cycles when the attention button is pressed. If no attention button exists on the component, this bit should be hardwired to 0.
		• Bit 1: Presence Detect: This signal is set when a component presence is detected, and cleared if no presence is detected.
		• Bit 2: MRL Sensor: This signal is asserted when an MRL sensor detects that a component is being removed from the fabric and should be powered down. Both the rising and falling edge of this signal are detected in order to set the corresponding hot-plug status register.
		• Bit 3: Power Fault detected: This signal is asserted for one or more clock cycles when a power fault is detected on this slot.
		• Bit 4: Power Controller Status: This signal reports the status of the power controller:
		  • 0: Power on
		  • 1: Power off
		• Bit 5: Electromechanical Interlock Status: This signal reports the status of the electromechanical interlock, if present:
		  • 0: Electromechanical Interlock Disengaged
		  • 1: Electromechanical Interlock Engaged
		• Bits 7:6: reserved

Table 50: Reporting signals

Signal	I/O	Description
tl_report_latency[32:0]	in	Latency Tolerance Reporting values: • Bits 9:0: Snoop latency value • Bits 12:10: Snoop latency scale • Bit 15: Snoop latency requirement • Bits 25:16: No-snoop latency value • Bits 28:26: No-snoop latency scale • Bit 31: No-snoop latency requirement • Bit 32: When Latency Tolerance Reporting is implemented and enabled, asserting this bit for one clock cycle causes the Core to transmit an LTR message with the data specified by bits 31:0. See the *PCI Express Specification* for more information. **Note**: The application must provide correct latency tolerance values on this signal at all times when Latency Tolerance Reporting is implemented. The Core uses these values to send LTR messages and to manage internal L1 PM sub-states.

Table 50: Reporting signals

13.2 Latency Tolerance Reporting

Latency Tolerance Reporting (LTR) is an optional feature that allows an endpoint device to autonomously report its tolerance of memory read/write latencies; this allows the system to fine tune its power management policies.

The Core automatically transmits LTR messages with the value indicated by TL_REPORT_LATENCY when:

- the LTR Mechanism Enable bit is set
- LTR is enabled

The Core automatically transmits LTR messages with all requirement bits cleared when:

- the LTR Mechanism Enable bit is cleared, if the last transmitted LTR message has requirement bit(s) set.
- the Device is directed to a non-D0 legacy power state, if the last transmitted LTR message has requirement bit(s) set.

The following waveform shows an example of LTR message transmission. Note that the 'No Requirement' message is sent only if there were requirement bit(s) set in the 'Latency2' message:

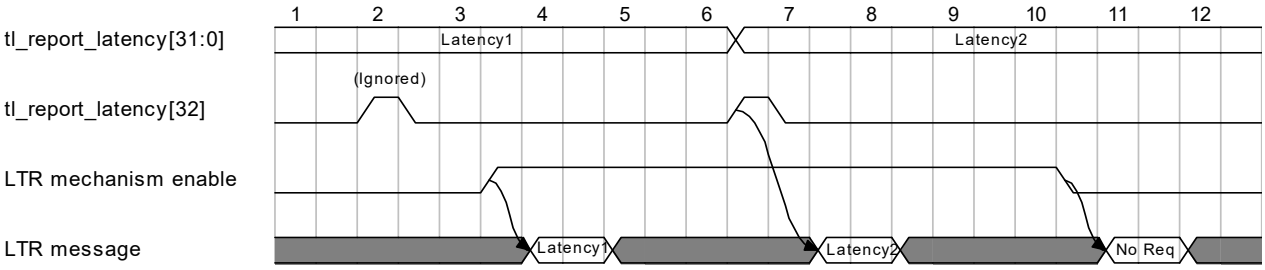


Figure 54: LTR message transmission

13.3 Access Control Services

Access Control Services (ACS) are used to control which devices are allowed to communicate with one another, in order to avoid the improper routing of packets. It is especially useful for switch/rootports, and when Address Translation Services (ATS) are supported.

ACS implementation depends on the device type:

- **Downstream ports**:
  The Core performs ACS Source Validation and ACS Translation Blocking checks. All other checks are related to device routing rules between different ports, and should therefore be implemented in application/switching logic, if required. This logic must report any ACS violations detected using the signals TL_REPORT_ERROR0 and TL_REPORT_HEADER.
- **Multi-function devices (endpoints)**:
  The Core does not perform any internal checks. Internal peer-to-peer between functions is not supported in the Core; all transmitted TLPs are sent to the PCI Express link, so ACS checks are not applicable.

The Core supports egress control vectors with up to 32bits, which means that up to 32 ports can be supported.

13.4 Precision Time Measurement

Precision Time Measurement is an optional feature that enables a protocol for timing measurement/synchronization messages. This feature is configured with k_pexconf (see Section 4.2) and PTM Extended Capability is implemented when it is enabled. The Core does not generate or consume PTM messages and it is the application's responsibility to transmit and process received PTM messages as necessary.

Chapter 14 Test Interface

14.1 Test Input Port

The table below describes the TEST_IN signal of the test interface:

Signal	Description
test_in[63:0]	• Bit [0]: **Simulation Mode**: This signal must be set to 1 to set simulation mode (see Section 14.2). • Bit [1]: **Disable Low Power State Negotiation**: When asserted, this signal disables all low power state negotiation. • Bit [2]: **Loopback Master**: This signal must be set to 1 to direct the Link to loopback (in Master mode). • Bit [3]: Enable warning assertions. • Bit [4]: Enable information assertions. • Bit [5]: reserved • Bit [6]: Disable scrambling (Gen1/Gen2 modes only). • Bit [7]: Set compliance receive bit in transmitted TS1 ordered set. • Bit [8]: reserved (was Selectable deemphasis for upstream devices, now replaced by K_PEXCONF[174]). • Bit [9]: Disable entry in Polling.Compliance from Polling.active (does not apply if the TS1 compliance receive bit is set). • Bit [10]: Force entry in Polling.Compliance from Polling.active. • Bit [11]: Force receiver detection on all implemented lanes in Detect.Active • Bits [13:12]: reserved • Bit [14]: reserved (was 'Disable PhyStatus timeout' until v175) • Bit [15]: reserved • Bits [17:16]: reserved • Bit [18]: Enables all nullified packets sent from the TX interface to be transmitted to the PCIe link (by default they are removed from the TX buffer automatically except when TX streaming is active). • Bit [19]: Extended simulation mode (see Section 14.2) • Bit [20]: Enable warnings when incorrect DC balance symbols are received in training sets. • Bit [21]: Disable 128b130b SKP OS parity checking & reporting. • Bit [22]: reserved (was 8GT/s Equalization problem until v192) • Bits [63:23]: reserved

Table 51: Test interface (test_in)

Note: These signals should only be modified when the Core is in reset mode. Do not modify these signals if reset is not active as this may cause a malfunction of the Core.

These signals must be considered as static for timing and clock domain crossing analysis for Core parameters (K_... signals) and TEST_IN.

14.2 Simulation Mode

Simulation mode is a special mode where some internal delays and parameters are adjusted in order to accelerate link training and to test certain features within a reasonable simulation time.

Note: When simulating with a PHY model, simulation mode delays may not be long enough, especially for receiver detection and power/rate changes. This typically causes unexpected timeouts to occur and can disrupt simulation. In this case, we recommend using “extended simulation mode'”.

The table below shows which delays are affected in simulation mode:

Parameter	Normal Mode test_in[0]=0	Simulation Mode test_in[0]=1, test_in[19]=0	Extended Simulation Mode test_in[0]=1, test_in[19]=1
Detect.Quiet timeout	12 ms	6 µs	6 µs
Detect.Active timeout	12 ms	6 µs	6 µs
Polling.Active timeout	24 ms	40 µs	240 µs
Polling.Config timeout	48 ms	48 µs	480 µs
Config.LinkWidthStart timeout	24 ms	240 µs	240 µs
Config.LinkWidthStart crosslink timeout	random 2...9*100 us	random 2...9*2 us	random 2...9*2 us
Config.LinkWidthStart max delay for upconfiguration (upstream)	1 ms	10 µs	10 µs
Config.LinkWidthStart min delay for upconfiguration (upstream)	0.9 ms	9µs	9µs
Config.LinkWidthAccept timeout	2 ms	20 µs	20 µs
Config.LaneNumWait timeout	2 ms	20 µs	20 µs
Config.LaneNumAccept timeout	24 ms	24 µs	24 µs
Config.Complete timeout	2 ms	20 µs	20 µs
Config.Idle timeout	2 ms	20 µs	20 µs
Recovery.ReceiverLock timeout	24 ms	48 µs	240 µs
Recovery.Equalization phases 0,1,3 timeout (downstream port)	24 ms	240 µs	240 µs
Recovery.Equalization phase 2 timeout (downstream port)	32 ms	320 µs	320 µs
Recovery.Equalization phases 0,1 timeout (upstream port)	12 ms	120 µs	240 µs
Recovery.Equalization phases 2 timeout (upstream port)	24 ms	240 µs	240 µs
Recovery.Equalization phases 3 timeout (upstream port)	32 ms	320 µs	320 µs
Recovery.Speed timeout	48 ms	480 µs	480 µs
Recovery.ReceiverConfig timeout	48 ms	480 µs	480 µs
Recovery.Idle timeout	2 ms	20 µs	20 µs
Disable timeout	2 ms	3 µs	3 µs

Table 52: Simulation mode

Parameter	Normal Mode test_in[0]=0	Simulation Mode test_in[0]=1, test_in[19]=0	Extended Simulation Mode test_in[0]=1, test_in[19]=1
Loopback.Entry timeout (to Loopback.Exit)	51 ms	50 µs	50 µs
Loopback.Entry timeout (to Loopback.Active)	2 ms	2 µs	2 µs
Loopback.Exit timeout	2 ms	20 µs	20 µs
Hot Reset timeout	2 ms	20 µs	20 µs
Equalization Remote transmitter adjustment max delay	20 ms	200 µs	200 µs
Equalization Remote transmitter adjustment request max delay to wait for remote device response	2 ms	20 µs	20 µs
Min delay before changing speed in Detect.Quiet	1 ms	6 µs	6 µs
Min time in Electrical Idle when changing speed in Polling.Compliance	1 ms	6 µs	6 µs
Additional time in Electrical Idle in Recovery.Speed	6 - 8 µs	6 - 8 µs	6 - 8 µs
Min time in Electrical Idle when changing speed in Loopback.Entry (if Core is loopback master)	1 ms	6 µs	6 µs
Min time in Electrical Idle when changing speed in Loopback.Entry (if Core is loopback slave)	2 ms	13 µs	13 µs
Flow-Control timeout	224 µs	40 µs	40 µs
Electrical Idle Inference in Recovery.Rcfg at 8 GT/s	4 ms	40 µs	40 µs
Idle_to_rlock_transitioned variable max value	255	7	7
Delay to enter L1 from L0 when all functions are in a non-D0 power state	8 ms	8 µs	8 µs

Table 52: Simulation mode

Note: In simulation mode, when the Core is in the Polling.active state, it transmits only 32 TS1s instead of 1024.

14.3 Test Output Port

The table below describes the TEST_OUT and TEST_OUT_EQU signals of the test output interface:

Signal	Clock Domain	Description
test_out[511:0]		
Transaction Layer - Tx		
test_out[61:0]	tl_clk	reserved (was transmit credit information until v175, this information is now available via TL_RX_CREDITS0)
test_out[62]		TX buffer overflow.
test_out[63]		Internal packet transmission request
Transaction Layer - Link Status & Low-Power		
test_out[89:64]	tl_clk	reserved
test_out[90]		L1.2 enabled
test_out[91]		L1.1 enabled
test_out[92]		ASPM L1.2 enabled
test_out[93]		ASPM L1.1 enabled
test_out[94]		Link up-configuration capability
test_out[95]		Port is upstream (this signal reports the current port direction when crosslinks are supported).
test_out[100:96]		LTSSM state (equivalent to PL_LTSSM but in the tl_ck domain).
test_out[101]		Data link layer is active.
test_out[102]		TLP transmission inhibited
test_out[103]		DLLP transmission inhibited
test_out[107:104]		Low-power state machine state: • 0h: L0_RST (Initialization state) • 1h: L0 (Idle state) • 2h: ASPM_L1_IN0 (Prepare ASPM L1 entry step #1) • 3h: ASPM_L1_IN1 (Prepare ASPM L1 entry step #2) • 4h: ASPM_L1_OUT (Exit from ASPM L1 because of NAK) • 5h: L1_IN0 (Prepare L1 entry step #1) • 6h: L1_IN1 (Prepare L1 entry step #2) • 7h: L1 (L1 state) • 8h: L0_IN (Prepare L0 entry) • 9h: L0_IN_WAIT (Wait for L0 entry) • Ah: L2_IN0 (Prepare L2 entry step #1) • Bh: L2_IN1 (Prepare L2 entry step #2) • Ch: L2 (L2 state)
test_out[108]		Transaction layer directs LTSSM to enter/stay in ASPM L0s.
test_out[109]		Transaction layer directs LTSSM to enter/stay in L1 or ASPM L1.
test_out[110]		Transaction layer directs LTSSM to enter/stay in L2.
test_out[111]	tl_clk	Downstream Port Containment is active and directing LTSSM to enter/stay in Disabled state.

Table 53: Test Output Interface

Signal	Clock Domain	Description
Transaction Layer - Rx		
test_out[112]	tl_clk	TLP header received
test_out[113]		TLP is malformed IO/Cfg
test_out[114]		TLP is malformed message
test_out[115]		TLP is malformed completion
test_out[116]		TLP is malformed MemRdLkTLP
test_out[117]		TLP type is unknown
test_out[118]		TLP FBE/LBE are incorrect
test_out[119]		TLP payload size is incorrect
test_out[120]		TLP crosses 4KB address boundary
test_out[121]		TLP is malformed atomic operation
test_out[122]		TLP is unexpected completion
test_out[123]		TLP address is not in any BAR
test_out[124]		TLP is unsupported
test_out[125]		ACS violation detection with TLP
test_out[127:126]		reserved
test_out[255:128]		Received TLP header
PHYMAC Layer		
test_out[257:256]	pl_pclk	RxL0s sub-state: • 0h: RxL0s.Inactive • 1h: RxL0s.Idle • 2h: RxL0s.FTS • 3h: RxL0s.OutRecovery
test_out[259:258]		TxL0s sub-state • 0h: TxL0s.Inactive • 1h: TxL0s.Idle • 2h: TxL0s.FTS • 3h: TxL0s.OutL0
test_out[261:260]		Equalization current phase
test_out[262]		Indicates that lane(s) are reversed.
test_out[263]		Timeout occurred in current LTSSM state.
test_out[264]		Indicates that a speed change is in progress.
test_out[265]		Indicates that the Core is directing the link to Recovery.
test_out[266]		LTSSM state change on next clock cycle: this signal is asserted for one clock cycle to indicate that LTSSM is going to change on the next clock cycle. This can typically be used to synchronize a logic analyzer.

Table 53: Test Output Interface

Signal	Clock Domain	Description
test_out[267]	pl_pclk	LTSSM State/Equalization phase change: this signal is asserted for one clock to indicate that the LTSSM state or Equalization phase has changed. This signal can typically be used to synchronize a logic analyzer.
test_out[268]		Port is upstream (this signal reports the current port direction when crosslinks are supported).
test_out[269]		8b/10b or 128/130b encoding error, or malformed SKP OS error detected
test_out[270]		PhyStatus error (LTSSM state changed before receiver detection/power state/speed was acknowledged by the PHY).
test_out[271]		Link is up
test_out[279:272]		Lane 7:0 detected: each bit indicates if a lane was detected during Detect.Active. Information about lanes 15:8 is not available in test_out.
test_out[287:280]		Lane 7:0 active: each bit indicates if a lane is currently part of a configured link. Information about lanes 15:8 is not available in test_out.
test_out[293:288]		Polling.Compliance setting #
test_out[319:294]		reserved
test_out[320]		Deskew Error
test_out[321]		Framing error (any reason). In 8b10b this can be caused by a misplaced STP/SDP or a DLLP without an END symbol.
test_out[322]		128b/130b framing error detected: incorrect RxSyncHeader or SDS followed by another ordered set).
test_out[323]		128b/130b framing error detected: received STP or EDS with incorrect length/framing parity/framing CRC.
test_out[324]		128b/130b framing error detected: incomplete EDB symbol.
test_out[325]		128b/130b framing error detected: unexpected symbol/token.
test_out[326]		128b/130b framing error detected: unexpected EDS or OS received without preceding EDS.
test_out[383:327]		reserved

Table 53: Test Output Interface

Signal	Clock Domain	Description
Data Link Layer - Rx		
test_out[395:384]	pl_pclk	Sequence number of last acknowledged received TLP
test_out[396]		Received good TLP
test_out[397]		Received bad TLP
test_out[398]		Received duplicate TLP
test_out[399]		Rx buffer overflow
test_out[400]		ACK sent (sequence number is contained in TEST_OUT[395:384])
test_out[401]		NAK sent (sequence number is contained in TEST_OUT[395:384])
test_out[402]		Received nullified TLP
test_out[407:403]		RX buffer fill level (00000: empty/nearly empty,. 11111: full/nearly full) Information is not always accurate because of CDC latency and when in Rx stream mode with small buffer size (especially when G_RXBUF_SIZE=8).
test_out[408]		Received bad DLLP
test_out[409]		Flow-control protocol error
test_out[410]		Flow-control timeout
test_out[415:411]		reserved
Data Link Layer - Tx		
test_out[427:416]	pl_pclk	Sequence number of last acknowledged transmitted TLP
test_out[439:428]		Highest transmitted sequence number
test_out[440]		Replay in progress
test_out[441]		Replay number error
test_out[442]		Replay timer error
test_out[443]		NAK received (with correct sequence number)
test_out[444]		ACK/NAK received with incorrect sequence number
test_out[445]		Replay buffer overflow (this indicates an internal error in the Core).
test_out[446]		Parity error detected in TLP read from TX buffer.
test_out[447]		ECC error detected when reading TLP from TX buffer.
test_out[511:448]		reserved

Table 53: Test Output Interface

Signal	Clock Domain	Description
test_out_equ [G_NUM_LANES*46]	pl_pclk	This signal reports the equalization settings accepted by the (local) Core and by the (remote) link partner at current link speed. It is only valid in Recovery.RcvrLock when this state was entered from Recovery.Equalization. There are 46 bits for each implemented lane: • [17:0]: Remote coefficients {C+1,C0,C-1} • [21:18]: Remote Preset • [22]: Remote Use Preset (1: preset is used, 0: coefficients are used) • [40:23]: Local coefficients {C+1,C0,C-1} • [44:41]: Local Preset • [45]: Local Use Preset (1: preset is used, 0: coefficients are used)

Table 53: Test Output Interface

Chapter 15 Bridge/Switch Interface

The Bridge/Switch interface is used for communication between the upstream and downstream ports of Bridges and Switches.

Signal	I/O	Description

tl_brsw_in[7:0]	In	**Bridge/Switch Input**:
		• Bit 0: Switch downstream: a rising edge of this signal should cause LTSSM to transition to HotReset, and remain in this state as long as this bit is asserted. (other: reserved)
		• Bit 1:
		  • Switch upstream: level sensitive, indicates that the upstream port can enter L1/ ASPM L1. This bit must be asserted once all downstream ports are in L1/ASPM L1 and all upstream traffic has been transmitted. It must be de-asserted as soon as one of the downstream ports exits L1/ASPM L1.
		  • Switch downstream: level sensitive, indicates that upstream port is in L1/ASPM L1
		  • Other: reserved
		• Bit 2:
		  • Switch upstream: a pulse indicates L0s exit on downstream port
		  • Switch downstream: a pulse indicates L0s exit on upstream port
		  • Other: reserved
		• Bit 3: Switch upstream: a pulse indicates that a wake-up event occurred (beacon received or WAKE# asserted) in one of the downstream ports (other: reserved)
		• Bit 4: Bridge forward (PCIe-PCI(-X) bridge): level sensitive: indicates that PME# pin is asserted on secondary side (other: reserved)
		• Bit 5: Bridge forward (PCIe-PCI(-X) bridge): a pulse asserts “secondary discard timer expired bit” (other: reserved)
		• Bit 6: Allow L0s entry:
		  • Switch upstream: level sensitive, indicates that none of the switch’s downstream ports are in L0, Recovery, or Configuration states; and that there are no pending TLPs in switching logic.
		  • Switch downstream: level sensitive, indicates that switch’s upstream port is not in L0, Recovery, or Configuration states; and that there are no pending TLPs in switching logic.
		  • Other: reserved
		• Bit 7: reserved

tl_brsw_out[7:0]	Out	**Bridge/Switch Output**:
		• Bit 0: Switch upstream and bridge forward (PCIe-PCI(-X) bridge): this bit is asserted when LTSSM is in the HotReset state; it is pulsed when the Secondary Bus Reset register is asserted. (other: reserved)
		• Bit 1: Switch upstream & switch downstream: level sensitive, indicates that port is in L1 (other: reserved)
		• Bit 2: Switch upstream & downstream: a pulse indicates L0s exit (other: reserved)
		• Bit 3: Switch downstream: a pulse indicates that a wake-up event occurred (beacon received or WAKE# asserted) (other: reserved)
		• Bit 4:5: reserved
		• Bit 6: Port idle indicator:
		  • Switch upstream and switch downstream: level sensitive; reports that LTSSM is not in L0, Recovery, or Configuration states.
		  • Other: reserved
		• Bits 7: reserved

Table 54: Bridge/Switch signals

Chapter 16 LTSSM Interface

The LTSSM signal in the Data Link Layer interface reports the current LTSSM state.

Signal	I/O	Description
pl_ltssm_enable	In	**Enable LTSSM**: This signal should be set to 1 in normal conditions; however, it can be set to 0 at power-up to prevent LTSSM from exiting Detect.Quiet for as long as this signal is asserted. This is typically used to prevent LTSSM from moving to another state when the PHY or PIPE interface is not ready. This signal has no effect on LTSSM states other than Detect.Quiet.
pl_ltssm[4:0]	Out	**LTSSM state**: LTSSM state encoding: • 00h: detect.quiet • 01h: detect.active • 02h: polling.active • 03h: polling.compliance • 04h: polling.configuration • 05h: config.linkwidthstart • 06h: config.linkwidthaccept • 07h: config.lanenumwait • 08h: config.lanenumaccept • 09h: config.complete • 0Ah: config.idle • 0Bh: recovery.receiverlock • 0Ch: recovery.equalization • 0Dh: recovery.speed • 0Eh: recovery.receiverconfig • 0Fh: recovery.idle • 10h: L0 • 11h: L0s • 12h: L1.entry • 13h: L1.idle • 14h: L2.idle/L2.transmitwake • 15h: reserved • 16h: disable • 17h: loopback.entry • 18h: loopback.active • 19h: loopback.exit • 1Ah: hotreset **Note:** This signal is clocked in the PL_PCLK clock domain: the same information is available for the TL_CLK clock domain via the TEST_OUT[100:96] signal.
pl_equ_phase[1:0]	Out	**Equalization phase**: indicates the equalization phase when LTSSM is in the Recovery.Equalization state: • 00: Phase 0 • 01: Phase 1 • 10: Phase 2 • 11: Phase 3

Table 55: LTSSM States

Appendix A: Register Content of the Configuration Space

This appendix describes the layout of the Configuration Space and provides the mapping for each register in the Space. It also describes how each register is implemented in the XpressRICH3 Core, and highlights any differences between the description of the register in the relevant specification, and the implementation of that register in the Core.

A.1 PCI Configuration Space

The following table describes the layout of the PCI Configuration Space:

Offset	Physical Function(s)	Virtual Function(s)
000h ... 03Ch	Type0 (Endpoint) or Type1 (Root port/Bridge/Switch) Standard PCI configuration header	Type0 (Endpoint) or Type1 (Root port/Bridge/Switch) Standard PCI configuration header
040h ... 07Ch	Application Specific Area (optional)	Application Specific Area (optional)
080h ... 0B8h	PCI Express Capability	PCI Express Capability
0BCh ... 0CCh	reserved	reserved
0D0h ... 0D8h	MSI-X Capability (optional)	MSI-X Capability (optional)
0DCh	reserved	reserved
0E0h ... 0F4h	MSI Capability	MSI Capability
0F8h ... 0FCh	PCI Power Management Capability	n/a

Table 56: PCI Configuration Space Layout

The application-specific area of the PCI configuration space is a 64-byte range that can be used to implement application-specific registers and capabilities. In order to add application-specific capability registers, the application must implement registers in the application logic; setting `K_PCICONF[111:104]` to point to the first user capability, and connecting them to the Core using the Configuration Expansion interface.

A.1.1 PCI Express Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
Capabilities Register		Next Cap PTR	Capability ID	080h
Device Capabilities				084h
Device Status		Device Control		088h
Link Capabilities				08Ch
Link Status		Link Control		090h
Slot Capabilities				094h
Slot Status		Slot Control		098h
Root Capabilities		Root Control		09Ch
Root Status				0A0h
Device Capabilities 2				0A4h
Device Status 2		Device Control 2		0A8h
Link Capabilities 2				0ACh
Link Status 2		Link Control 2		0B0h
Slot Capabilities 2				0B4h
Slot Status 2		Slot Control 2		0B8h

Table 57: PCI Express Capability Structure

A.1.1.1 PCI Express Capability List Register (Offset 080h)

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
7:0	Capability ID	Always 10h.	RO
15:8	Next Capability Pointer	Depends on which capabilities are implemented.	RO

Table 58: PCI Express Capability List Register

A.1.1.2 PCI Express Capabilities Register (Offset 082h)

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

For more information about Core Variables (k_xxx signals), see Section 2.3.2.

Bits	Field	Additional Description	Attr.
3:0	Capability Version	Hardwired to 2h.	RO
7:4	Device/Port Type	Defined by k_gen.	RO
8	Slot Implemented	Defined by k_pexconf.	HwInit

Table 59: PCI Express Capabilities Register

Bits	Field	Additional Description	Attr.
13:9	Interrupt Message Number	Defined by k_pexconf.	RO
14	Reserved	Hardwired to 0	RO
15	Undefined	Hardwired to 0	-

Table 59: PCI Express Capabilities Register

A.1.1.3 Device Capabilities Register (Offset 084h)

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

For more information about Core Variables (k_xxx signals), see Section 2.3.2

Bits	Field	Additional Description	Attr.
2:0	Max_Payload_Size_Supported	Defined by k_pexconf.	RO
4:3	Phantom Functions Supported	Hardwired to 00.	RO
5	Extended Tag Field Supported	Hardwired to 1: 8-bit tag field.	RO
8:6	Endpoint L0s Acceptable Latency	Defined by k_pexconf.	RO
11:9	Endpoint L1 Acceptable Latency	Defined by k_pexconf.	RO
14:12	Undefined	Hardwired to 000.	RO
15	Role-Based Error Reporting	Hardwired to 1.	RO
17:16	Reserved	Hardwired to 00.	RO
25:18	Captured Slot Power Limit Value	–	RO
27:26	Captured Slot Power Limit Scale	–	RO
28	Function Level Reset Capability	Defined by k_pexconf.	RO

Table 60: Device Capabilities Register

A.1.1.4 Device Control Register (Offset 088h)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.5 Device Status Register (Offset 08Ah)

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
4	AUX Power Detected	Signaled using `TL_PM_AUXPWR`.	RO
5	Transactions Pending	Signaled using `TL_REPORT_STATUS0`.	RO

Table 61: Device Status Register

A.1.1.6 Link Capabilities Register (Offset 08Ch)

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

For more information about Core Parameters, see Section 2.3.

Bits	Field	Additional Description	Attr.
3:0	Max Link Speed	Defined by k_gen.	RO
9:4	Maximum Link Width	Defined by G_NUM_LANES.	RO
11:10	ASPM Support	Defined by k_pexconf.	RO
14:12	L0s Exit Latency		RO
17:15	L1 Exit Latency		RO
18	Clock Power Management		RO
19	Surprise Down Error Reporting Capable		RO
20	Data Link Layer Active Reporting Capable		RO
21	Link Bandwidth Notification Capability	• 0 for upstream ports. • 1 for downstream ports.	RO
22	ASPM Optionality Compliance	Set to 1b.	HwInit
23	reserved	–	–
31:24	Port Number	Defined by k_pexconf.	HwInit

Table 62: Link Capabilities Register

A.1.1.7 Link Control Register (Offset 090h)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.8 Link Status Register (Offset 092h)

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
12	Slot Clock Configuration	Defined by k_pexconf.	HwInit

Table 63: Link Status Register

A.1.1.9 Slot Capabilities Register (Offset 094h)

All the bits in this register are defined by the Core Variable k_pexconf (see Section 2.3.2).

For a full description of all the bits in this register, see the *PCI Express Specification.*

A.1.1.10 Slot Control Register (Offset 098h)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.11 Slot Status Register (Offset 09Ah)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.12 Root Control Register (Offset 09Ch)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.13 Root Capabilities Register (Offset 09Eh)

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For more information, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
0	CRS Software Visibility	Defined by k_pexconf. This field must be 0.	RO
15:1	reserved	Hardwired to 0s.	–

Table 64: Root Capabilities Register

A.1.1.14 Root Status Register (Offset 0A0h)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.15 Device Capabilities 2 Register (Offset 0A4h)

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For more information, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
3:0	Completion Timeout Ranges Supported	Defined by k_pexconf.	HwInit
4	Completion Timeout Disable Supported		RO
5	ARI Forwarding Supported		RO
6	AtomicOp Routing Supported		RO
7	32-Bit AtomicOp Completer Supported		RO
8	64-Bit AtomicOp Completer Supported		RO
9	128-Bit CAS Completer Supported		RO
10	No RO-enabled PR-PR Passing		RO
11	LTR Mechanism Supported	Defined by k_pexconf.	RO
13:12	TPH Completer Support		RO
16	10-bit tag completer supported		RO
17	10-bit tag requester supported		RO
19:18	OBFF Supported		RO
20	Extended FMT Field Supported	Hardwired to 1.	RO

Table 65: Device Capabilities Register

Bits	Field	Additional Description	Attr.
21	End_End TLP Prefix Supported	Defined by k_pexconf.	HwInit
23:22	Max End_End TLP Prefixes		HwInit

Table 65: Device Capabilities Register

A.1.1.16 Device Control 2 Register (Offset 0A8h)

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
5	ARI Forwarding Enable	This bit is RO when k_pexconf[37]=0, RW when k_pexconf[37]=1.	RO/RW
8	IDO Request Enable	–	RW
9	IDO Completion Enable		RW
10	LTR Mechanism Enable	This bit is RO when k_pexconf[43]=0, RW when k_pexconf[43]=1.	RO/RW
12	10-bit Tag Requester Enable	This bit is RO when k_pexconf[49]=0, RW when k_pexconf[49]=1 / RO/RW.	RO/RW
14:13	OBFF Enable	This bit is RO when k_pexconf[51:50]=00, otherwise it is RW.	RO/RW
15	End-End TLP Prefix Blocking	This bit is RO when k_pexconf[53]=0, otherwise it is as defined in the *PCIe Specifications*.	RW

Table 66: Device Control 2 Register

A.1.1.17 Device Status 2 Register (Offset 0AAh)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.18 Link Capabilities 2 Register (Offset 0ACh)

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For more information, see the *PCI Express Specification.*

For more information about Core Variables (k_xxx signals), see Section 2.3.2.

Bits	Field	Additional Description	Attr.
0	reserved	Hardwired to 0.	–
7:1	Supported Link Speeds Vector	Defined by k_gen.	RO
8	Crosslink Supported	Defined by k_pexconf.	RO
15:9	Lower SKP OS Generation Supported Speeds Vector		RO
22:16	Lower SKP OS Reception Supported Speeds Vector		RO
23	Retimer presence detection supported		RO
24	Two retimers presence detection supported		RO
31:245	reserved	Hardwired to 0.	–

Table 67: Link Capabilities 2 Register

A.1.1.19 Link Control 2 Register (Offset 0B0h)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.20 Link Status 2 Register (Offset 0B2h)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.21 Slot Capabilities 2 Register (Offset 0B4h)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.22 Slot Control 2 Register (Offset 0B6h)

This register is implemented as defined by the *PCI Express Specification.*

A.1.1.23 Slot Status 2 Register (Offset 0B8h)

This register is implemented as defined by the *PCI Express Specification.*

A.1.2 MSI-X Capability Structure

31:24	23:16	15:8	7:3	2:0	Byte Offset
Message Control		Next Pointer	Capability ID		0D0h
Table Offset				Table BIR	0D4h
PBA Offset				PBA BIR	0D8h

Table 68: MSI-X Capability Structure

A.1.2.1 Capability ID for MSI-X

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Local Bus Specification 3.0.*

Bits	Field	Additional Description	Attr.
7:0	Capability ID	Always 11h.	RO

Table 69: Capability ID for MSI-X

A.1.2.2 Next Pointer for MSI-X

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Local Bus Specification 3.0.*

Bits	Field	Additional Description	Attr.
7:0	Next Pointer	Depends on which capabilities are implemented.	RO

Table 70: Next Pointer for MSI-X

A.1.2.3 Message Control for MSI-X

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *PCI Local Bus Specification 3.0.*

For more information about Core Variables (k_xxx signals), see Section 2.3.2.

Bits	Field	Additional Description	Attr.
10:0	Table Size	Defined by k_pciconf.	RO

Table 71: Message Control for MSI-X

A.1.2.4 Table Offset/Table BIR for MSI-X

All bits in this register are defined by the Core Variable k_pciconf (see Section 2.3.2).

A.1.2.5 PBA Offset/PBA BIR for MSI-X

All bits in this register are defined by the Core Variable k_pciconf (see Section 2.3.2).

A.1.3 MSI Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
Message Control		Next Pointer	Cap ID	0E0h
Message Address				0E4h
Message Upper Address				0E8h
		Message Data		0ECh
Mask Bits				0F0h
Pending Bits				0F4h

Table 72: Message Signaled Interrupt (MSI) Capability Structure

A.1.3.1 Capability ID for MSI

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Local Bus Specification 3.0.*

Bits	Field	Additional Description	Attr.
7:0	Capability ID	Always 05h.	RO

Table 73: Capability ID for MSI

A.1.3.2 Next Pointer for MSI

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Local Bus Specification 3.0.*

Bits	Field	Additional Description	Attr.
7:0	Next Pointer	Depends on which capabilities are implemented.	RO

Table 74: Next Pointer for MSI

A.1.3.3 Message Control for MSI

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *PCI Local Bus Specification 3.0.*

For more information about Core Variables (k_xxx signals), see Section 2.3.2.

Bits	Field	Additional Description	Attr.
3:1	Multiple Message Capable	Defined by `K_PCICONF` for PFs; defined by `K_PEXCONF` for VFs.	RO
7	64-Bit Address Capable	Hardwired to 1.	RO

Table 75: Message Control for MSI

Bits	Field	Additional Description	Attr.
8	Per-Vector Masking Capable	Defined by K_PCICONF for PFs, hardwired to 1 for VFs.	
15:9	reserved	Hardwired to 0.	–

Table 75: Message Control for MSI

A.1.3.4 Message Address for MSI

This register is implemented as defined by the *PCI Local Bus Specification 3.0.*

A.1.3.5 Message Upper Address for MSI

This register is implemented as defined by the *PCI Local Bus Specification 3.0.*

A.1.3.6 Message Data for MSI

This register is implemented as defined by the *PCI Local Bus Specification 3.0.*

A.1.3.7 Mask Bits for MSI

This register is implemented as defined by the *PCI Local Bus Specification 3.0.*

A.1.3.8 Pending Bits for MSI

This register is implemented as defined by the *PCI Local Bus Specification 3.0.*

A.1.4 Power Management Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
Power Management Capabilities		Next Item PTR	Cap ID	0F8h
Data	PMCSR_BSE Bridge Support Extensions	Power Management Control & Status Register		0FCh

Table 76: Power Management Capability Structure

A.1.4.1 Capability ID

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Bus Power Management Interface Specification 1.2*.

Bits	Field	Additional Description	Attr.
7:0	Cap_ID	Always 01h.	RO

Table 77: Capability ID

A.1.4.2 Next Item Pointer

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Bus Power Management Interface Specification 1.2*.

Bits	Field	Additional Description	Attr.
7:0	Next Pointer	Depends on which capabilities are implemented.	RO

Table 78: Next Item Pointer

A.1.4.3 Power Management Capabilities

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Bus Power Management Interface Specification 1.2*.

Bits	Field	Additional Description	Attr.
15:11	PME_Support	Defined by k_pciconf.	RO
10	D2_Support		RO
9	D1_Support		RO
8:6	Aux_Current		RO
5	DSI		RO
4	reserved	Hardwired to 0.	RO
3	PME Clock		RO
2:0	Version	Set to 011b.	RO

Table 79: Power Management Capabilities

A.1.4.4 Power Management Control/Status (PMCSR)

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *PCI Bus Power Management Interface Specification 1.2*. For more information about Power Management, see Section 2.8.

Bits	Field	Additional Description	Attr.
14:13	Data_Scale	Defined by `TL_PM_DATA0[9:8]`.	RO
3	No_Soft_Reset	Hardwired to 1.	RO

Table 80: PMCSR

A.1.4.5 PMCSR_BSE

All bits of this register are hardwired to 0.

A.1.4.6 Data

The following table provides additional information about the implementation of this register in the XpressRICH3 Core.

For more information about Power Management, see Section 2.8.

Bits	Field	Additional Description	Attr.
7:0	Data_Register	Defined by `TL_PM_DATA0[7:0]`.	RO

Table 81: Data Register

A.2 PCI Express Extended Configuration Space

The following table describes the layout of the PCI Express Extended Configuration Space:

Offset	Physical Function(s)	Virtual Function(s)
100h ... 104h	Vendor-specific capability with VSECID=1556h; RevID=1h	Vendor-specific capability with VSECID=1556h; RevID=1h
108h ... 10Ch	Latency Tolerance Reporting capability (PF0 only) (optional)	n/a
110h ... 11Ch	L1 PM Substates capability (PF0 only) (optional)	n/a
120h ... 124h	Address Translation Service capability (optional)	not currently supported for VFs
128h ... 12Ch	Alternate Routing ID Interpretation capability (upstream only)	n/a
130h ... 13Ch	Page Request Interface capability (optional)	n/a
140h ... 17Ch	SR-IOV capability (optional)	n/a
180h ... 1ACh	Multicast capability (optional)	not currently supported for VFs
1B0h ... 1B8h	TPH Requester Capability (optional)	not currently supported for VFs
1C0h ... 1C8h	ACS Extended Capability (optional)	not currently supported for VFs
1D0h ... 1D8h	Precision Time Measurement Capability (PF0 only) (optional)	n/a
1E0h ... 1E8h	Data Link Feature Extended Capability (optional)	n/a
1F0h ... 1F4h	PASID (optional)	n/a

Table 82: PCI Express Extended Configuration Space Layout

Offset	Physical Function(s)	Virtual Function(s)
200h ... 244h	Advanced Error Reporting Capability (optional)	not currently supported for VFs
250h ... 258h	DPC Extended Capability (optional)	n/a
300h ... 328h	Secondary PCI Express extended capability (PF0only if the device supports speeds of 8 GT/s)	n/a
340h ... 36Ch	reserved	n/a
378h ... 3BCh	reserved	n/a
404h ... 434h	Resizable BAR extended capability (optional)	n/a
440h ... 47Ch	reserved	n/a
800h ... FFCh	Application-specific area (optional)	Application-specific area (optional)

Table 82: PCI Express Extended Configuration Space Layout

The application-specific area of the PCI Express extended configuration space is a 2K-byte range that can be used to implement application-specific registers and extended capabilities. In order to add application-specific extended capability registers, the application must set the `K_PCICONF[103]` bit, implement registers in then application logic, and connect them to the Core using the Configuration Expansion interface. First capability must always be implemented at address 800h.

A.2.1 Vendor Specific Extended Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
Vendor-Specific Extended Capability Header				100h
Vendor-Specific Header				104h

Table 83: Vendor-Specific Extended Capability Structure

A.2.1.1 Vendor-Specific Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 000Bh.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 84: Vendor-Specific Extended Capability Header

A.2.1.2 Vendor-Specific Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
15:0	VSEC ID	Always 1556h.	RO
19:16	VSEC Rev	Hardwired to 1h.	RO
31:20	VSEC Length	Hardwired to 008h.	RO

Table 85: Vendor-Specific Header

A.2.2 Latency Tolerance Reporting Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
PCI Express Extended Capability Header				108h
Max No-Snoop Latency Register		Max Snoop Latency Register		10Ch

Table 86: Latency Tolerance Reporting Capability Structure

A.2.2.1 LTR Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 0018h.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 87: LTR Extended Capability Header

A.2.2.2 Max No-Snoop/Snoop Latency Registers

These registers are implemented as defined by the *PCI Express Specification.*

A.2.3 L1 Substates Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
PCI Express Extended Capability Header				110h
L1 PM Substates Capability Register				114h
L1 PM Substates Control 1 Register				118h
L1 PM Substates Control 2 Register				11Ch

Table 88: L1 Substates Capability Structure

A.2.3.1 L1 PM Substates Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 001Eh.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 89: L1 PM Substates Extended Capability Header

A.2.3.2 L1 PM Capabilities Register

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For more information about Core Variables (k_xxx signals), see Section 2.3.2.

Bits	Field	Additional Description	Attr.
0	PCI-PM L1.2 Supported	Defined by k_pexconf[287:256].	HwInit
1	PCI-PM L1.1 Supported		HwInit
2	ASPM L1.2 Supported		HwInit
3	ASPM L1.1 Supported		HwInit
4	L1 PM Substates Supported		HwInit
7:5	reserved		RsvdP
15:8	Port Common Mode Roster Time		HwInit/RsvdP
17:16	Port T POWER ON Scale		HwInit/RsvdP
18	reserved		RsvdP
23:19	Port T POWER ON Value		HwInit/RsvdP
31:24	reserved		RsvdP

Table 90: L1 PM Substates Capability Register

A.2.3.3 L1 PM Control 1/2 Registers

These registers are implemented as defined by the *L1 PM Substates with CLKREQ ECN*.

A.2.4 Address Translation Service Extended Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
ATS Extended Capability Header				120h
ATS Control Register		ATS Capability Register		124h

Table 91: PCI Express ATS Extended Capability Structure

A.2.4.1 ATS Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *Address Translation Services Specification 1.1.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 000Fh.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 92: ATS Extended Capability Header

A.2.4.2 ATS Capability Register

The following table provides additional information about the implementation of this register in the XpressRICH3 Core.

For more information about Core Variables (k_xxx signals), see Section 2.3.2.

Bits	Field	Additional Description	Attr.
4:0	Invalidate Queue Depth	Defined by k_pciconf.	RO
5	Page aligned Request		RO
6	Global Invalidate Supported		RO
15:7	reserved	–	–

Table 93: ATS Capability Register

A.2.4.3 ATS Control Register

This register is implemented as defined by the *Address Translation Services Specification 1.1.*

A.2.5 Alternate Routing ID Interpretation Capability Structure

ARI capability is implemented in all upstream port functions; its fields are set to fixed values according to the Core configuration and can not be configured by users.

31:24	23:16	15:8	7:0	Byte Offset
ARI Extended Capability Header				128h
ARI Control Register		ARI Capability Register		12Ch

Table 94: ARI Extended Capability Structure

A.2.5.1 ARI Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *Alternative Routing-ID Interpretation (ARI) Engineering Change Notice.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 000Eh.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 95: ARI Extended Capability Header

A.2.5.2 ARI Capability/Control Register

The following table provides additional information about the implementation of this register in the XpressRICH3 Core.

Bits	Field	Additional Description	Attr.
0	MFVC Function Groups Capability (M)	Hardwired to 0.	RO
1	ACS Function Groups Capability (A)	Hardwired to 0.	RO
15:8	Next Function Number	Indicates ID of next physical function; otherwise 0.	RO

Table 96: ARI Capability/Control Register

A.2.6 Page Request Extended Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
Page Request Extended Capability Header				130h
Page Request Status Register		Page Request Control Register		134h
Outstanding Page Request Capacity				138h
Outstanding Page Request Allocation				13Ch

Table 97: Page Request Extended Capability Structure

A.2.6.1 Page Request Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *Address Translation Services Specification 1.1.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 0013h.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 98: Page Request Extended Capability Header

A.2.6.2 Page Request Control Register

This register is implemented as defined by the *Address Translation Services Specification 1.1.*

A.2.6.3 Page Request Status Register

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *Address Translation Services Specification 1.1.*

For more information about the Reporting Interface, see Section 2.11.

Bits	Field	Additional Description	Attr.
8	Stopped (S)	Defined by `TL_REPORT_STATUS0[15:0]`.	RO
15	PRG Response PASID Required	Defined by k_pciconf.	RO

Table 99: Page Request Status Register

A.2.6.4 Outstanding Page Request Capacity

All bits in this register are defined by the Core Variable k_pciconf (see Section 2.3.2).

A.2.6.5 Outstanding Page Request Allocation

This register is implemented as defined by the *Address Translation Services Specification 1.1.*

A.2.7 Single Root I/O Virtualization Extended Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
SR-IOV Extended Capability Header				140h
SR-IOV Capabilities				144h
SR-IOV Status		SR-IOV Control		148h
Total VFs		Initial VFs		14Ch
RsvdP	Function Dependency Link	Number VFs		150h
VF Stride		First VF Offset		154h
VF Device ID		RsvdP		158h
Supported Page Sizes				15Ch
Return Page Size				160h
VF BAR0				164h
VF BAR1				168h
VF BAR2				16Ch
VF BAR3				170h
VF BAR4				174h
VF BAR5				178h
VF Migration State Array Offset				17Ch

Table 100: SR-IOV Extended Capability Structure

A.2.7.1 SR-IOV Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *Single Root I/O Virtualization and Sharing Specification 1.1.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 0010h.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 101: SR-IOV Extended Capability Header

A.2.7.2 SR-IOV Capabilities

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *Single Root I/O Virtualization and Sharing Specification 1.1.*

Bits	Field	Additional Description	Attr.
0	VF Migration Capable	Hardwired to 0; VF migration is not supported.	RO
1	ARI-Capable Hierarchy Preserved	Tied to 1 in lowest numbered PF that has VFs; tied to 0 in other PFs.	RO
2	VF 10-bit Tag Requester Supported	Set by `K_SRIOV[46]`.	RO
20:3	reserved	Hardwired to 0.	RO
31:21	VF Migration Interrupt Message Number	Hardwired to 0; VF migration is not supported.	RO

Table 102: SR-IOV Capabilities

A.2.7.3 SR-IOV Control

All bits in this register are hardwired to 0s, except Bits 0, 3 and 4, which are implemented as described by the *Single Root I/O Virtualization and Sharing Specification 1.1.*

A.2.7.4 SR-IOV Status

All bits in this register are hardwired to 0s.

A.2.7.5 Initial VFs

The value of this register is set to reflect the number of VFs assigned to this PF, according to the VF top and VF base fields of `K_SRIOV` (see Section 4.2).

A.2.7.6 Total VFs

The value of this register is set to reflect the number of VFs assigned to this PF, according to the VF top and VF base fields of `K_SRIOV` (see Section 4.2).

A.2.7.7 NumVFs

This register is implemented as defined by the *Single Root I/O Virtualization and Sharing Specification 1.1.*

A.2.7.8 Function Dependency Link

This register is defined by the Core Variable k_sriov (see Section 2.3.2).

A.2.7.9 First VF Offset

This register is set according to the VF base field of `K_SRIOV` and the value of the ARI_capable_hierarchy bit of SRIOV capability.

A.2.7.10 VF Stride

This register is hardwired to 0001h.

A.2.7.11 VF Device ID

This register is defined by the Core Variable k_sriov (see Section 2.3.2).

A.2.7.12 Supported Page Sizes

This register is defined by the Core Variable k_sriov (see Section 2.3.2). It is set to either logical or 0553h.

A.2.7.13 System Page Size

This register is implemented as defined by the *Single Root I/O Virtualization and Sharing Specification 1.1.*

A.2.7.14 VF Bar0, VF Bar1, ... VF Bar5

This register is implemented as defined by the *Single Root I/O Virtualization and Sharing Specification 1.1*, except where a BAR mask has been defined by the Core Variable k_sriov (see Section 2.3.2).

A.2.7.15 VF Migration State Array Offset

All bits in this register are hardwired to 0s. VF migration is not supported.

A.2.8 Multicast Capability

31:24	23:16	15:8	7:0	Byte Offset
PCI Express Extended Capability Header				180h
Multicast Control Register		Multicast Capability Register		184h
MC Base Address Register				188h
MC Receive Register				190h
MC Block All Register				198h
MC Block Untranslated Register (reserved if ATS is not implemented)				1A0h
MC Overlay BAR (Switch/Rootport only)				1A8h

Table 103: Multicast Capability Structure

A.2.8.1 Multicast Extended Capabilities Register (Offset 180h)

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 0012h.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 104: Multicast Extended Capability Header

A.2.8.2 Multicast Control/Capability Registers (Offset 184h)

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *PCI Express Specification.*

For more information about Core Variables, see Section 2.3.2.

Bits	Field	Additional Description	Attr.
5:0	MC_Max_Group	Defined by `K_PCICONF[245:240]`.	RO
13:8	MC_Window_Size_Requested	Defined by `K_PCICONF[253:248]`.	RO
15	MC_ECRC_Regeneration_Supported	Defined by `K_PCICONF[255]`.	RO

Table 105: Multicast Control/Capability Register

A.2.8.3 MC Base Address Register (Offset 188h)

This register is implemented as defined by the *PCI Express Specification.*

A.2.8.4 MC Receive Register (Offset 190h)

This register is implemented as defined by the *PCI Express Specification.*

A.2.8.5 MC Block All Register (Offset 198h)

This register is implemented as defined by the *PCI Express Specification.*

A.2.8.6 MC Block Untranslated Register (Offset 1A0h)

This register is implemented as defined by the *PCI Express Specification.* It is reserved if Address Translation Service support is not implemented (set by k_pciconf[112], see Section 2.3.2).

A.2.8.7 MC Overlay BAR (Offset 1A8h)

This register is implemented as defined by the PCI Express Specification if core is Rootport or Switch, it is reserved otherwise.

A.2.9 TPH Requester Capability

The TPH Requester Capability structure is required for all functions that are capable of generating Request TLPs with TPH. For a multi-function device, this capability must be present in each function that is capable of generating Requests with TPH.

31:24	23:16	15:8	7:0	Byte Offset
PCI Express Extended Capability Header				1B0h
TPH Requester Capability Register				1B4h
TPH Requester Control Register				1B8h

Table 106: TPH Requester Capability Structure

A.2.9.1 TPH Requester Extended Capability Header

The Next Capability Offset field depends on which capabilities are implemented.

A.2.9.2 TPH Requester Capability Register

All bits in this register are defined by k_pexconf.

A.2.9.3 TPH Requester Control Register

This register is implemented as defined by the *PCI Express Specification.*

A.2.10 ACS Extended Capability

The ACS Extended Capability structure is required for all functions that implement Access Control Services.

31:24	23:16	15:8	7:0	Byte Offset
ACS Extended Capability Header				1C0h
ACS Control Register		ACS Capability Register		1C4h
Egress Control Vector				1C8h

Table 107: ACS Capability Structure

A.2.10.1 ACS Extended Capability Header

The Next Capability Offset field depends on which capabilities are implemented.

A.2.10.2 ACS Capability and Control Register

Capability bits in this register are defined by k_pexconf.

Control bits are implemented when the corresponding capability bit is set.

A.2.10.3 Egress Control Vector

This register is present when ACS P2P Egress Control is supported. It is limited to 32 bits.

A.2.11 Precision Time Measurement Capability

Precision Time Measurement Capability is implemented when either the PTM Requester or Responder capable bit is set in k_pexconf (see Section 4.2).

31:24	23:16	15:8	7:0	Byte Offset
PTM Extended Capability Header				1D0h
PTM Capability Register				1D4h
PTM Control Register				1D8h

Table 108: PTM Capability Structure

A.2.11.1 PTM Extended Capability Header

The Next Capability Offset field depends on which capabilities are implemented.

A.2.11.2 PTM Capability Register

Capability bits in this register are defined by k_pexconf (see Section 4.2)

A.2.11.3 PTM Control Register

This register is implemented as defined by the *Precision Time Measurement (PTM), Revision 1.0a* ECN.

A.2.12 Data Link Feature Extended Capability

Data Link Feature Extended Capability is implemented using K_PEXCONF[344] (see Section 4.2).

31:24	23:16	15:8	7:0	Byte Offset
Data Link Feature Extended Capability Header				1E0h
Data Link Feature Capability Register				1E4h
Data Link Feature Status Register				1E8h

Table 109: Data Link Feature Capability Structure

A.2.12.1 Data Link Feature Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification*.

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 0025h	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 110: Data Link Feature Extended Capability Header Register

A.2.12.2 Data Link Feature Capability Register

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification*.

Bits	Field	Additional Description	Attr.
22:0	Local Data Link Feature Supported	Defined by K_PEXCONF[342:320].	HwInit
30:23	reserved	–	RsvdP
31	Data Link Feature Exchange Enable	Defined by K_PEXCONF[343].	HwInit

Table 111: Data Link Feature Capability Register

A.2.12.3 Data Link Feature Status Register

This register is implemented as defined by the *PCI Express Specification*.

A.2.13 PASID Extended Capability

PASID Extended Capability is implemented using K_PCICONF[136] (see Section 4.2).

31:24	23:16	15:8	7:0	Byte Offset
PASID Extended Capability Header				1F0h
PASID Control Register		PASID Capability Register		1F4h

Table 112: PASID Capability Structure

A.2.13.1 PASID Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification*.

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 001Bh	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 113: PASID Extended Capability Header Register

A.2.13.2 PASID Capability Register

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
0	reserved	–	RsvdP
1	Execute Permission Supported	Defined by K_PCICONF.	RO
2	Privileged Mode Supported		RO
7:3	reserved	–	RsvdP
12:8	Max PASID Width	Defined by K_PCICONF.	RO
12:8	reserved	–	RsvdP

Table 114: PASID Capability Register

A.2.13.3 PASID Control Register

This register is implemented as defined by the *PCI Express Specification*.

A.2.14 Advanced Error Reporting Capability Structure

The following table shows the Advanced Error Reporting Extended Capability Structure.

31:24	23:16	15:8	7:0	Byte Offset
PCI Express Enhanced Capability Header				200h
Uncorrectable Error Status Register				204h
Uncorrectable Error Mask Register				208h
Uncorrectable Error Severity Register				20Ch
Correctable Error Status Register				210h
Correctable Error Mask Register				214h
Advanced Error Capabilities and Control Register				218h
Header Log Register				21Ch
Root Error Command				22Ch
Root Error Status				230h
Error Source Identification Register		Correctable Error Source Identification Register		234h
TLP Prefix Log Register				238h ... 244h

Table 115: Advanced Error Reporting Extended Capability Structure

A.2.14.1 Advanced Error Reporting Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 0001h.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 116: AER Extended Capability Header

A.2.14.2 Uncorrectable Error Status Register

This register is implemented as defined by the *PCI Express Specification.*

Bit 17 (Receiver overflow) is not implemented and is hardwired to 0.

A.2.14.3 Uncorrectable Error Mask Register

This register is implemented as defined by the *PCI Express Specification.*

Bit 17 (Receiver overflow) is not implemented and is hardwired to 0.

A.2.14.4 Uncorrectable Error Severity Register

This register is implemented as defined by the *PCI Express Specification.*
Bit 17 (Receiver overflow) is not implemented and is hardwired to 0.

A.2.14.5 Correctable Error Status Register

This register is implemented as defined by the *PCI Express Specification.*

Bit 15 (Header log overflow) is not implemented and is hardwired to 0.

A.2.14.6 Correctable Error Mask Register

This register is implemented as defined by the *PCI Express Specification.*

Bit 15 (Header log overflow) is not implemented and is hardwired to 0.

A.2.14.7 Advanced Error Capabilities and Control Register

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *PCI Express Specification.*

Bits 9 - 11 are hardwired to 0.

For more information about Core Variables (k_xxx signals), see Section 2.3.2.

Bits	Field	Additional Description	Attr.
5	ECRC Generation capable	Defined by k_pexconf.	RWS
7	ECRC Check Capable		RO

Table 117: Advanced Error Capabilities and Control Register

A.2.14.8 Header Log Register

This register is implemented as defined by the *PCI Express Specification.*

A.2.14.9 Root Error Command Register

This register is implemented as defined by the *PCI Express Specification.*

A.2.14.10 Root Error Status Register

The following table provides additional information about the implementation of certain bits of this register in the XpressRICH3 Core.

All other bits of this register are implemented as defined by the *PCI Express Specification.*

For more information about Core Variables (k_xxx signals), see Section 2.3.2.

Bits	Field	Additional Description	Attr.
31:27	Advanced Error Interrupt Message Number	Defined by k_pexconf.	RO

Table 118: Root Error Status Register

A.2.14.11 Error Source and Correctable Error Source Identification Register

This register is implemented as defined by the *PCI Express Specification.*

A.2.14.12 TLP Prefix Log Register

This register is implemented as defined by the *PCI Express Specification.*

A.2.15 DPC Extended Capability Structure

DPC extended capability is implemented for switch downstream or rootport devices that support DPC, as indicated by K_PEXCONF[303].

31:24	23:16	15:8	7:0	Byte Offset
DPC Extended Capability Header				2~~8~~50h
DPC Control Register		DPC Capability Register		2~~8~~54h
DPC Error Source ID Register		DPC Status Register		2~~8~~58h

Table 119: DPC Extended Capability Structure

A.2.15.1 DPC Extended Capability Header

The Next Capability Offset field depends on which capabilities are implemented.

A.2.15.2 DPC Capability Registers

The Capability register bits are defined by K_PEXCONF (see Section 4.2).

Note: RP Extensions for DPC, which include RP PIO features, are not supported.

A.2.15.3 DPC Control/Status/Error Source ID Register

These registers are implemented by the PCI Express specification.

Note: RP Extensions for DPC, which include RP PIO features, are not supported.

A.2.16 Secondary PCI Express Extended Capability Structure

The following table shows Secondary PCI Express Extended Capability structure, which is implemented in function #0 of devices that support speeds of 8GT/s or higher.

31:24	23:16	15:8	7:0	Byte Offset
Secondary PCI Express Extended Capability Header				300h
Link Control 3 Register				304h
Lane Error Status Register				308h
Link Equalization Control Register				30Ch ... 328h

Table 120: Secondary PCI Express Extended Capability Structure

A.2.16.1 Secondary PCI Express Extended Capability Header

The following table provides additional information about the implementation of this register in the XpressRICH3 Core. For a full description, see the *PCI Express Specification.*

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 0019h.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

Table 121: Secondary PCI Express Extended Capability Header

A.2.16.2 Link Control 3 Register

This register is implemented as defined by the *PCI Express Specification.*

A.2.16.3 Lane Error Status Register

Bits corresponding to implemented lanes (as indicated by G_NUM_LANES) are implemented as defined by the *PCI Express Specification.*

All other bits are hardwired to 0s.

A.2.16.4 Link Equalization Control Register

- For downstream ports, this register is defined by k_equpreset[255:0] (see Section 2.3.2).
- For upstream ports, this register is implemented as defined by the *PCI Express Specification.*

A.2.17 Resizable BAR Extended Capability Structure

The Resizable BAR capability is implemented in each function in which k_bar indicates that a resizable BAR is present.

31:24	23:16	15:8	7:0	Byte Offset
Resizable BAR Extended Capability Header				404h
Resizable BAR Capability Register (0)				408h
Reserved		Resizable BAR Control Register (0)		40Ch
...				
Resizable BAR Capability Register (n)				(n*8+4)
Reserved		Resizable BAR Control Register (n)		(n*8+8)

Table 122: Resizable BAR Extended Capability Structure

A.2.17.1 Resizable BAR Extended Capability Header

The Next Capability Offset field depends on which capabilities are implemented.

A.2.17.2 Resizable BAR Capability and Control Registers

The Capability and Control register bits are defined by k_bar (see Section 4.2).

Appendix B: Message, Error, and Event Handling

B.1 Received Message Processing

The following table describes how PCI Express messages are handled by the XpressRICH3Core for both Endpoints and Rootports.

For more information about each of these message types, see the *PCI Express™ Base Specification.*

Received Message	Endpoint	Switch Upstream/ Bridge Forward	Rootport	Switch Downstream
Unlock	Passed to Rx interface		Unsupported; discarded	
Invalidate Request (ATS)	Passed to Rx interface		Unsupported; discarded	
Invalidate Completion (ATS)	Unsupported; discarded		Passed to Rx interface	
Page Request (ATS)	Passed to Rx interface		Unsupported; discarded	
PRG Response (ATS)	Passed to Rx interface		Unsupported; discarded	
Latency Tolerance Reporting	Unsupported; discarded		Unsupported and discarded if LTR is not supported and enabled; passed to Rx interface otherwise.	
Optimized Buffer Flush/Fill	Processed internally. Any change is reported on `TL_PM_OBFF_STATUS`.		Unsupported; discarded	
PM_Active_State_Nak	If the Core receives this message while it is preparing for transition to ASPM L1, it stops the transition.		Unsupported; discarded	
PM_PME	Unsupported; discarded		Processed internally. Captured in the PCIe Capability Root Status Register. The PME# interrupt is reported by the assertion of tl_report_event[0], if enabled. See the *PCIe Spec. Chapter 6.1.6* for more information.	Passed to Rx interface.
PME_Turn_Off	Processed internally. Causes assertion of `TL_PM_L2_STATUS`.		Unsupported; discarded	
PME_To_Ack	Unsupported; discarded	Treated as malformed; discarded	Processed internally. Causes assertion of `TL_PM_L2_STATUS`. The Core enters L2 if `TL_PM_L2_CONTROL` is asserted.	
Assert_INTA/B/C/D	Treated as malformed; discarded		Processed internally. Causes assertion of relevant `TL_INT_PINSTATE` bit.	
Deassert_INTA/B/C/D				

Table 123: Received Message Processing

Received Message	Endpoint	Switch Upstream/ Bridge Forward	Rootport	Switch Downstream
ERR_COR	Unsupported; discarded	Treated as malformed; discarded	Processed internally. Logged in the PCIe Config Space. Can trigger AER/ System error interrupt, which is reported by asserting TL_REPORT_EVENT[2:1]. See the *PCIe Spec., Chapter 6.2.6* for more information.	Passed to Rx interface.
ERR_NONFATAL				
ERR_FATAL				
Ignored messages	Silently discarded			
Set_Slot_Power_Limit	Processed internally. Values are captured in the configuration space.		Unsupported; discarded	
Vendor Defined Type 0	Passed to Rx interface			
Vendor Defined Type 1				
PTM Request/PTM Response (D)	Unsupported & discarded if PTM support is not implemented; Silently discarded if PTM enable bit is clear; otherwise passed to application.			

Table 123: Received Message Processing

B.2 TLP Event Processing

The following table describes how PCI Express errors are processed in the XpressRICH3 Core, depending on whether they are detected by the Core itself or by the application.

Note: Error logging and reporting is subject to error processing and precedence rules. See the *PCIe Specification, Chapter 6.2* for more information.

Error	Mode	If detected by the XpressRICH3...	If detected by the Application...
Received Malformed TLP	Normal Mode	The error is logged and reported. The TLP is either: • passed to the Rx interface and reported by asserting TL_RX_ERR[1] if TLP payload does not match length field or ECRC is not present as expected. • discarded.	
	Rx Stream Mode	The error is logged and reported. The TLP is either: • passed to the Rx interface and reported by asserting TL_RX_ERR[1] if TLP payload does not match length field or ECRC is not present as expected. • discarded.	

Table 124: TLP Event Processing

Error	Mode	If detected by the XpressRICH3...	If detected by the Application...
Received TLP ECRC Check failed	Normal Mode	The error is logged and reported: • If the TLP destination is the PLDA IP Core, the TLP is discarded, and a CPL with CA status is returned when applicable. • Otherwise, the TLP is passed to the Rx interface and reported by asserting TL_RX_ERR[0].	
	Rx Stream Mode	The error is logged and reported: • • If the TLP destination is the PLDA IP core, • the TLP is passed to the RX interface and reported by asserting TL_RX_ERR[0] and TL_RX_ERR[3]. A CPL with CA status is returned when applicable. • Otherwise, the TLP is passed to the Rx interface and reported by asserting TL_RX_ERR[0].	
Received Nullified TLP	Normal Mode	• If cut-through is disabled, the TLP is silently discarded. • If cut-through is enabled, the TLP is passed to the Rx interface and reported by asserting TL_RX_ERR[1].	
	Rx Stream Mode	The TLP is passed to the Rx interface and reported by asserting TL_RX_ERR[1].	
RX buffer data corruption detected in TLP	Normal Mode	The TLP is passed to the RX interface and reported by the assertion of TL_RX_ERR[2].	
	Rx Stream Mode		
Received Unsupported Request	Normal Mode	The error is logged and reported. The TLP is discarded, and a CPL with UR status is returned if applicable.	
	Rx Stream Mode	The error is logged and reported. The TLP is discarded (P/CPL) or the TLP (NP) is sent to the application using TL_RX_ERR[4], which must either send it to the configuration space (TL_RX_ERR[3]); or discard it and send CPL UR.	
Received Unexpected Completion (BUS# DEV# do not match, or function doesn't exist)	Normal Mode	The error is logged and reported. The TLP is discarded.	
	Rx Stream Mode		

Table 124: TLP Event Processing

Error	Mode	If detected by the XpressRICH3...	If detected by the Application...
Received Poisoned TLP	Normal Mode	The error is logged and reported. • If the TLP destination is the PLDA IP Core, the TLP is discarded, and a CPL with UR status is returned if applicable. • Otherwise, the TLP is passed to the Rx interface.	
	Rx Stream Mode	The error is logged and reported. • If the TLP destination is the PLDA IP core, the TLP is passed to the RX interface and reported by asserting TL_RX_ERR[3]. The Core will detect the EP bit in the header and send a CPL with UR status when applicable • Otherwise, the TLP is passed to the Rx interface. The application needs to check if the EP bit is set	
Completion timeout	Normal Mode		The application asserts TL_REPORT_ERROR[2:1].The error is logged and reported by the Core.
	Rx Stream Mode		
Received config read/write TLP but Config Space not ready	Normal Mode	Occurs when the application asserts TL_REPORT_STATE[1:0]. A CPL with CRS status is returned.	
	Rx Stream Mode	The TLP is sent to application, which must send it to configuration space (TL_RX_ERR[3]).	
Received non-posted request abort	Normal Mode	The error is logged and reported. The TLP is processed normally.	• The application send CPL with CA status and asserts TL_REPORT_ERROR[5]. • The error is logged and reported by the Core.
	Rx Stream Mode		
Received completion TLP with CA status	Normal Mode	The error is logged. The TLP is processed normally.	
	Rx Stream Mode		
Received completion TLP with UR or undefined status	Normal Mode	The error is logged. The TLP is processed normally.	
	Rx Stream Mode		
Received Malformed Completion	Normal Mode		• The application discards the TPL and asserts TL_REPORT_ERROR[4]. • The error is logged and reported by the Core.
	Rx Stream Mode		

Table 124: TLP Event Processing

Error	Mode	If detected by the XpressRICH3...	If detected by the Application...
Atomic Egress Blocked	Normal Mode		• The application asserts TL_REPORT_ERROR[12]. • The error is logged and reported by the Core.
	Rx Stream Mode		
Page Request Interface Error	Normal Mode		• The application discards the TPL and asserts TL_REPORT_ERROR[11:9]. • The error is logged and reported by the Core.
	Rx Stream Mode		
System Error (Rootport only)	Normal Mode	The Core logs the error in the PCIe Config Space Root Status Register and asserts TL_REPORT_EVENT[2] if System Error Interrupt is enabled.	
	Rx Stream Mode		
AER Error (Rootport only)	Normal Mode	The Core asserts TL_REPORT_EVENT[1]if AER Error Interrupt is enabled.	
	Rx Stream Mode		
TLP Prefix Blocked	Normal Mode		• The application asserts TL_REPORT_ERROR[15]. • The error is logged and reported by the Core.
	Rx Stream Mode		
ACS Violation	Normal Mode	The error is logged and reported. The TLP is discarded and a completion with CA status is returned when applicable.	• The application asserts TL_REPORT_ERROR[16]. • The error is logged and reported by the Core. See Section 13.3 for information about which ACS violations are checked by the Core.
	Rx Stream Mode	The error is logged and reported. The TLP is sent to the application, which must either send it to the configuration space (TL_RX_ERR[3]) or discard it and send CPL CA (TL_RX_ERR[5]).	

Table 124: TLP Event Processing

B.3 Event Processing

The following table describes how PCI Express events are processed in the XpressRICH3 Core, and the actions taken by both XpressRICH3 and by the application.

Event	XpressRICH3 Action	Application Action
Power Management Event received (Rootport only)	Captured in PCIe Capability Root Status Register. The PME# interrupt is reported by asserting tl_report_event[0], if enabled.	Processing is application-specific.
Legacy interrupt received (Rootport only)	For Rootports/Downstream Switches only: The Core asserts/deasserts the relevant TL_INT_PINSTATE bits to reflect interrupt status.	For Upstream Switches only: The application asserts/deasserts the relevant TL_INT_PINCONTROL bits to reflect interrupt status.
MSI received (Rootport only)		MSI are standard MemWr TLPs and are not detected or captured by the Core. Detection and processing is application-specific.
AER error (Rootport only)	The Core asserts TL_REPORT_EVENT[1] if AER Error Interrupt is enabled.	Processing is application-specific.
System Error (Rootport only)	The Core asserts TL_REPORT_EVENT[2] if System Error Interrupt is enabled.	Processing is application-specific.
Hotplug event (Rootport and Downstream Switch)	The Core asserts TL_REPORT_EVENT[3] if Hotplug Interrupt is enabled.	Processing is application-specific.
Equalization Request (Rootport and Downstream Switch)	The Core asserts TL_REPORT_EVENT[4] if Equalization Interrupt is enabled.	Processing is application-specific.
Link partner requests L1 entry (can only be initiated by Upstream Ports)	For Downstream Ports only: The Core enters L1 as soon as it's ready.	
Link partner requests L2 entry (can only be initiated by Downstream Ports)	For Upstream Ports only: The Core asserts TL_PM_L2_STATUS to inform the application.	The application must assert TL_PM_L2_CONTROL when it is ready to enter L2.
Link partner enters ASPM L0s	No action required. The application is not informed	
Link partner requests ASPM L1 entry (can only be initiated by Upstream Ports)	If ASPM L1 is not enabled, then the Core denies ASPM L1 entry. Otherwise it waits until the transmit buffer is empty and enters ASPM L1. The application is not informed.	
Legacy Power Management state change	In D1/D2/D3 hot/cold, the Core is not allowed to transmit interrupts and TLPs from the application.	The application must check configuration register address 0FCh bits 1:0 to detect state changes. Action is application specific.
Link exits L2 state	The Core deasserts pl_exit[0] in order to reset all Core logic, except sticky registers.	An application logic reset is optional.
PCIe commanded Hot Reset	The Core deasserts pl_exit[1] in order to reset all Core logic, except sticky registers.	An application logic reset is optional.

Table 125: Event Processing

Event	XpressRICH3 Action	Application Action
Data link goes down	The Core deasserts pl_exit[2] in order to reset all Core logic, except sticky registers and configuration registers (if the Core is a Rootport).	An application logic reset is optional.

Table 125: Event Processing

Appendix C: Electrical Idle Usage

The PIPE signal RxElecIdle received by the Core is not a reliable method for checking electrical idle state, entry, or exit at all speeds.

The optional setting k_gen[28] enables the Core to treat a falling edge of RxElecIdle at 2.5/5.0 GT/s as a received EIOS in some states; this can help troubleshoot some problems that may occur when RxElecIdle is de-asserted too early.

The table below describes where and how electrical idle is checked by the Core:

State	Event when k_gen[28] = 0 or at 8.0 GT/s or higher	Event when k_gen[28] = 1 (2.5/5.0 GT/s only)
Rec.Rcfg -> Rec.speed transition	EIOS received or EI inferred (see *Section 4.2.4.3* of the *PCIe Specification*)	–
Rec.Speed (speed change)	EIOS received or EI inferred (see *Section 4.2.4.3* of the *PCIe Specification*)	–
Lpbk.active -> Lpbk.exit transition	EIOS received or EI inferred (see *Section 4.2.4.3* of the *PCIe Specification*)	–
Disable -> Det.quiet transition	EIOS received, then RxElecIdle falling edge	EIOS received or RxElecIdle rising edge, then RxElecIdle falling edge
L1.idle -> Rec.Rcvlock transition	RxElecIdle falling edge*	–
L2.idle -> Det.quiet transition	RxElecIdle = 0	–
L2.twake -> Det.quiet transition	RxElecIdle falling edge*	–
L0/L0s -> L1.idle transition	EIOS received	EIOS received or RxElecIdle rising edge
L0/L0s -> L2.idle transition	EIOS received	EIOS received or RxElecIdle rising edge
RxL0s.inact -> RxL0s.idle transition	EIOS received	EIOS received or RxElecIdle rising edge
RxL0s.idle -> RxL0s.fts transition	EIEOS or FTS received	EIEOS or FTS received or RxElecIdle falling edge
Pol.comp -> Pol.active transition #1	EIOS received	EIOS received or RxElecIdle rising edge
Pol.comp -> Pol.active transition #2	EIEOS received or RxElecIdle falling edge	–
Pol.comp -> Det.quiet transition	EIOS received	EIOS received or RxElecIdle rising edge
Pol.active -> Pol.Comp transition	RxElecidle falling edge	–
Det.quiet -> Det.active transition	RxElecidle = 0*	–
L2.idle (downstream)	RxElecidle = 0 (beacon on lane 0)	–
L2.idle (upstream)	RxElecidle falling edge	

Table 126: Electrical Idle checking

State	Event when k_gen[28] = 0 or at 8.0 GT/s or higher	Event when k_gen[28] = 1 (2.5/5.0 GT/s only)
L0 -> Rec.Rcvlock transition	EI inferred (see *Section 4.2.4.3* of the *PCIe Specification*) without receiving EIOS	EI inferred (see *Section 4.2.4.3* of the *PCIe Specification*) without receiving: EIOS or RxElecIdle rising edge
RxStandby assertion/deassertion	RxElecIdle at 2.5/5.0 GT/s EIOS received and RxElecidle at 8GT/s	–

Table 126: Electrical Idle checking

*PHY is in P1 or P2 power mode in this state and thus an ordered set can not be received to detect electrical idle or exit from electrical idle.

Appendix D: CDC Implementation Considerations

The Clock Domain Crossing (CDC) is described in Section 2.3: Clock Domain Crossing Layer; this appendix contains implementation-specific information that you should take into consideration when implementing the CDC in your design flow.

D.1 Synchronization Modules

The CDC module instantiates various sub-modules for each synchronized data set depending on its type. Two of these sub-modules are of particular interest:

- The “pcie3_synchronizer” is a basic synchronizer that contains just two consecutive flip-flops. There is purposefully no reset; this is not necessary as long as the reset sequence follows the rules outlined in Chapter 3: Clocks and Resets. This module can be replaced by a technology-specific synchronizer cell; the designer may apply special constraints on this module when using a CDC checking tool.
- The “pcie3_syncignore” module does not perform synchronization; it is implemented on data paths that do not need to be synchronized because there is a mechanism that guarantees that data is only sampled in the destination clock domain when is safe to do so. The purpose of this module is to allow the designer to apply special constraints when using a CDC checking tool.

D.2 Unsynchronized Paths

All the signals used in the Core are synchronized using appropriate mechanisms, except for the paths detailed in the table below:

Destination	Source	Description
pl_wake_oen		This output is used for L2 exit (when CDC is implemented) and OBFF. It is controlled by registers located in both the TL_CLK and PL_PCLK clock domains. As this pin is asynchronous to the PIPE clock, any path going through it should be ignored for timing analysis.
pl_clkreq_oen		This output is used for Clock Power Management and L1 PM substates. It is controlled by registers located in both the TL_CLK and PL_PCLK clock domains. As this pin is asynchronous to the PIPE clock, any path going through it should be ignored for timing analysis.
pl_txelecidle	req_wake (pcie3_mfpowermgt module)	This path is used to transmit a beacon for L2 exit (requires CDC to be implemented)
pl_powerdown	When CDC is implemented: req_trans_r[2] & lpm_state (pcie3_mfpowermgt module)	When CDC is implemented: This path is used to exit from the P2 power state when PL_PCLK is turned off.
pl_powerdown	When CDC is not implemented: latch_p2_exit (pcie3_top_pipe or pcie3_tlb_pipe)	When CDC is not implemented: A latch is implemented to detect beacon/WAKE# signaling in L2, and the output of this latch is used to exit from the P2 power state when PL_PCLK is turned off.

Table 127: Unsynchronized paths