TJ-Series PCIe® Controller User Guide

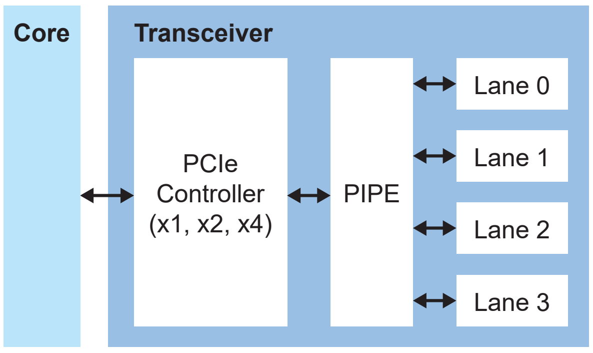
UG-TiPCIe-v1.6
June 2025
www.elitestek.com

Contents

Introduction

TJ-Series transceivers consist of a physical medium attachment (PMA) and a physical coding sublayer (PCS). The PMA connects the FPGA to the lane, generates the required clocks, and converts the data from parallel to serial or serial to parallel. The PCS contains the digital processing interface between the PMA and the FPGA fabric. The PCS supports SGMII, 10GBase-KR, and PCIe® Gen4 as well as PMA Direct. This user guide provides the specifications for the PCIe Controller interface.

Figure 1: Transceiver Used for PCIe



The following table shows the high-level controller configuration. It supports up to Gen4 x4, which is equivalent to a 16 Gbps lane rate or up to 64 Gbps link bandwidth.

Table 1: PCIe Controller Configuration

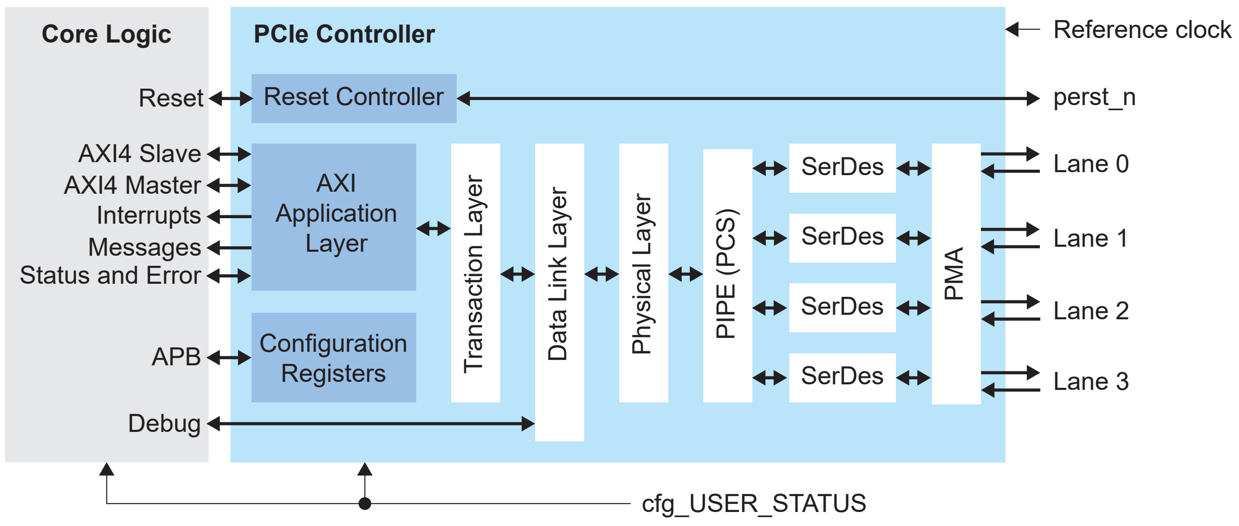
Parameter	Setting
Operational mode	Endpoint or root port[1]
Link width	x1, x2, x4
PIPE interface f_{MAX}	500 MHz, Gen 4 250 MHz, Gen 3 125 MHz, Gen 2 62.5 MHz, Gen 1
PCIe Controller core clock[2]	500 MHz
FPGA user clock (AXI interface) f_{MAX}[3]	125 - 250 MHz
FPGA user data path width (AXI interface)	256 bits
AXI interface address width	64 bits
Power management clock f_{MAX}	40 MHz

The PCIe Controller can be configured to be either endpoint (EP) or root port (RP) mode, depending on your requirements. PCIe operations initiated by the user side are driven through

(1) Root port capabilities are limited in the Efinity® software v2024.1.
(2) The PCIe Controller core clock is an internal clock.
(3) This clock, `AXI_CLK`, is available to the user application

the AXI4 slave port; PCIe operations initiated by the host side are driven through the AXI4 master port.

Figure 2: PCIe Controller Block Diagram



Note: Refer to **Appendix A: Acronyms and Abbreviations** on page 111 for terms used in this document. Refer to "PCI Express Interface" in the **TJ-Series Interfaces User Guide** for configuration options.

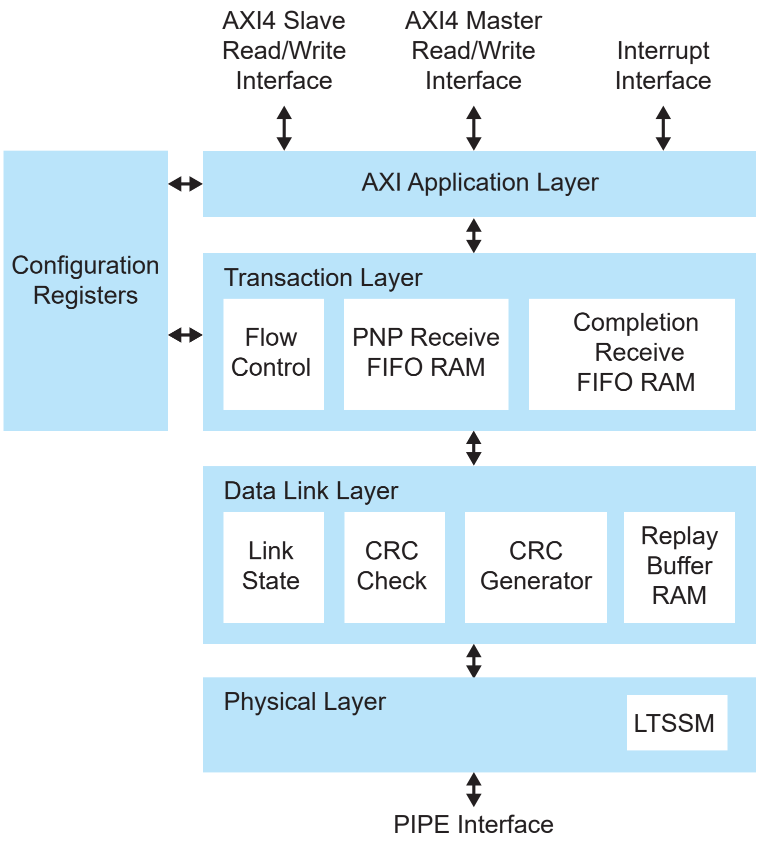
Features

- Fully integrated PMA with PIPE interface and controller (consisting of the physical layer, data link layer, and transaction layer)
- Programmable as endpoint (EP) or root port (RP)
- AXI4 slave interface port
- AXI4 master interface port
- Dedicated interrupt interface and inbound message interface supporting conventional interrupts, MSI, and MSI-X
- Supports:
  - Power management
  - Function-level reset (FLR)
  - SR-IOV
  - Up to four physical functions; each physical function can support up to 16 virtual functions
  - Up to 64 virtual functions
- Advanced error reporting (AER)
- TLP processing hints (TPH)
- Steering tag

Functional Description

The following figure shows a high-level overview of the PCIe Controller. This topic provides an overview of the layers, which are described in detail later.

Figure 3: PCIe Controller High-Level Block Diagram



Physical Layer

On the physical layer's receive (RX) side, data arrives from the link over the PIPE interface. For all link speeds, each lane is de-scrambled independently. The data from the lanes is de-skewed to generate aligned data. The PCIe Controller decodes the aligned data and sends the packets to the data link layer.

On the physical layer's transmit (TX) side, data arrives from the data link layer over a single interface. The PCIe Controller formats the data into packets by appending SOP and EOP, and aligns it on the outgoing lanes. For all link speeds, the data from each lane is scrambled independently before being transmitted on the outgoing PIPE. The physical layer has one instance of the Link Training and Status State Machine.

Data Link Layer

The data link layer receives packet data from the physical layer's RX. A CRC checker checks the incoming packet LCRC. The PCIe Controller sends the LCRC-stripped data to the transaction layer. A separate state machine performs the data link layer initialization.

On the TX side, the data link layer receives packets from the transaction layer over a 128-bit data path. It then adds the LCRC to the packets, multiplexes them with other data link layer packets (such as ACKs and flow control DLLPs), and forwards them to the physical layer. The TX side of the data link layer also has the replay buffer that is required for re-transmitting packets.

Transaction Layer

At the transaction layer, data arrives on the RX side from the data link layer. The arriving packets go into a receive FIFO buffer, and packet forwarding only begins when the FIFO buffer has a complete packet. Packets are decoded and forwarded to the appropriate host interface, or to an internal module (for example, interrupt messages). The host interfaces include separate interfaces for posted/non-posted (PNP) and completion packets.

The TX side of the transaction layer receives data from the client logic through separate interfaces for each type (posted/non-posted and completion). A state machine processes the data, schedules the packets, and forwards them over a common data path to the data link layer.

AXI Application Layer

The application layer provides a simple interface to a host bus or DMA engine on the user side. The application layer has three separate interfaces to the user logic:

- *Target memory read/write interface*—Provides a straightforward interface to the user memory controller or DMA engine. This interface also delivers I/O requests and messages received from the link to the client. EPs need this interface.
- *Master read/write interface*—Lets an EP generate memory transactions to the host as bus master; an RP can generate memory, I/O, configuration, and message requests. Devices that require bus master capability need this interface, e.g., all RPs and EPs that have master capability.
- *Interrupt interface*—Communicates the interrupt state between the user application and the PCIe Controller.

The application interface can maintain the state of up to 256 non-posted transactions (memory reads, I/O reads and writes, configuration reads and writes) generated on the master side, allowing their completions to be matched to the requests.

PCIe Controller Configuration

Many of the PCIe Controller's interfaces and features are user configurable with the Efinity Interface Designer. The settings you make in the Interface Designer are the defaults that the PCIe Controller uses when you power it up or perform a cold reset. You can also change many of the settings via the APB interface (if you enable it).

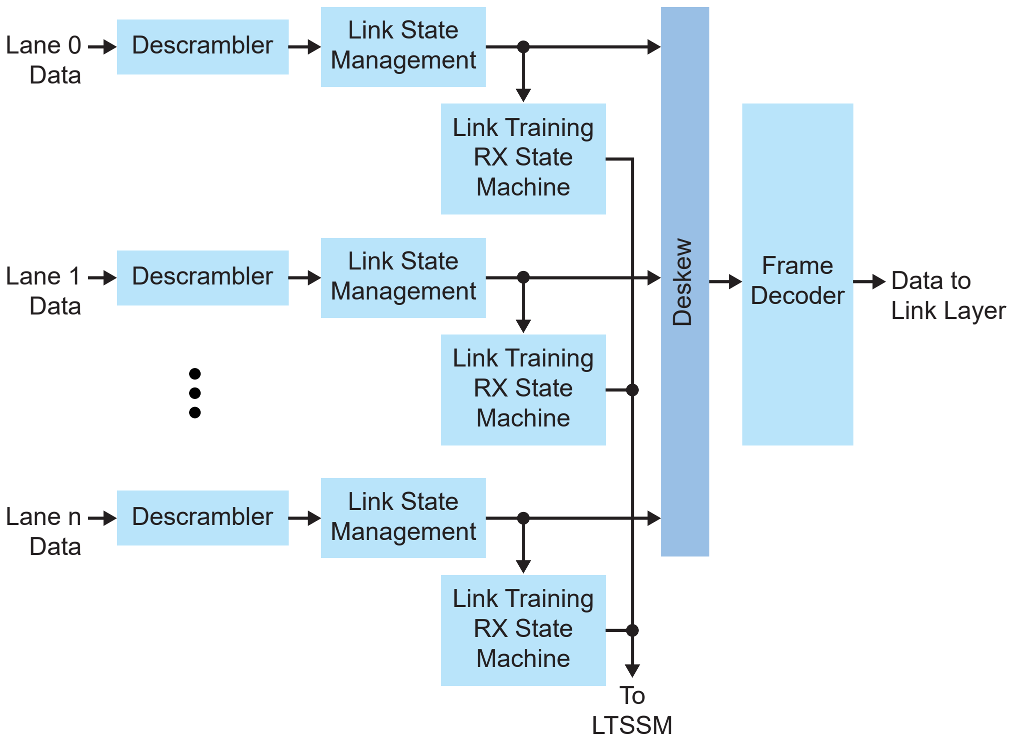
Learn more: Refer to the **TJ-Series Interfaces User Guide** for a complete description of the settings you can configure with the Interface Designer.

Physical Layer

Data arrives from the PIPE interface over one or more lanes. Each lane has a 32-bit interface and a clock frequency of 62, 125, 250, or 500 MHz depending on the link speed. The data flow happens as follows:

1. The data is converted to the core clock domain.
2. The PCIe Controller de-scrambles each lane's data independently.
3. Logic checks the data to detect any link power state transitions.
4. Tthe lanes are de-skewed using FIFOs that are aligned on SKP sequences. The lanes are aligned as a single unit.

Figure 4: RX PHY Layer

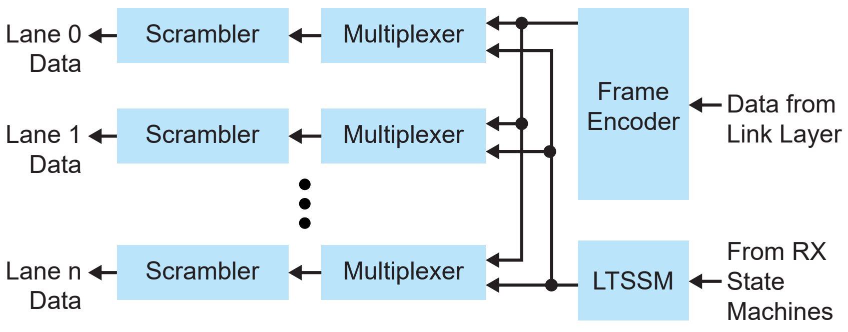


A frame decoder decodes the de-skewed data, removes the SOP/EOP framing delimiters from the packet, and aligns them on the internal data path. The frame decoder can handle varying link widths and all potential packet alignments on the lanes. The decoded data is sent to the data link layer with indicators for packet type and errors detected.

The PCIe Controller sends each lane's received data to the Link Training Receive State Machine, which detects and decodes any training sequences from the lane. Each state machine passes information extracted from the training sequences to the LTSSM.

On the TX side, data arrives from the data link layer over a 128-bit data path, plus sideband signals. A frame decoder adds SOP and EOP delimiters to the packets and aligns them on the lanes. The frame encoder can handle varying link widths and all legal packet alignments on the lanes.

Figure 5: TX PHY Layer



The PCIe Controller multiplexes outgoing packets from the frame encoder with training sequences generated by the LTSSM. The multiplexer disables the data path from the frame encoder during link training, and allows the LTSSM to control the lanes. Each lane has its own scrambler to scramble the data before sending it to the PIPE interface. The outgoing PIPE interface has 32 bits per lane for all speeds with a PIPE clock frequency of 62, 125, 250, or 500 MHz), which determines the link speed.

SRIS Operation

The PCIe Controller supports the Separate Reference Clock Independent Spread Spectrum (SRIS) ECN. With SRIS enabled, the PCIe Controller is in SRIS mode upon power-up and can transmit and receive the SKP ordered set (OS) as required by the SRIS specifications.

Note: The SRIS feature enables a higher clock tolerance from 600 ppm to 5,600 ppm in a separate reference clock configuration. Using an incompatible clock tolerance in your system may result in an unstable L0 state or packet timeout issues. Refer to the PCIe Base Specification 3.0 or higher for more information.

In SRIS mode, the PCIe Controller transmits SKP OS (as per the SRIS and PCIe 3.0 specifications) as follows:

- In 8b/10b encoding mode, the PCIe Controller transmits SKP OS every 128 symbols. If, due to a transmission of a large TLP, the SKP OS cannot be sent at the 128 symbol boundary, the controller accumulates all SKP OS and sends them at the end of the TLP.
- In 128/130b encoding mode, the PCIe Controller transmits SKP OS every 32 blocks. If, due to a transmission of a large TLP, the SKP OS cannot be sent at the 32 block boundary, the controller accumulates the SKP OS and send them at the end of the TLP.

On the RX side, the PCIe Controller can handle the higher frequency SKP OS reception as mandated by the SRIS specifications.

When SRIS mode is enabled, the following two features (as defined in the PCIe 3.0 specification) are changed:

- L0s capability is not advertised by the core in the link control register in PCI Express capability structure in the PCI compatible configuration space.
- The modified compliance pattern at 8G or higher is different. See the SRIS ECN specification for further details.

When the SRIS control register power-on default value is changed, the L0s capability in the Link Control register should be updated accordingly via the Local Management/APB interface.

The SRIS specification has an optional feature called Lower SKP OS generation/reception. The PCIe Controller implements this feature. With this feature, the PCIe device (if needed) can revert to the non-SRIS frequency of SKP OS generation when the PCIe link is in the L0 mode. This capability is advertised in the Lower SKP OS Generation/Reception Supported Speeds Vector field of the Link Capabilities 2 register. This feature is enabled/disabled using the Enable Lower SKP OS Generation Vector field of the Link Control 3 register.

When the SRIS feature is disabled using the SRIS control register, the Lower SKP OS Generation/Reception Supported Speeds Vector field of the Link Capabilities 2 register is disabled by forcing setting the value to zero.

Note: You can enable SRIS in the Interface Designer (**PCI Express block** > **Base tab** > **SRIS Enable**).

During operation, you can update the setting using the APB interface. As a control and debug feature, you can enable/disable the SRIS feature using a control register in the local management space (refer to "SRIS Control Register" in the "Local Management Registers" chapter of the **TJ-Series PCIe Controller Registers User Guide**). If you want to enable/disable SRIS mode, set/reset the `SRIS Enable` register field before link training begins.

Important: You cannot enable SRIS if active state power management (ASPM) is enabled.

RX Lane Margining

The Receiver lane margining enables system software to obtain the receiver's margin information while the link is in L0 state. The PCIe Controller:

- Supports RX lane margining for timing and voltage in either direction from the current RX position.
- Supports RX lane margining in endpoint and root port modes.
- Supports RX lane margining for PHYs that implement an independent error sampler. (i.e., `MIndErrorSampler ==1`). `MIndErrorSampler==0` is not supported.
- Supports all lanes being margined simultaneously.

- Supports PIPE interface revision 4.4.1 for margining.
- Implements programmable registers in the local management space for all PHY parameters related to margining.
- Implements the Lane Margining Capability Register Set in the PF0 configuration space at address offset 12'h920.
- Implements logic to detect and report invalid margining commands received from host software and the PHY.

Lane margining is driven by software. Software uses the Lane Margin Control and Status register in each port (downstream or upstream) for margin control and to obtain status information for the corresponding RX associated with the port.

When the host writes a new margining command to the Lane Margin Control Register, the PCIe Controller decodes the command and performs two checks to determine whether the command is:

- Valid:
  - Commands that do not match defined command formats are treated as invalid.
  - If the received command is invalid, the PCIe Controller discards the command and reports the error in LM register.
- Supported (step margin commands):
  - Checks if the step margin offset is within the range supported by the device.
  - If a step margin command is unsupported, the PCIe Controller responds with `NAK` status in the Lane Margin Status Register. An error is not flagged in the LM registers.

Note: Refer to **Exception Handling** on page 19 for more information on these checks.

The PCIe Controller then processes commands that are valid and supported. The PCIe Controller internally executes commands that do not require any action from the PHY, such as report commands. All other commands are delivered to the PHY over the PIPE interface.

The PCIe Controller responds to the host by updating the status appropriately in the Lane Margin Status Register. The register format is:

Table 2: Margining Lane Control and Status Register (i_margining_lane_control_status_regX)

[31:24]	[23]	[22]	[21:19]	[18:16]	[15:8]	[7]	[6]	[5:3]	[2:0]
MPSTS	R1	UMSTS	MTSTS	RNSTS	MRGPAY	R0	USGMOD	MRGTYP	RCVNUM
Margin Payload Status	Reserved	Usage Model Status	Margin Type Status	RX Number Status	Margin Payload	Reserved	Usage Model	Margin Type	RX Number

Command Processing (Endpoint)

The following table shows how the PCIe Controller processes margining commands in endpoint mode.

Table 3: Margining Command Processing in Endpoint Mode

Command (Margin Control Register)	PIPE Interface	Response (Margin Status Register)
No Command Margin Type [2:0]: 111b Receiver Number [2:0]: 000b Margin Payload [7:0]: 9Ch	No change	Margin Type [2:0]: 111b Receiver Number [2:0]: 000b Margin Payload [7:0]: 9Ch
Access Retimer Register Margin Type [2:0]: 001b Receiver Number [2:0]: 010b/100b Margin Payload [7:0]: XXh	No change	Invalid command for EP. Reported as an invalid command in LM register.
Report Margin Control Capabilities Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:0]: 88h	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:5]: 000b Margin Payload [4:0]: {MIndErrorSampler, MSampleReportingMethod, MIndLeftRightTiming, MIndUpDownVoltage, MVoltageSupported}
Report MNumVoltageSteps Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:0]: 89h	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7]: 0 Margin Payload [6:0]: MNumVoltageSteps
Report MNumTimingSteps Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:0]: 8Ah	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:6]: 00 Margin Payload [5:0]: MNumTimingSteps
Report MMaxTimingOffset Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:0]: 8Bh	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7]: 0 Margin Payload [6:0]: MMaxTimingOffset
Report MMaxVoltageOffset Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:0]: 8Ch	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7]: 0 Margin Payload [6:0]: MMaxVoltageOffset
Report MSamplingRateVoltage Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:0]: 8Dh	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:6]: 00 Margin Payload [5:0] = {MSamplingRateVoltage [5:0]}
Report MSamplingRateTiming Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:0]: 8Eh	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:6]: 00 Margin Payload [5:0]: {MSamplingRateTiming [5:0]}
ReportMSampleCount Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:0]: 8Fh	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7]: 0 Margin Payload [6:0]: MSampleCount

Command (Margin Control Register)	PIPE Interface	Response (Margin Status Register)
ReportMMaxLanes Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:0]: 90h	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 110b Margin Payload [7:5]: 00 Margin Payload [6:0]: MMaxLane
Set Error Count Limit Margin Type [2:0]: 010b Receiver Number [2:0]: 110b Margin Payload [7:6]: 11b Margin Payload [5:0]: Error Count Limit	No change Register Error Limit [5:0] in the Local Management Register in the core clock domain.	Margin Payload [7:6]: 11b Margin Payload [5:0]: Error Count Limit registered by the target receiver
Go to Normal Settings Margin Type [2:0]: 010b Receiver Number [2:0]: 000b or 110b Margin Payload [7:0]: 0Fh	• Write committed to RX Margin Control 0 with Start Margin = 0. • Wait for Write Ack response from PHY or a 10 ms timeout. • Wait for PHY2MAC write committed to Margin Status or a 10 ms timeout.	Margin Type [2:0]: 010b Receiver Number [2:0]: 110b Margin Payload [7:0]: 0Fh
Clear Error Log Margin Type [2:0]: 010b Receiver Number [2:0]: 000b or 110b Margin Payload [7:0]: 55h	• Write committed to RX Margin Control 0 with Error Count Reset = 1. Other fields picked up from the RX Margin Control 0 Mirror Register. • Wait for Write Ack response from PHY or a 10 ms timeout.	Margin Type [2:0]: 010b Receiver Number [2:0]: 110b Margin Payload [7:0]: 55h
Step Margin to Timing Offset to Right/Left of Default Margin Type [2:0]: 011b Receiver Number [2:0]: 110b Margin Payload [7:0]: XX	If a step margin to voltage is already in progress or if a step margin to timing in the opposite direction is in progress: • Stop margining by issuing write committed to RX Margin Control 0 with Start Margin = 0. • Other fields picked up from the RX Margin Control 0 Mirror Register. • Wait for Write Ack response from PHY or a 10 ms timeout. • Wait for PHY2MAC Write Committed to Margin Status or a 10 ms timeout. Check if RX margin command is supported. If margin offset is supported: • Issue Write Uncommitted to RX Margin Control 1 with Margin Offset [6:0] = Margin Payload [5:0]. • Issue Write Committed to RX Margin Control 0 with StartMargin: 1 and MarginTiming: 1. • Wait for Write Ack response from PHY or a 10ms timeout. • Wait for PHY2MAC Write Committed to Margin Status or a 10 ms timeout. Otherwise: • Issue NAK Status and exit.	Margin Type [2:0]: 011b Receiver Number [2:0]: 110b IF (Unsupported Range in Command) Margin Payload [7:6]: 11 ELSIF (Write ACK received for Margin Command) Margin Payload [7:6]: 01 ELSIF (PIPE MAC RX Margin Register 0 Margin Status) Margin Payload [7:6]: 10 ELSIF (Error Count > Limit) Margin Payload [7:6]: 00 Margin Payload [5:0]: Error Count from RX Margin Status 2 Register

Command (Margin Control Register)	PIPE Interface	Response (Margin Status Register)
Step Margin to Voltage Offset to Up/ Down of Default Margin Type [2:0]: 100b Receiver Number [2:0]: 110b Margin Payload [7:0]: XX	If a step margin to timing is already in progress or if a step margin to voltage in the opposite direction is in progress: • Stop Margining by issuing Write Committed to RX Margin Control 0 with Start Margin = 0. Other fields picked up from the RX Margin Control 0 Mirror Register. • Wait for Write Ack response from PHY or a 10ms timeout. • Wait for PHY2MAC Write Committed to Margin Status or a 10 ms timeout. Check if an RX margin command is valid. If margin offset is supported: • Issue Write Uncommitted to RX Margin Control 1 with Margin Offset [6:0] = Margin Payload [6:0]. • Issue Write Committed to RX Margin Control 0 with StartMargin: 1 and MarginVoltage: 1 • Wait for Write Ack response from PHY or a 10ms timeout. • Wait for PHY2MAC Write Committed to Margin Status or a 10 ms timeout. Otherwise: • Issue NAK Status and exit.	Margin Type [2:0]: 100b Receiver Number [2:0]: 110b IF (Unsupported Range in Command) Margin Payload [7:6]: 11 ELSIF (Write ACK received for Margin Command) Margin Payload [7:6]: 01 ELSIF (PIPE MAC RX Margin Register 0 Margin Status) Margin Payload [7:6]: 10 ELSIF (Error Count > Limit) Margin Payload [7:6]: 00 Margin Payload [5:0]: Error Count from RX Margin Status 2 Register
Vendor Defined Margin Type [2:0]: 101b Receiver Number [2:0]: 110b Margin Payload [7:0]: Vendor Defined	No change	Margin Type [2:0]: 101b Receiver Number [2:0]: 110b Margin Payload [7:0]: Vendor Defined Margin Payload status same as received in control register.

Command Processing (Root Port)

The following table shows how the PCIe Controller processes margining commands in root port mode.

Table 4: Command Processing in Roor Port Mode

Command (Margin Control Register)	PIPE Interface	Response (Margin Status Register)
No Command Margin Type [2:0]: 111b Receiver Number [2:0]: 000b Margin Payload [7:0]: 9Ch	No change	Margin Type [2:0]: 111b Receiver Number [2:0]: 000b Margin Payload [7:0]: 9Ch
Access Retimer Register Margin Type [2:0]: 001b Receiver Number [2:0]: 010b/100b Margin Payload [7:0]: XXh	No change	Command sent on Control SKP sent by downstream port. Margin Status Updated from the Control SKP OS received by the downstream port.
Report Margin Control Capabilities Margin Type [2:0]: 001b Receiver Number [2:0]: 001b through 101b Margin Payload [7:0]: 88h	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7:5]: 000b Margin Payload [4:0]: {MIndErrorSampler, MSampleReportingMethod, MIndLeftRightTiming, MIndUpDownVoltage, MVoltageSupported}

Command (Margin Control Register)	PIPE Interface	Response (Margin Status Register)
Report MNumVoltageSteps Margin Type [2:0]: 001b Receiver Number [2:0]: 001b through 101b Margin Payload [7:0]: 89h	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7]: 0 Margin Payload [6:0]: MNumVoltageSteps
Report MNumTimingSteps Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7:0]: 8Ah	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7:6]: 00 Margin Payload [5:0]: MNumTimingSteps
Report MMaxTimingOffset Margin Type [2:0]: 001b Receiver Number [2:0]: 001b through 101b Margin Payload [7:0]: 8Bh	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7]: 0 Margin Payload [6:0]: MMaxTimingOffset
Report MMaxVoltageOffset Margin Type [2:0]: 001b Receiver Number [2:0]: 001b through 101b Margin Payload [7:0]: 8Ch	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7]: 0 Margin Payload [6:0]: MMaxVoltageOffset
Report MSamplingRateVoltage Margin Type [2:0]: 001b Receiver Number [2:0]: 001b through 101b Margin Payload [7:0]: 8Dh	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7:6]: 00 Margin Payload [5:0]: {MSamplingRateVoltage [5:0]}
Report MSamplingRateTiming Margin Type [2:0]: 001b Receiver Number [2:0]: 001b through 101b Margin Payload [7:0]: 8Eh	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7:6]: 00 Margin Payload [5:0]: {MSamplingRateTiming [5:0]}
ReportMSampleCount Margin Type [2:0]: 001b Receiver Number [2:0]: 001b through 101b Margin Payload [7:0]: 8Fh	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7]: 0 Margin Payload [6:0]: MSampleCount
ReportMMaxLanes Margin Type [2:0]: 001b Receiver Number [2:0]: 001b through 101b Margin Payload [7:0]: 90h	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7:5]: 00 Margin Payload [6:0]: MMaxLane
Set Error Count Limit Margin Type [2:0]: 010b Receiver Number [2:0]: 001b through 101b Margin Payload [7:6]: 11b Margin Payload [5:0]: Error Count Limit	No change	Margin Type [2:0]: 001b Receiver Number [2:0]: 001b Margin Payload [7:6]: 11b Margin Payload [5:0]: Error Count Limit registered by the target Receiver
Go to Normal Settings Margin Type [2:0]: 010b Receiver Number [2:0]: 000b through 101b Margin Payload [7:0]: 0Fh	• Write Committed to RX Margin Control 0 with Start Margin = 0. • Wait for Write Ack response from PHY or a 10 ms timeout. • Wait for PHY2MAC Write Committed to Margin Status or a 10 ms timeout.	Margin Type [2:0]: 010b Receiver Number [2:0]: 001b Margin Payload [7:0]: 0Fh

Command (Margin Control Register)	PIPE Interface	Response (Margin Status Register)
Clear Error Log Margin Type [2:0]: 010b Receiver Number [2:0]: 000b through 101b Margin Payload [7:0]: 55h	• Write Committed to RX Margin Control 0 with Error Count Reset = 1. Other fields picked up from the RX Margin Control 0 Mirror Register. • Wait for Write Ack response from PHY or a 10 ms timeout.	Margin Type [2:0]: 010b Receiver Number [2:0]: 001b Margin Payload [7:0]: 55h
Step Margin to Timing Offset to Right/Left of Default Margin Type [2:0]: 011b Receiver Number [2:0]: 001b through 101b Margin Payload [7:0]: XX	If a step margin to voltage is already in progress or if a step margin to timing in the opposite direction is in progress: • Stop Margining by issuing Write Committed to RX Margin Control 0 with Start Margin = 0. Other fields picked up from the RX Margin Control 0 Mirror Register. • Wait for Write Ack response from PHY or a 10 ms timeout. • Wait for PHY2MAC Write Committed to Margin Status or a 10 ms timeout. Check if RX margin command is supported. If margin offset is supported: • Issue Write Uncommitted to RX Margin Control 1 with Margin Offset [6:0] = Margin Payload [5:0]. • Issue Write Committed to RX Margin Control 0 with StartMargin: 1 and MarginTiming: 1. • Wait for Write Ack response from PHY or a 10 ms timeout. • Wait for PHY2MAC Write Committed to Margin Status or a 10 ms timeout. Otherwise: • Issue NAK Status and exit.	Margin Type [2:0]: 011b Receiver Number [2:0]: 001b IF (Unsupported Range in Command) Margin Payload [7:6]: 11 ELSIF (Write ACK received for Margin Command) Margin Payload [7:6]: 01 ELSIF (PIPE MAC RX Margin Register 0 Margin Status) Margin Payload [7:6]: 10 ELSIF (Error Count > Limit) Margin Payload [7:6]: 00 Margin Payload [5:0]: Error Count from RX Margin Status 2 Register
Step Margin to Voltage Offset to Up/Down of Default Margin Type [2:0]: 100b Receiver Number [2:0]: 001b through 110b Margin Payload [7:0]: XX	If a step margin to timing is already in progress or if a step margin to voltage in the opposite direction is in progress: • Stop Margining by issuing Write Committed to RX Margin Control 0 with Start Margin = 0. Other fields picked up from the RX Margin Control 0 Mirror Register. • Wait for Write Ack response from PHY or a 10 ms timeout. • Wait for PHY2MAC Write Committed to Margin Status or a 10 ms timeout. Check if RX margin command is valid. If margin offset is supported: • Issue Write Uncommitted to RX Margin Control 1 with Margin Offset [6:0] = Margin Payload [6:0]. • Issue Write Committed to RX Margin Control 0 with StartMargin: 1 and MarginVoltage: 1 • Wait for Write Ack response from PHY or a 10 ms timeout. • Wait for PHY2MAC Write Committed to Margin Status or a 10 ms timeout. Otherwsie: • Issue NAK Status and exit.	Margin Type [2:0]: 100b Receiver Number [2:0]: 001b IF (Unsupported Range in Command) Margin Payload [7:6]: 11 ELSIF (Write ACK received for Margin Command) Margin Payload [7:6]: 01 ELSIF (PIPE MAC RX Margin Register 0 Margin Status) Margin Payload [7:6]: 10 ELSIF (Error Count > Limit) Margin Payload [7:6]: 00 Margin Payload [5:0]: Error Count from RX Margin Status 2 Register
Vendor Defined Margin Type [2:0]: 101b Receiver Number [2:0]: 001b Margin Payload [7:0]: Vendor Defined	No change	Margin Type [2:0]: 101b Receiver Number [2:0]: 001b Margin Payload [7:0]: Vendor Defined Margin Payload status same as received in control register.

Step Margin Command Execution

Figure 6: Step Margin for Timing Command Execution

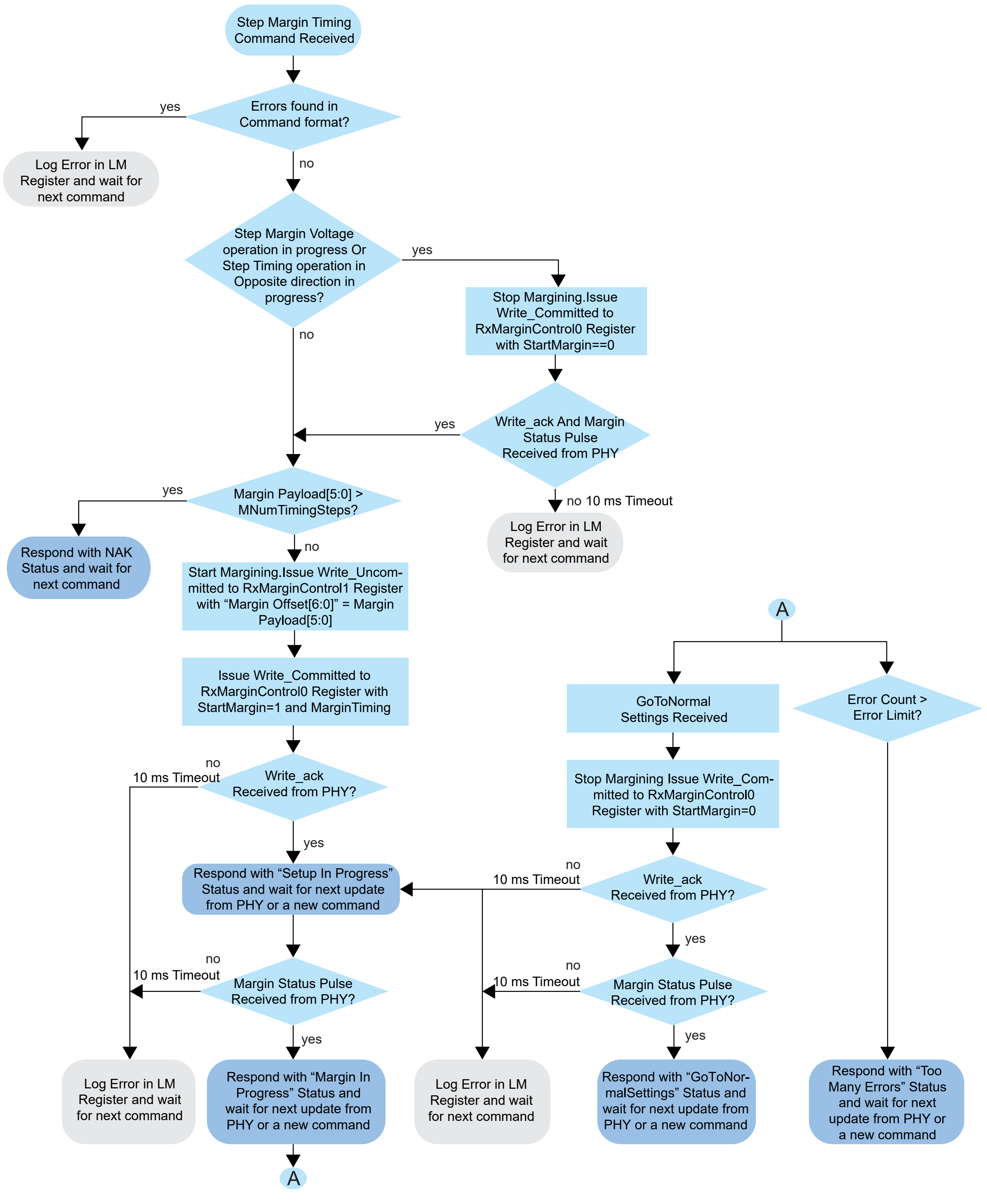
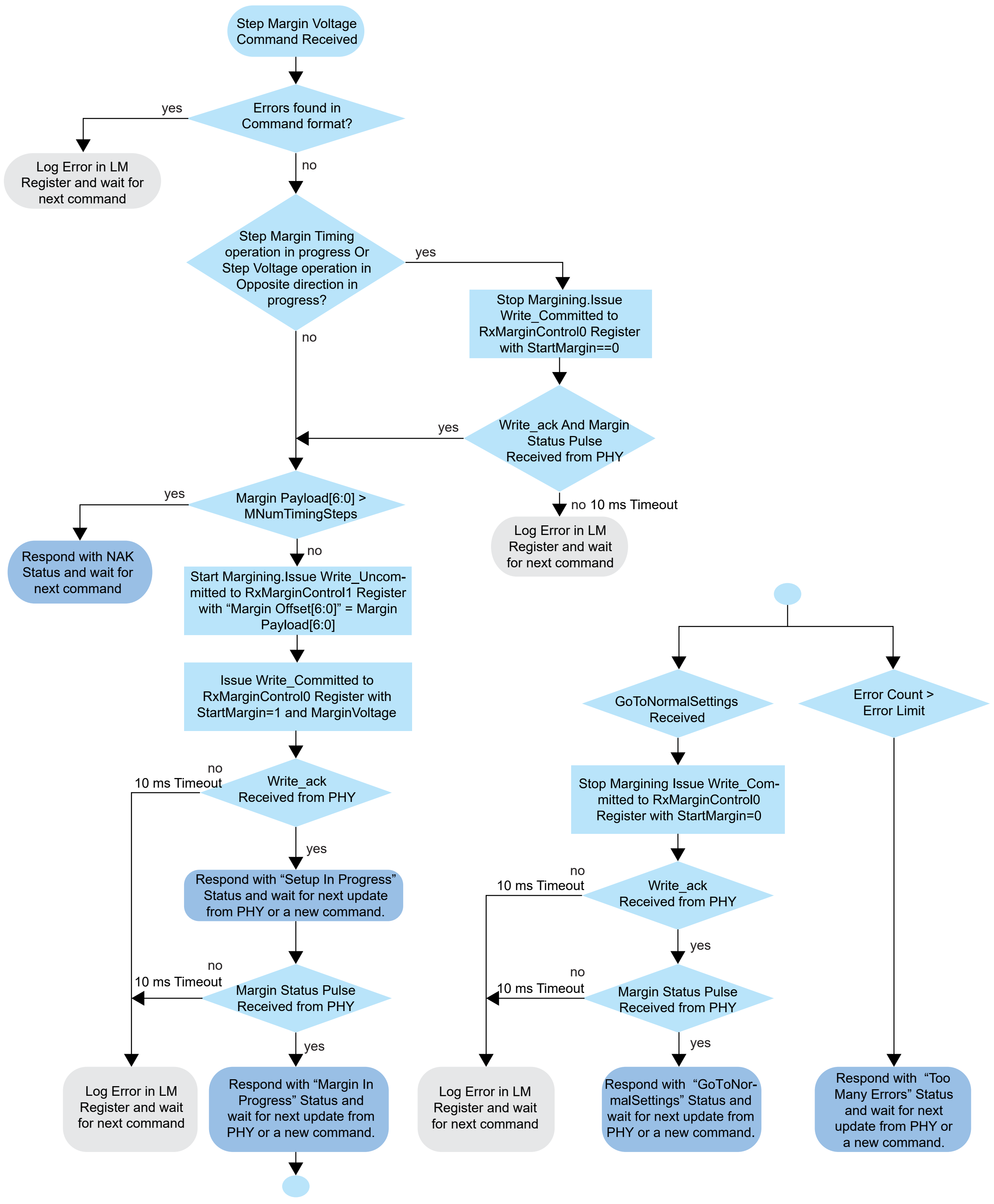

Step Margin Timing Command Received
Errors found in Command format?
yes
Log Error in LM Register and wait for next command
no
Step Margin Voltage operation in progress Or Step Timing operation in Opposite direction in progress?
yes
Stop Margining.Issue Write_Committed to RxMarginControl0 Register with StartMargin==0
no
Write_ack And Margin Status Pulse Received from PHY
yes
no 10 ms Timeout
Log Error in LM Register and wait for next command
Margin Payload[5:0] > MNumTimingSteps?
yes
Respond with NAK Status and wait for next command
no
Start Margining.Issue Write_Uncommitted to RxMarginControl1 Register with "Margin Offset[6:0]" = Margin Payload[5:0]
Issue Write_Committed to RxMarginControl0 Register with StartMargin=1 and MarginTiming
Write_ack Received from PHY?
no 10 ms Timeout
yes
Respond with "Setup In Progress" Status and wait for next update from PHY or a new command
Margin Status Pulse Received from PHY?
no 10 ms Timeout
yes
Log Error in LM Register and wait for next command
Respond with "Margin In Progress" Status and wait for next update from PHY or a new command
A
A
GoToNormal Settings Received
Stop Margining Issue Write_Committed to RxMarginControl0 Register with StartMargin=0
Write_ack Received from PHY?
no 10 ms Timeout
yes
Margin Status Pulse Received from PHY?
no 10 ms Timeout
yes
Log Error in LM Register and wait for next command
Respond with "GoToNormalSettings" Status and wait for next update from PHY or a new command
Error Count > Error Limit?
Respond with "Too Many Errors" Status and wait for next update from PHY or a new command

Figure 7: Step Margin for Voltage Command Execution


Step Margin Voltage Command Received
Errors found in Command format?
yes
Log Error in LM Register and wait for next command
no
Step Margin Timing operation in progress Or Step Voltage operation in Opposite direction in progress?
yes
Stop Margining.Issue Write_Committed to RxMarginControl0 Register with StartMargin==0
no
Write_ack And Margin Status Pulse Received from PHY
yes
no 10 ms Timeout
Log Error in LM Register and wait for next command
Margin Payload[6:0] > MNumTimingSteps
yes
Respond with NAK Status and wait for next command
no
Start Margining.Issue Write_Uncommitted to RxMarginControl1 Register with "Margin Offset[6:0]" = Margin Payload[6:0]
Issue Write_Committed to RxMarginControl0 Register with StartMargin=1 and MarginVoltage
no
10 ms Timeout
Write_ack Received from PHY
yes
Respond with "Setup In Progress" Status and wait for next update from PHY or a new command.
no
10 ms Timeout
Margin Status Pulse Received from PHY
yes
Log Error in LM Register and wait for next command
Respond with "Margin In Progress" Status and wait for next update from PHY or a new command.
GoToNormalSettings Received
Error Count > Error Limit
Stop Margining Issue Write_Committed to RxMarginControl0 Register with StartMargin=0
no
10 ms Timeout
Write_ack Received from PHY
yes
no
10 ms Timeout
Margin Status Pulse Received from PHY
yes
Log Error in LM Register and wait for next command
Respond with "GoToNormalSettings" Status and wait for next update from PHY or a new command.
Respond with "Too Many Errors" Status and wait for next update from PHY or a new command.

Step Margin Execution Status

The step margin execution status is updated when a write committed is received from the PHY. The 2-bit status is derived as shown in the following table.

Table 5: Step Margin Status

Inputs	Step Margin Execution Status [1:0]	Description
PHY issues Write_Ack in response to a write committed by the PCIe Controller to start margining.	01	01b: Set up for margin in progress. The receiver is getting ready but has not yet started executing the step margin command. MErrorCount is 0.
PHY sets the Margin Status bit in RX Margin Status 0 PIPE MAC Register in response to a write committed by the PCIe Controller to start margining.	10	10b: Margining in progress. The receiver is executing the step margin command. MErrorCount reflects the number of errors detected.
PHY sets the margin NAK bit in the RX Margin Status 0 PIPE MAC Register in response to a write committed by the PCIe Controller to start margining.	11	11b: NAK. Indicates that an unsupported lane margining command was issued. For example, timing margin beyond +/- 0.2 UI. MErrorCount is 0.
PHY updates error count bits [5:0] by issuing a write committed to the MAC RX Margin Status 2 Register. When error count bits [5:0] is greater than error limit[5:0], update execution status.	00	00b: Too many errors. The receiver autonomously went back to its default settings. MErrorCount reflects the number of errors detected. Note that MErrorCount might be greater than the error count limit.

Control SKIP for Lane Margining at Receiver

The Step Margin Execution Status is updated when write committed is received from the PHY. The 2-bit status is derived as shown in the following tables.

Table 6: Control SKP Ordered Set Format

Where *N* is 1 - 5.

Symbol Number	Value	Description
0 through (4**N* - 1)	AAh	SKP Symbol. Symbol 0 is the SKP Ordered Set identifier.
4**N*	78h	SKP_END_CTL Symbol. Signifies the end of the Control SKP Ordered Set after three more symbols.
4**N* + 1	00-FFh	Bit 7: Data Parity Bit 6: First Retimer Data Parity Bit 5: Second Retimer Parity Bits [4:0]: Margin CRC [4:0]
4**N* + 2	00-FFh	Bit 7: Margin Parity Bit 6: Usage Model : Set to 0b to indicate RX Lane Margining Bit [5:3]: Margin Type Bits [2:0]: Receiver Number
4**N* + 3	00-FFh	Bits [7:0] : Margin Payload

Table 7: Control SKP during Root Port and Endpoint Mode

Types	Root Port Mode	Endpoint Mode
Control SKP TX	The contents of the four control fields of the Lane Margin Control and Status Register in the downstream port are always shown in the identical fields in the transmitted Control SKP Ordered Sets.	The Control SKIP is always transmitted with No Command.

Types	Root Port Mode	Endpoint Mode
Control SKP RX	The PCIe Controller checks the Margin CRC and Margin Parity in the received Control SKIP Ordered Sets. Any mismatch detected is reported in the Lane Error Status Register. The contents of the Control SKP Ordered Set received in the downstream port is reflected in the corresponding status fields of the Lane Margin Control and Status Register in the downstream port if either of these conditions are met in the Lane Margin Control and Status Register: • Receiver number is 010b - 101b. • Receiver number is 000b, margin command is clear, rrror log is no command or Go to Normal Settings, and there are retimer(s) in the link.	The PCIe Controller checks the margin CRC and margin parity in the received Control SKIP Ordered Sets. Detected mismatches are reported in the Lane Error Status Register. The contents of the Control SKIP Ordered Sets are ignored in endpoint mode.

Exception Handling

When host software writes a command into the PCIe Controller's Lane Margin Control Register, the PCIe Controller performs a Command Valid Check and a Command Supported Check.

Command Valid Check

The PCIe Controller checks for errors in the Margining Lane Control Register. Commands that do not match any defined formats are treated as invalid. If the host software writes an invalid command to the register, the PCIe Controller detects and reports the error in the Local Management Register Margining Error Status 1 Register.

The PCIe Controller logs the first error and sets the error status bit. The software must clear this status bit needs before another error can be logged. PCIe Controller continues to accept subsequent commands written by software in the Margining Lane Control Register regardless of previous errors.

Table 8: Valid Commands for EndPoint Mode

Any other commands are considered invalid.

Margin Command	Margin Type[2:0]	Receiver Number[2:0]	Margin Payload[7:0]
No Command	111	000	9Ch
Report	001	110	88h to 90h
SetErrorCountLimit	010	110	{11xx_xxxx}
GoToNormalSettings	010	000, 110	0Fh
ClearErrorLog	010	000, 110	55h
StepMarginTimingOffset	011	110	{xxxx_xxxx}
StepMarginVoltageOffset	100	110	{xxxx_xxxx}
VendorDefined	101	110	{xxxx_xxxx}

Table 9: Valid Commands for Root Port Mode

Any other commands are considered invalid.

Margin Command	Margin Type[2:0]	Receiver Number[2:0]	Margin Payload[7:0]
No Command	111	000	9Ch
Access Retimer	001	010, 100	{xxxx_xxxx}
Report	001	001 through 101	88h to 90h
SetErrorCountLimit	010	001 through 101	{11xx_xxxx}
GoToNormalSettings	010	000 through 101	0Fh
ClearErrorLog	010	000 through 101	55h
StepMarginTimingOffset	011	001 through 101	{xxxx_xxxx}
StepMarginVoltageOffset	100	001 through 101	{xxxx_xxxx}

Margin Command	Margin Type[2:0]	Receiver Number[2:0]	Margin Payload[7:0]
VendorDefined	101	001 through 101	{xxxx_xxxx}

Command Supported Check

The PCIe Controller performs this check for Step Margin commands. When it receives a valid Step Margin command, the PCIe Controller further checks whether the Step Margin Offset is within the supported range. If a Step Margin command is unsupported, the PCIe Controller responds with NAK status in the Lane Margin Status Register. No error is flagged in LM registers in this case.

- The Step Margin Timing check is: Check Margin Payload[5:0] <= MNumTimingSteps
- The Step Margin Voltage check is: Check Margin Payload[6:0] <= MNumVoltageSteps

RX Margining PIPE Interface: Write Ack Timeout

During the execution of the GoToNormalSettings, ClearErrorLog, StepMarginTimingOffset or StepMarginVoltageOffset commands, the PCIe Controller issues a write committed command to the PHY over the PIPE interface and waits for the PHY to respond with WriteAck.

PCIe Controller waits for 10 ms to receive a WriteAck. If it is not received, the PCIe Controller reports an error in the Local Management Register Margining Error Status 2 Register.

Link Transition from Gen4 L0 State

The PCIe Controller only accepts margining commands when the link is in Gen4 L0 state. After the command is accepted, the PCIe Controller continues processing the command as long as the link remains in the Gen4 L0 or Recovery states.

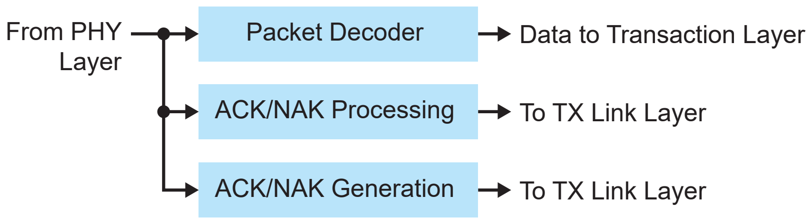
While a margining command is being processed, if the link transitions out of the Gen4 L0 or Recovery states, the PCIe Controller:

- Terminates all margining commands in progress.
- Resets all state machines related to RX Lane Margining to their default states.
- Resets all PIPE MAC registers, defined for RX Lane Margining, to their default values.
- Leaves the margining status in Lane Margin Control and Status Register at the last valid status just before the link state transition.

Data Link Layer

On the RX side, the data from the link passes through a decoder state machine for each link. The decoders verify the packet integrity by matching the received CRC with the generated CRC, and comparing their sequence numbers with the expected values (for TLPs). There is a separate CRC module for TLPs (32-bit CRC) and DLLPs (16-bit CRC).

Figure 8: RX Data Link Layer



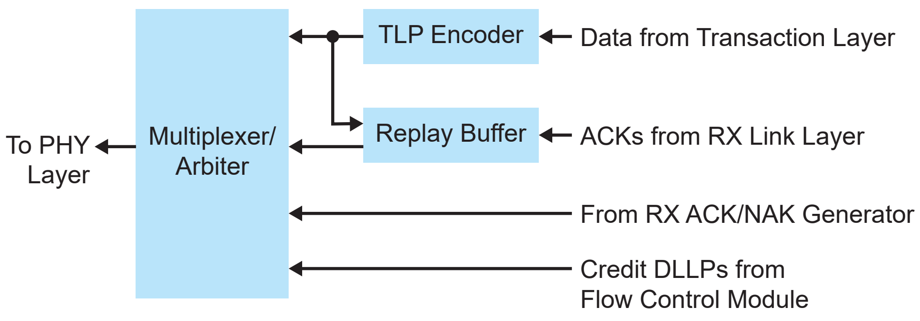
The packet decoder extracts the sequence number, CRC check, and strips the CRC.

After comparing and removing the link-layer CRC, the DLLP decoders pass the received packet to its target module based on the packet type. The DLLP decoders pass all TLPs they receive onto the the transaction layer (after first removing their sequence number and LCRC fields). The PCIe Controller processes data link layer acknowledgements (ACKs and NAKs) within the data link layer itself, and sends credit DLLPs to the flow control module.

After the CRC check, the PCIe Controller sends arriving data link layer acknowledgements to the transmit side for processing. Logic on the transmit side matches the acknowledgements with outstanding packets and handles any errors.

The receive side also generates acknowledgements (ACKs and NAKs) for the received TLPs. The PCIe Controller sends these packets to the transmit side where they are multiplexed with outgoing TLPs.

Figure 9: TX Data Link Layer


To PHY Layer
Multiplexer/ Arbiter
TLP Encoder
Replay Buffer
Data from Transaction Layer
ACKs from RX Link Layer
From RX ACK/NAK Generator
Credit DLLPs from Flow Control Module

The TLP encoder adds the sequence number and CRC.

On the TX side, the PCIe Controller formats the received transaction layer data for transmission to the physical layer by inserting a sequence number and CRC. The PCIe Controller multiplexes the formatted TLPs with other outgoing DL packets—such as ACKs, NAKs, and credit packets—and sends them to the physical layer over a data path shared by both links.

The TX side also contains the replay buffer associated with the link. The reply buffer is responsible for re-transmitting packets when needed and uses an external single-port RAM for storing the packets. There is also an internal pointer RAM for keeping track of packets stored in the replay buffer.

The data link layer also generates power management DLLPs to facilitate transitions of the link to the L1 and L2 states.

Data Link Feature Exchange

The PCIe Controller supports the data link feature exchange as per the PCI Express Base 4 specification. You enable/disable this feature by programming the DL Feature Exchange Enable bit in the dl_feature_capabilities_reg Configuration register. When enabled, the PCIe Controller's Data Link Control and Management State Machine enters the DL_Feature state from the DL_Inactive state after the LTSSM is in L0. In the DL_Feature state, the PCIe Controller transmits data link feature DLLPs continuously. It does not transmit any other TLPs or DLLPs in this state.

The PCIe Controller transitions from the DL_Feature state to the DL_Init state when a DL Feature DLLP is received with the feature Ack bit set. If the remote end device does not support DL Feature Exchange, the PCIe Controller transitions from the DL_Feature state to the DL_Init State when it receives a a InitFC1 DLLP.

The PCIe Controller supports the Scaled Flow Control Data Link features as described in the following topics.

RX Scaled Flow Control

The PCIe Controller advertises the maximum available header and payload credit limits for posted, non-posted, and completion RX buffers. When scaled flow control is activated, the PCIe Controller advertises a scale factor of 01 by default.

The firmware can:

- Override the scale and limit values prior to link training.
- Program the scale factor in the Local Management DL Layer flow control scaling management Register.
- Program the limit value in the Local Management Receive Credit Limit Register 0/1.

Additionally, the firmware must ensure that the programmed values do not exceed the maximum credits that were present upon reset.

If Flow Control Scaling is not activated during DL Feature Exchange, the PCIe Controller overrides the programmed scale factor with 00. The programmed credit values are adjusted to the scale factor of 00.

TX Scaled Flow Control

The PCIe Controller captures the current posted, non-posted, and completion limit values received in the INIT_FC and UPDATE_FC DLLPs.

Firmware can read:

- Received credit limit in Local Management Transmit Credit Limit Register 0/1
- Received credit scale factors in the Local Management DL Layer flow control scaling management register

TX Flow Control Error Handling

The PCIe Controller detects the following errors and reports them as flow control protocol errors:

- An RX that does not support scaled flow control must never cumulatively issue more than 2,047 data payload outstanding unused credits to the TX or 127 unused header credits. Additionally The RX must never cumulatively issue more outstanding unused data or headers to the TX than the maximum credit values based on the scaled flow control scaling factors. The PCIe Controller checks for violations of this rule and reports a Flow Control Protocol Error (FCPE).
- If scaled flow control is activated for a virtual channel, the HdrScale and DataScale fields in the UpdateFCs must match the values advertised during initialization. The PCIe Controller checks for violations of this rule and reports a Flow Control Protocol Error (FCPE).

The PCIe Controller does not support infinite credit advertisement.

Aggregating ACK DLLPs

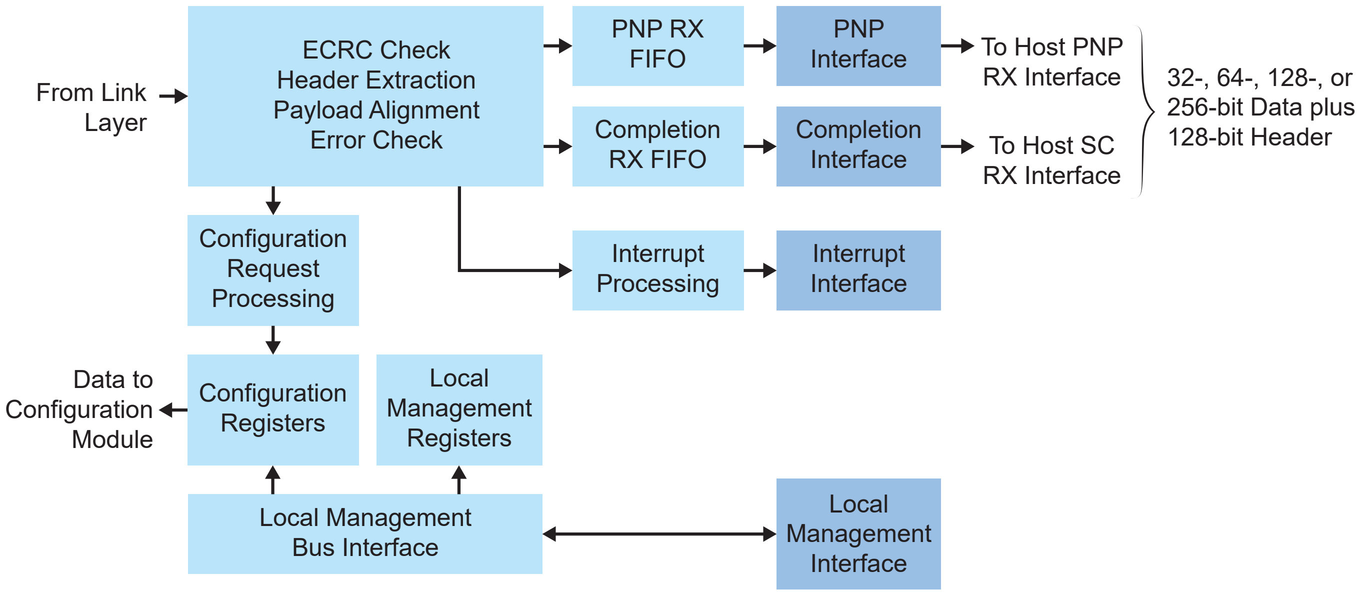
The PCIe Controller supports ACK aggregation in certain conditions:

- Typically, the PCIe Controller schedules one ACK DLLP for transmission for each TLP that it receives.
- If the TX is idle, it transmits an ACK DLLP immediately. However, if the TX is busy with a TLP transmission, the ACK DLLP waits till the ongoing TLP is completely transmitted.
- While the ACK DLLP is waiting for transmission, if the TX receives another TLP, the PCIe Controller aggregates the two pending ACK DLLPs into a single ACK DLLP with the higher sequence number.

Transaction Layer

On the RX side, data arrives from the data link layer over a 128-bit data path. Logic in the transaction layer decodes the packet header, performs ECRC check when the packet has a TLP digest, and aligns the payload on the data path. The data then goes through store-and-forward FIFO buffers. The PCIe Controller has separate FIFO buffers for posted, non-posted, and completion packets.

Figure 10: RX Transaction Layer



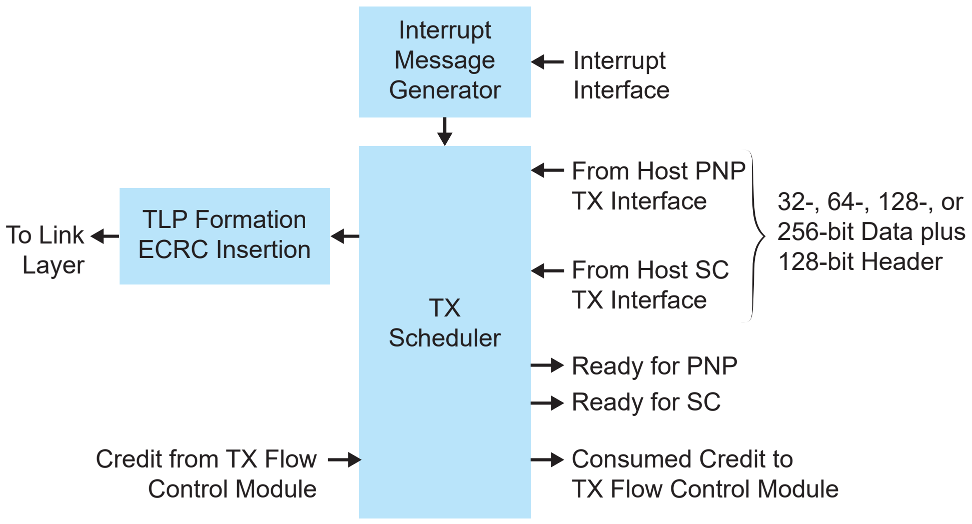
The PCIe Controller only reads a packet from the FIFO when the entire packet has been received. The decoding logic classifies packets based on their TLP header and forwards them to the appropriate modules.

The PCIe Controller processes all read/write requests to configuration registers within the transaction layer, which routes these requests to the register set of the function addressed by the request, and returns completion packets back to the link. All interrupt-related messages are processed by a separate interrupt processing module, which controls the interrupt interface. An error handling module processes error messages.

The RX flow control parameters (payload and header credit for posted, non- posted, and completion) are set based on the available space in the receive FIFO buffers. The flow control protocol ensures that the FIFO buffers do not overflow. The FIFO buffers communicate their state to the flow control module so that when the packet is forwarded out of the FIFO buffers, the corresponding credit becomes available and can be advertised to the link.

In the TX side of the transaction layer, PNP requests and completion (SC) packets arrive from the host over separate interfaces. The transaction layer multiplexes the packets, inserts the TLP header (and optionally the ECRC) and forwards them to the data link layer. The completions and messages generated are multiplexed on the same data path to the data link layer.

Figure 11: TX Transaction Layer

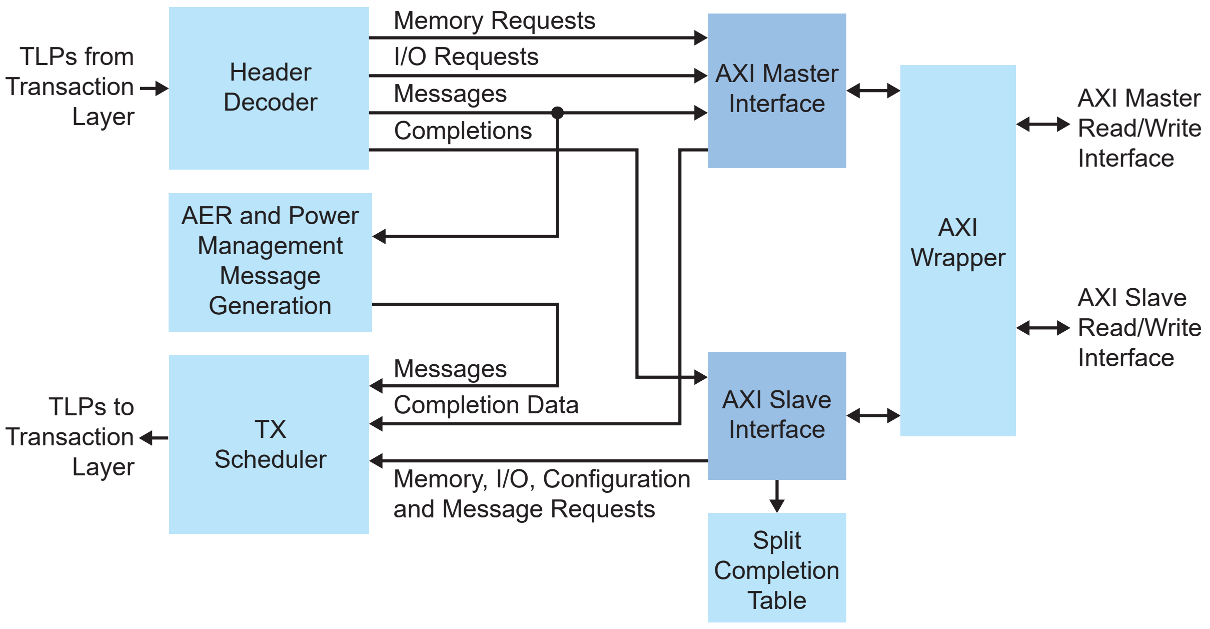

Interrupt Message Generator
Interrupt Interface
From Host PNP TX Interface
From Host SC TX Interface
32-, 64-, 128-, or 256-bit Data plus 128-bit Header
To Link Layer
TLP Formation ECRC Insertion
TX Scheduler
Ready for PNP
Ready for SC
Credit from TX Flow Control Module
Consumed Credit to TX Flow Control Module

AXI Application Layer

The application layer provides an AXI interface (that conforms to the AMBA AXI protocol) to the client application. This feature lets you use AXI signals to communicate with the PCIe Controller instead of having to use PCIe protocols. The AXI interface has:

- An AXI master read/write interface that connects to a memory controller or DMA engine. This interface is needed for all endpoints, and for root ports that allow endpoints to access their memory space. All AXI master interface signals in the core start with the prefix `TARGET_AXI` (see **AXI Master Interface Signals** on page 92).
- An AXI slave read/write interface that enables the endpoint or root port to generate memory transactions to the link partner as bus master. This interface is required for root ports. It is only required for endpoints that need bus master capability (see **AXI Slave Interface Signals** on page 97).
- An interface to signals received messages from the PCIe link.

Figure 12: AXI Modules


TLPs from Transaction Layer
Header Decoder
Memory Requests
I/O Requests
Messages
Completions
AXI Master Interface
AXI Wrapper
AXI Master Read/Write Interface
AER and Power Management Message Generation
AXI Slave Read/Write Interface
Messages
Completion Data
TLPs to Transaction Layer
TX Scheduler
AXI Slave Interface
Memory, I/O, Configuration and Message Requests
Split Completion Table

The PCIe Controller first decodes TLPs arriving from the transaction layer into memory requests, I/O requests, messages, and completion packets. Configuration requests are forwarded to the configuration module. The PCIe Controller sends decoded requests to the AXI master interface. The PCIe Controller performs write operations as indivisible operations (that is, address followed by data). Read operations allow delayed completions, thus allowing the address and data phases of multiple transactions to be interleaved.

With the AXI slave interface, the client can perform DMA reads and writes to the link partner's memory space, acting as a PCI master (this capability is required for root ports, and optional for endpoints). In root port, the the client application also uses the AXI slave interface to transmit I/O and configuration requests on the link. The AXI slave interface receives the client request and its associated parameters from the client application. If the transaction is a memory write or message operation, the AXI slave interface constructs a TLP containing the data to be written, and sends it to the transaction layer for delivery to the PCIe link. The core maintains no state for these posted transactions.

If the transaction is a non-posted transaction—memory read, I/O read, I/O write, configuration read, or configuration write—the AXI slave interface module forwards the request to the transaction layer and records the request parameters in the completion state memory (or split completion table). The PCIe Controller permits storage for the state of up to 256 pending transactions in the split completion table. When the corresponding completion TLP is received from the transaction layer, the AXI slave interface module matches it with the request and forwards the data to the application over the AXI slave interface. The AXI slave interface also handles completion timeouts by removing the transaction state from the completion state memory and signaling to the client using a dummy response across the AXI slave interface.

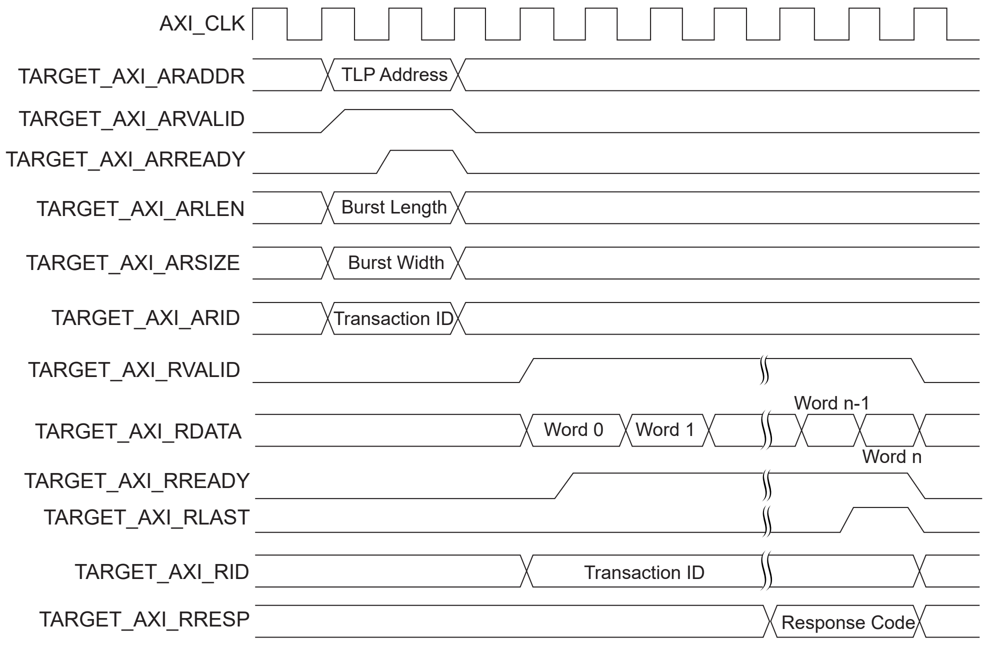
AXI Master Read Operation

An AXI read is a split transaction with independent address and data signals associated with the corresponding channels. That is, the client application can return the data for the read later. When the data for a read request becomes available, the client application transfers it using the AXI master read data channel signals. The client must send back all of the requested data in one single burst transaction.

The client must follow the slave protocol for the read address and data channel as described in the AXI Specification v1.0. The PCIe Controller follows the master protocol for the read address and data channel as described in the AXI specification v1.0.

Note: Refer to **AXI Master Interface Signals** on page 92 for the signal descriptions.

Figure 13: AXI Master Read Waveform



The read operation starts by placing the parameters associated with the read request on the AXI master's read address channel signals. The read address channel signals are ready the `TARGET_AXI_ARVALID` signal is asserted. The starting read address is in on `TARGET_AXI_ARADDR`. The client logic accepts the requests when it asserts `TARGET_AXI_ARREADY`. The PCIe Controller maintains the request and its associated descriptor until it receives acknowledgement from the client.

The client initiates a read data transfer by asserting `TARGET_AXI_RVALID` and placing the data aligned to the request address on `TARGET_AXI_RDATA`. The alignment requirement is only for the first cycle of the burst transfer. Subsequent data transfer should have valid data from the least significant byte. The client should flag the last data transfer cycle by asserting `TARGET_AXI_RLAST`. The PCIe Controller may pace the data transfer by asserting `TARGET_AXI_RREADY`; in this case the client should hold the each data cycle on the `TARGET_AXI_RDATA` bus until the PCIe Controller asserts the ready signal.

If the inbound TLP length is greater than the maximum AXI burst size, the PCIe Controller splits the PCIe transaction into multiple AXI read transactions with the same `TARGET_AXI_ARID`. This process ensures that read data coming back for the read requests are in order. The PCIe Controller issues multiple split completions back to the requester on receipt of every read data from the AXI interface.

Note: The AXI master interface does not support read data interleaving.

TLP (2 bytes, Aligned Address)

For TLP read of lengths (2^n up to 32 bytes), which are naturally aligned to the 256 bit AXI data bus, the PCIe Controller issues a single cycle burst request and controls `TARGET_AXI_ARSIZE` accordingly to read the exact amount of data from the AXI sub-system.

TLP (2^n and $2^{n>1}$, up to 32 Bytes, Unaligned Address)

A TLP read of lengths 2^n and $2^{n>1}$ up to 32 bytes, with an un-aligned address results in a read on the AXI sub-system that is greater than the number of bytes requested in the TLP. Due to the address alignment, the PCIe Controller nay read a few extra bytes from the aligned starting address lower than the unaligned address. Additionally, a few extra bytes are read beyond the intended bytes.

Note: The client should ensure that the extra reads from the sub-system does not cause undesirable side effects.

Table 10: TLP Read Lengths 2^n and $2^{n>1}$ up to 32 bytes, Unaligned Address

TLP Parameter		AXI Read Address Channel		
Addr[4:0] (Hex)	**Length in Bytes (Decimal)**	**Addr[4:0] (Hex)**	**ARSIZE (Binary)**	**ARLEN (Hex)**
0x5	2	0x5	010	0x0
0x3	2	0x3	011	0x0
0x1	4	0x1	011	0x0
0x7	4	0x7	100	0x0
0x2	8	0x2	100	0x0
0x4	16	0x4	101	0x0
0x12	16	0x12	101	0x1
0x1	32	0x1	101	0x1

Example: Unaligned Address Read 1

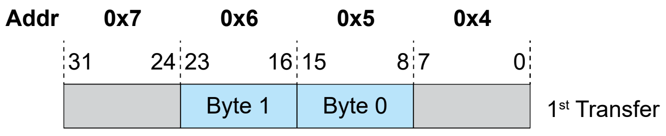
Condition:

Address = 0x5

ARSIZE = 2 (32 bit width)

ARLEN = 0

Byte Request = 2



Example: Unaligned Address Read 2

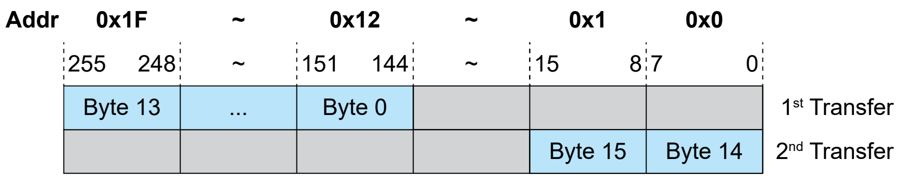
Condition:

Address = 0x12

ARSIZE = 5 (256 bit width)

ARLEN = 1

Byte Request = 16



TLP (Other, up to 32 Bytes)

Table 11: TLP Read of Other Lengths up to 32 Bytes

TLP Parameter		AXI Read Address Channel		
Addr[4:0] (Hex)	**Length in Bytes (Decimal)**	**Addr[4:0] (Hex)**	**ARSIZE (Binary)**	**ARLEN (Hex)**
0x0	5	0x0	011	0x0
0x1	3	0x1	010	0x0
0x2	3	0x2	011	0x0
0x3	5	0x3	011	0x0
0x5	7	0x5	100	0x0
0x4	18	0x4	101	0x0
0x1A	30	0x1A	101	0x1

TLP Read of Lengths > 32 Bytes

Table 12: TLP Read of Lengths > 32 Bytes

TLP Parameter		AXI Read Address Channel		
Addr[4:0] (Hex)	**Length in Bytes (Decimal)**	**Addr[4:0] (Hex)**	**ARSIZE (Binary)**	**ARLEN (Hex)**
0x1	34	0x1	101	0x1
0x5	50	0x5	101	0x1
0x1E	72	0x1E	101	0x3

Error Handling

During read data transfers, the client can indicate read data errors by asserting `TARGET_AXI_RRESP` to an error code (i.e., SLVERR or DECERR) on any valid data transfer cycle. The PCIe Controller sends a completer abort completion status back to the requester. The client must not do an early burst termination and transfer all the data cycles as indicated by the `TARGET_AXI_ARLEN` signal during the request cycle.

AXI ID Handling

The PCIe Controller can place outstanding read requests on the AXI master read channel. That is, the PCIe Controller might place subsequent read requests before the read data of a previous request has come back. At any point, the PCIe Controller has no more than 32 outstanding read requests on the AXI master read interface for link 0 and link 1. The PCIe Controller ensures that each outstanding read request has a unique `TARGET_AXI_ARID` so that the client can send back data for the outstanding read requests in any order, as per the AXI ordering rules. The

PCIe Controller keeps a table that maps inbound outstanding PCIe tags to the outstanding AXI master read requests IDs.

When the client returns read data for a particular ID, the PCIe Controller does an internal lookup to find the corresponding PCIe tag and forms an appropriate completion TLP to be sent on to the link.

Note: Although the PCIe Controller issues outstanding read requests with unique `TARGET_AXI_ARID`, it expects the client to return the entire data for one AXI outstanding request before sending data for a different outstanding request. In other words, the AXI master interface does not support read data interleaving.

Zero Length Reads

The AXI master interface signals a zero-length memory read transaction as a normal read request with a burst size `TARGET_AXI_ARLEN` of 0. The client must respond to a zero-length request in the same manner as a one-cycle read request by transferring a dummy one-cycle read data burst. The PCIe Controller then sends a completion TLP with a one-DWORD payload and byte count set to 1, as required by the PCIe specification.

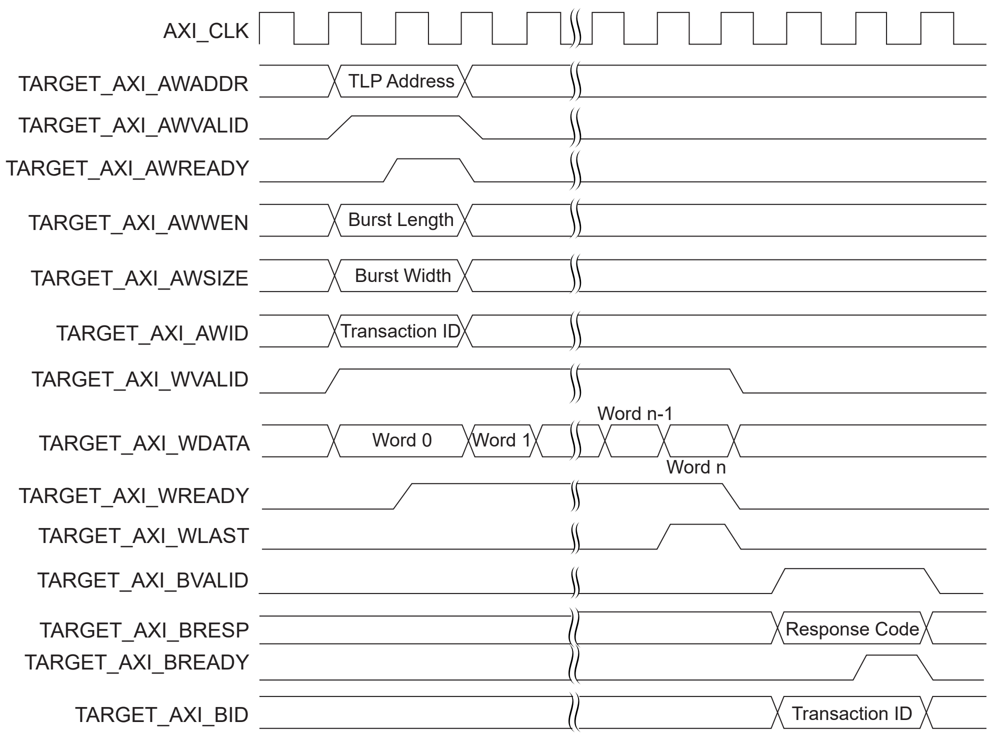
Non-Contiguous Reads

The AXI master interface does not distinguish between memory read requests received from the link with non-contiguous byte enables versus contiguous byte enables. The PCIe Controller presents a single DWORD, non-contiguous read on the AXI master interface with `TARGET_AXI_ARLEN = 3'b0` and `TARGET_AXI_ARSIZE = 3'b010`. A two DWORD non-contiguous read can be presented on the AXI master interface with `TARGET_AXI_ARLEN` equal to `3'b000` or `3'b001`, depending on the address alignment of the read request with respect to the 256-bit AXI data bus. In either case, the `TARGET_AXI_ARSIZE` is `3'b011`. The user must implement memory reads free of side effects so that an entire word can be read from memory without side effects even when only a part of the word is requested by the read transaction.

AXI Master Write Operation

An AXI write transaction is a split transaction with independent address and data signals associated with the corresponding channels. The client must follow the slave protocol for the write address and data channel as described in the AXI Specification v1.0. The PCIe Controller follows the master protocol for the write address and data channel as described in the AXI specification v1.0.

Figure 14: AXI Master Write Waveform



Example: Unaligned Address Write 1

Condition:

Address = 0x5

AWSIZE = 2 (32 bit width)

AWLEN = 0

Byte Send = 2

Addr	**0x7**	**0x6**	**0x5**	**0x4**	
	31 24	23 16	15 8	7 0	
		Byte 1	Byte 0		1st Transfer

Example: Unaligned Address Write 2

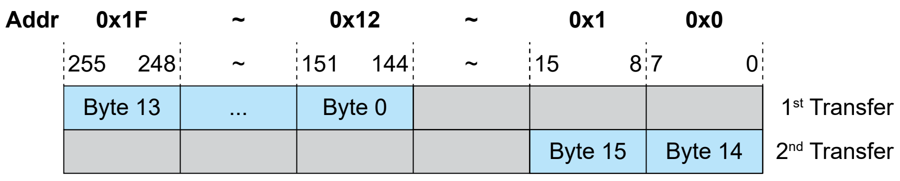
Condition:

Address = 0x12

AWSIZE = 5 (256 bit width)

AWLEN = 1

Byte Send = 16



The write operation starts by placing the write request parameters on the AXI master write address channel signals. The write address channel signals are qualified by asserting the `TARGET_AXI_AWVALID` signal. The starting write address is placed on `TARGET_AXI_AWADDR`. The client accepts requests when it asserts `TARGET_AXI_AWREADY` input to the PCIe Controller. The PCIe Controller maintains the request and its associated descriptor until it receives acknowledgement from the client.

The PCIe Controller begins to transfer the data words by placing them on the AXI master write data channel signals and asserting `TARGET_AXI_WVALID`. The client can pace the data transfer by controlling the `TARGET_AXI_WREADY` input to the PCIe Controller. The PCIe Controller keeps each data word on the `TARGET_AXI_WDATA` data bus until it samples the ready input as high on a positive edge of the clock. The PCIe Controller indicates the last data transfer cycle by asserting the `TARGET_AXI_WLAST` signal. It does not perform an early burst termination and transfers the entire burst data as indicated in the `TARGET_AXI_AWLEN` signal during the request cycle.

The `TARGET_AXI_WSTRB[31:0]` outputs to indicate the valid bytes in the data transfer cycle on the first and last cycles of the data transfer. The transfer may start and finish at any byte position in the data path. For writes of a single DWord, the byte valids may be non-contiguous, as allowed by the PCIe specification. Likewise, for two-DWORD writes, the byte valids may be non-contiguous if the starting address is aligned on an even DWORD boundary.

If the inbound TLP length is greater than the maximum AXI burst size, the PCIe Controller splits the PCIe transaction into multiple AXI write transaction with the same `TARGET_AXI_AWID`. This process ensures that write data read requests are committed to the client in order.

Poison Bit Forwarding to AXI

If poisoned bit forwarding is enabled in the AXI features control register, the poisoned TLP is flagged in the AWUSER bit [43]. Additionally, the PCIe Controller adds a message interface bit to indicate when a poisoned TLP is forwarded to the message interface.

The `MASTER_AXI_BUSER` is added to send a UR completion for poisoned non-posted write responses. You need to set this bit along with the write response if the non-posted write request had a poisoned TLP bit set in bit 43 of `TARGET_AXI_AWUSER`.

Error Handling

The client can indicate an error response for a write request on the AXI master write response channel signals by asserting `TARGET_AXI_BVALID` and indicating an error code on the `TARGET_AXI_BRESP` signal. The PCIe Controller ignores the response type (i.e., good or bad) for a posted write request. For a non-posted write request, the PCIe Controller sends back a completer abort completion to the requester; otherwise, on receipt of a good response for a non-posted write request, PCIe Controller sends back a good completion to the requester.

AXI ID Management

All inbound posted write TLPs are issued with the same `TARGET_AXI_AWID` so that they complete in order on the client memory subsystem. Each inbound non-posted write TLP is issued a unique TARGET_AXI_AWID so that the write responses can come back in any order. The PCIe Controller internally manages the mapping between an incoming PCIe tag of a TLP and the corresponding TARGET_AXI_AWID issued on the AXI master interface for a non-posted TLP. The PCIe Controller returns a completion TLP on receipt of a write response; it maps the incoming TARGET_AXI_BID to the corresponding PCIe tag of the TLP and sends back the completion with the appropriate tag information. The PCIe Controller cannot have more than 32 outstanding write transactions for link 0 and link 1 at any time.

Note: The PCIe Controller does not support write data interleaving. That is, the write interleaving depth is 1.

Zero-Length Writes

The PCIe Controller ignores zero-length memory write transactions received from the inbound PCIe link. The PCIe specification does not require completions for posted writes and the AXI specification does not support zero-length write requests.

Non-Contiguous Writes

The PCIe specification allows memory writes with non-contiguous byte enables for single-DWord writes, and for two-DWORD writes when the address is aligned on an 8-byte boundary. For these write transactions, the AXI master interface sets the byte valid bits on the TARGET_AXI_WSTRB signal based on the valid bytes indicated in the header of the request TLP. The client must ensure that the individual bytes on the TARGET_AXI_WDATA bus are only written to memory when the corresponding byte valid is asserted.

Ordering Between Posted and Non-Posted Writes

The PCIe Controller ensures strict ordering between posted and non-posted writes on the AXI master write interface. All posted write requests are issued with the same TARGET_AXI_AWID so that they complete in order in the AXI subsystem.

When a non-posted write follows a posted write, the PCIe Controller ensures that all outstanding posted writes complete (i.e., TARGET_AXI_BVALID is received for all outstanding write transactions) before issuing the non-posted write. This process ensures that non-posted transactions are not processed before posted transactions.

End-to-End Data Protection

The PCIe protocol helps to ensure the integrity of data transferred via serial link. The Data Link Layer (DLL) provides this assurance by running a Cyclic Redundancy Check (CRC) on the integrity of Transaction Layer Packets (TLPs) and Data Link Layer Packets (DLLPs). When the DLL finds an instance of data corruption in a DLLP, the DLL initiates a retry mechanism until the data passes the CRC check. However, TLPs traveling outside the DLL (from the Transaction Layer toward the Application Layer) do not benefit from these PCIe data protection protocols. Therefore, our PCIe controller enforces additional layer of data protection in the form of a byte-wide parity check, ensuring end-to-end TLP data integrity.

Depending upon your AXI bus, customers may need to implement byte-wide odd parity in a cycle-by-cycle fashion. You generate this parity bus using the following RTL:

```
genvar i;
generate
    for (i = 0; i< AXI_PCIE_SIGNAL_PAR_WIDTH; i=i+1)
        assign axi_pcie_signal_par[i] = ~(^axi_pcie_signal[i*8 +: 8]); // odd parity
    end
endgenerate
```

- On the inbound path, the RTL generates parity one cycle ahead of the DLL CRC check to ensure a one-cycle overlap between the two protection methods. The controller transmits this parity data in tandem with the original data across the pipeline to the application/client logic at the controller boundary.
    - At the pcie_target_AXI interface, the controller drives parity for all bytes of pcie_target_AXI_WDATA on pcie_target_AXI_WUSER output regardless of pcie_target_AXI_WSTRB.

    - At the pcie_master_AXI interface, the controller drives parity for all bytes of pcie_master_AXI_RDATA on the pcie_master_AXI_RUSER output.
- On the outbound path, the application/client logic provides cycle-by-cycle parity at the controller's interface boundary (this can be either the AXI or HAL boundary, depending upon the controller configuration). The controller maintains this parity across the datapath pipeline up to the link layer. Upon receipt, the link layer generates the CRC and, one cycle later, the controller checks for parity with the data, thus ensuring a one-cycle overlap between the two protection methods.
    - At the pcie_master_AXI interface, the client must drive parity for all bytes of pcie_master_AXI_WDATA on pcie_master_AXI_WUSER, regardless of pcie_master_AXI_WSTRB.
    - At the pcie_target_AXI interface, the client must drive parity for all bytes of pcie_target_AXI_RDATA on the pcie_target_AXI_RUSER input.

Inbound Message Interface

The PCIe Controller includes a dedicated interface for inbound messages. The inbound message interface is suitable for driving a message gathering FIFO (the PCIe Controller does not include this FIFO). You can place message-type decode logic to filter messages into different FIFOs, take specific action, or discard redundant messages, depending on the application needs.

The interface is synchronous to `AXI_CLK` and does not support back pressuring. It includes valid, start, and end strobes, as well as strobes to identify vendor-defined header and data. The message interface width is the same as the AXI master port data bus.

The message header always occupies 64 bits with an additional 64 bits for header bits [127:64] of a vendor-defined message.

Table 13: Message Header Bit Allocation

Bits	Bit Description	Header Stripe
255:128	Unused	0
127:64	Vendor Defined Message Header Page Request Messages: • [127:120] Page Address [63:56] • [119:112] Page Address [55:48] • [111:104] Page Address [47:40] • [103:96] Page Address [39:32] • [95:88] Page Address [31:24] • [87:80] Page Address [23:16] • [79:76] Page Address [15:12] • [75:67] Page Request Group Index • [66] L bit (last request in PRG) • [65] W bit (write access requested) • [64] R bit (read access requested) For Page Request Group Response Messages: • [127:112] Destination ID • [111:108] Response code • 0000b: Success • 0001b: Invalid request • 1110b to 0010b: Unused • 1111b: Response failure • [104:96] PRG Index Stop Marker Messages: • [71:67] Marker type (expected value 5'b00000) • [66] L bit (expected value 1'b1) • [65] W bit • [64] R bit Invalidation Request Messages: • [127:112] Destination ID Invalidate Completion Messages: • [127:112] Device ID • [98:96] CC value • [95:64] ITAG Vector For OBFF messages, [123:120] carries the OBFF message code. Other bits are unused. For LTR messages: • [127:120] Snoop latency bits [7:0] • [119:112] Snoop latency bits [15:8] • [111:104] No-snoop latency bits [7:0] • [103:96] No-snoop latency bits [15:8]	0
63:60	Unused	0
59:52	PCIe tag for normal messages For invalidation request messages: [56:52] -ITAG	0
51:36	If bit 32 (TPH present) is set to 1, this field has the steering tag. If bit 32 is cleared this field has the PCIe tag for the vendor-defined messages.	0
35:34	Processing hint	0
33	1: 16-bit steering tag 0: 8-bit steering tag	0
32	TPH Present	0
31:24	Message Code	0
23:8	Requester ID	0
6:4	Routing	0
3:1	Attributes	0

Bits	Bit Description	Header Stripe
0	0: Normal vendor defined message	0

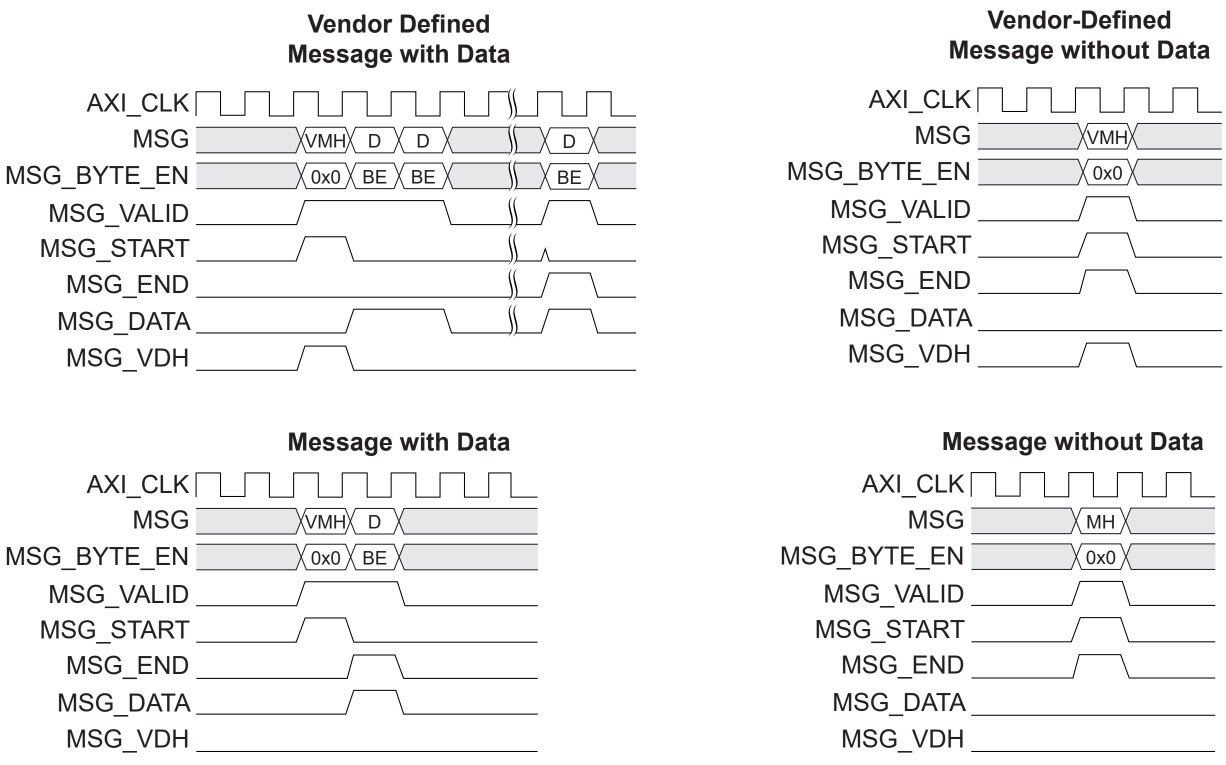
Message Interface Signals

The message interface has the following functionality:

- The MSG_VALID strobe signal indicates when MSG has valid data or a valid header.
- The MsgD data always starts at byte 0 of the new cycle (LSB).
- Byte enables (MSG_BYTE_EN) are valid when the interface is outputting message data (the MSG_DATA strobe signal is asserted).
- MSG_BYTE_EN is driven low when the interface is outputting a message header or vendor defined header.
- When transferring data, MSG_BYTE_EN does not have any 0s in between 1s. Therefore, the data will be contiguous without any byte valid low.
- The PCIe Controller may de-assert the MSG_VALID signal during the message transfer (in between MSG_START and MSG_END).
- The MSG output is only valid when MSG_VALID is 1.
- The MsgD payload size is limited by the PCIe Controller's MAX_PAYLOAD_SIZE value.
- All header bits are in a single stripe.

Note: Refer to **Message Interface** for message signals description.

Figure 15: Message Interface Waveforms

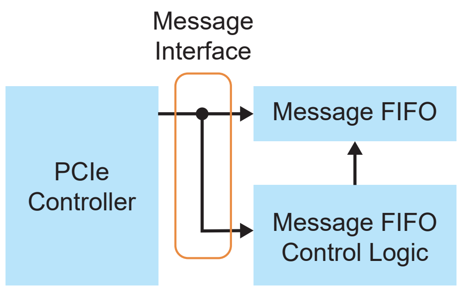


BE: Byte enable.
D: Data.
MH: Only normal message bits [63:0] are valid.
VMH: Only vendor-defined message header bits [127:0] are valid.

Message Interface FIFO Buffer

The FIFO depth can be selected to anything more than 32 with a programmable threshold for handling the message overflow (with an interrupt to the local processor when overflow is detected). You can decide which message types you require. The PCIe Controller's AXI interface contains logic to decode messages for the assertion and de-assertion of legacy interrupts.

Figure 16: Message Interface FIFO



These messages are output on the message interface the same as any other message, with the addition of assertion or deassertion of the relevant `INTA_OUT`, `INTB_OUT`, `INTC_OUT`, or `INTD_OUT` signal. The change on the relevant `INTx_OUT` signal occurs during the same clock cycle that the message is output on the message interface. The `INTx_OUT` signal levels are not changed if an assert message is received for an interrupt that is already asserted, or if a deassert message is received for an interrupt that is already deasserted. The message is output on the message interface as usual. All four interrupts are deasserted when the AXI reset is asserted and when the `LINK_DOWN_RESET` signal is asserted. In-bound messages from the link that are directed to the message interface (intended for the message FIFO) do not appear in the main AXI master interface.

Message Interface Codes

Table 14: Inbound Message Codes

Message Code	Routing	Type	Description	Mode	Integration Comment	Number of DW
0000_0000	011	Msg	Unlock	DM	-	2
0000_0001	010	MsgD	Invalidate Request Message	EP	The PCIe Controller takes no action and forwards the message to the message interface. Endpoint client logic must invalidate the corresponding address translation table entries upon receiving this message.	4
0000_0010	010	Msg	Invalidate Completion Message	RP	The PCIe Controller takes no action and forwards the message to the message interface. Indicates completion of invalidation operation.	4
0000_0100	000	Msg	Page Request Message	RP	The PCIe Controller takes no action and forwards the message to the message interface. Client must take appropriate action.	4
0000_0101	010	Msg	PRG Response Message	EP	The PCIe Controller takes no action and forwards the message to the message interface. Client must take appropriate action.	4
0001_0000	100	Msg	Latency Tolerance Reporting Message	RP	Internally captured by PCIe Controller and compared with the L1_pm_substates_control1_reg_LTR_ L1_2_threshold to determine L1.2 substate entry. Client can ignore this message.	4
0001_0010	100	Msg	OBFF	EP	If OBFF_ENABLE[1:0] == 01 or 10, the PCIe Controller forwards the OBFF message to the message interface. Otherwise it reports it as UR. Client can optionally process the OBFF message to determine the CPU activity.	4
0001_0100	100	Msg	PM_Active_ State_Nak	EP	Internally processed during ASPM L1 Entry negotiation. Also forwarded to message interface. Client can ignore this message.	2

Message Code	Routing	Type	Description	Mode	Integration Comment	Number of DW
0001_1000	000	Msg	PM_PME	RP	The PCIe Controller takes no action and forwards the message to the message interface. Client must process this message per PCIe power management specifications. For example, the root port can issue a CfgWr to change the requesting function power state to D0.	2
0001_0001	011	Msg	PME_Turn_Off	EP	If all function power states are in non-D0 state and if PME Turnoff Ack Delay > 0x0000, the PCIe Controller automatically transmits a PME_TO_Ack message after the PME Turnoff Ack Delay time. In this case, the client logic should not send PME_TO_ACK. Otherwise, the client logic must respond with the PME_TO_Ack message.	2
0001_0011	101	Msg	PME_To_Ack	RP	The PCIe Controller takes no action and forwards the message to the message interface. Client can choose to turn off the power after receiving this message.	2
0010_0000	100	Msg	Assert_INTA	RP	The PCIe Controller asserts INTA_OUT upon receiving this message. Client can ignore this message and only use INTA_OUT.	2
0010_0001	100	Msg	Assert_INTB	RP	The PCIe Controller asserts INTB_OUT upon receiving this message. Client can ignore this message and only use INTB_OUT.	2
0010_0010	100	Msg	Assert_INTC	RP	The PCIe Controller asserts INTC_OUT upon receiving this message. Client can ignore this message and only use INTC_OUT.	2
0010_0011	100	Msg	Assert_INTD	RP	The PCIe Controller asserts INTD_OUT upon receiving this message. Client can ignore this message and only use INTD_OUT.	2
0010_0100	100	Msg	Deassert_INTA	RP	The PCIe Controller de-asserts INTA_OUT upon receiving this message. Client can ignore this message and only use INTA_OUT.	2
0010_0101	100	Msg	Deassert_INTB	RP	The PCIe Controller de-asserts INTB_OUT upon receiving this message. Client can ignore this message and only use INTB_OUT.	2
0010_0110	100	Msg	Deassert_INTC	RP	The PCIe Controller de-asserts INTC_OUT upon receiving this message. Client can ignore this message and only use INTC_OUT.	2
0010_0111	100	Msg	Deassert_INTD	RP	The PCIe Controller de-asserts INTD_OUT upon receiving this message. Client can ignore this message and only use INTD_OUT.	2
0011_0000	000	Msg	ERR_CORR	RP	The PCIe Controller asserts CORRECTABLE_ERROR_DETECTED _OUT for one clock cycle when it receives a ERR_CORR message. Client can ignore this message and only use CORRECTABLE_ERROR_DETECTED_OUT.	2
0011_0001	000	Msg	ERR_NONFATAL	RP	The PCIe Controller asserts NON_FATAL_ERROR_DETECTED_OUT for one clock cycle when it receives a ERR_CORR message. Client can ignore this message and only use NON_FATAL_ERROR_DETECTED_OUT.	2
0011_0001	000	Msg	ERR_FATAL	RP	The PCIe Controller asserts FATAL_ERROR_DETECTED_OUT for one clock cycle when it receives a ERR_CORR message. Client can ignore this message and only use FATAL_ERROR_DETECTED_OUT.	2
0100_0000	100	Msg	Ignored	NA	–	2
0100_0000	100	Msg	Ignored	NA	–	–

Message Code	Routing	Type	Description	Mode	Integration Comment	Number of DW
0100_0001	100	Msg	Ignored	NA	–	–
0100_0011	100	Msg	Ignored	NA	–	–
0100_0100	100	Msg	Ignored	NA	–	–
0100_0101	100	Msg	Ignored	NA	–	–
0100_0111	100	Msg	Ignored	NA	–	–
0101_0000	100	Msg	Set_Slot_ Power_Limit	EP	The PCIe Controller stores the data from the received message in the Captured Slot Power Limit Scale and Value fields in Device Capabilities Register. Client can ignore this message.	2
0111_1110	000, 010, 011, 100	Msg, MsgD	VD Msg Type0	DM	The PCIe Controller takes no action and forwards the message to the message interface. Processing of Vendor Defined Message is implementation specific.	4
0111_1111	000, 010, 011, 100	Msg, MsgD	VD Msg Type1	DM	The PCIe Controller takes no action and forwards the message to the message interface. Processing of a vendor-defined message is implementation specific.	4

Ordering Between AXI Master Write and Read Channels

The PCIe Controller issues posted writes on the AXI master write channel, non-posted writes on the AXI master write channel, and non-posted reads on the AXI master read channel. The PCIe Controller enforces PCIe ordering between posted and non-posted reads and writes on the AXI master interface. Posted writes are always sent before of non-posted reads or writes.

Before issuing non-posted transactions on the AXI master write or AXI master read channels, the PCIe Controller ensures that previously issued posted writes have completed on the client by waiting for all `TARGET_AXI_BVALID` responses to come back.

On the AXI master interface, the PCIe ordering rules are followed as shown in the following tables. The columns represent a first issued transaction and the rows represent a subsequently issued transaction. The table entry indicates the ordering relationship between the two transactions. The table entries are defined as follows:

- *Yes*—The second transaction (row) must be allowed to pass the first (column) to avoid deadlock. When blocking occurs, the second transaction must pass the first transaction. Fairness must be comprehended to prevent starvation.
- *Y/N*—There are no requirements. The second transaction may optionally pass the first transaction or be blocked by it.
- *No*—The second transaction must not be allowed to pass the transaction to support the producer–consumer strong ordering model.

Table 15: Inbound Ordering (Endpoint Mode)

Row Pass Column?		Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
			Read Request (Col 3)	With Data (Col 4)	
Posted Request (Row A)		No	Yes	Yes *(1)*	Yes
Non-Posted Request	Read Request (Row B)	No	No	Yes	Yes
	NPR with Data (Row C)	No	Yes	No	Yes
Completion (Row D)		A: No *(2)* B: Y/N *(3)*	Yes	Yes	No *(4)*

Table 16: Inbound Ordering (Root Port Mode)

Row Pass Column?		Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
			Read Request (Col 3)	With Data (Col 4)	
Posted Request (Row A)		No	Yes	N/A *(1)*	Yes
Non-Posted Request	Read Request (Row B)	No	No	N/A	Yes
	NPR with Data (Row C)	N/A	N/A	N/A	N/A
Completion (Row D)		A: No *(2)* B: Y/N *(3)*	Yes	N/A	No *(4)*

Notes:

1. Posted reads and writes always pass non-posted reads and writes in the transaction layer. However, a non-posted write can stall on the AXI write channel for a long time if the client cannot service the non-posted write, which in turn blocks a posted write coming in later from the link. To address this issue, the client can use the `TARGET_NON_POSTED_REJ` input signal to indicate that the PCIe Controller should not service non-posteds from the transaction layer's non-posted FIFO. This action allows posted transactions to go through the AXI write channel when the client cannot service non-posted read and writes.
2. A completion must not pass a posted request unless Row D Column 2 B applies.
3. An I/O or configuration write completion can pass a posted request. A completion with a relaxed ordering set can pass a posted request. A completion with an ID-based ordering set can pass a posted request if the completer ID of the completion is different from the requester ID of the posted request.
4. Although completions do not pass each other at the transaction layer, completions are reordered back to the AXI bus because AXI bus reads with same ID have to come in order. However, there is no relaxed ordering or ID-based ordering effect.

Inbound PCIe to AXI Address Translation (Root Port)

The PCIe Controller performs root port inbound PCIe to AXI address translation on memory and I/O TLPs. The PCIe Controller chooses which address translation registers to use for translation based on the BAR match of the incoming TLP. There are two BARs in root port mode, so the registers are BAR0 and BAR1. Additionally, the PCIe Controller uses the BAR7 register for cases in which there are no matches. The PCIe Controller sends any address that does not match the root port BARs as a BAR7 TLP.

Each BAR register has two 32-bit registers, `addr0` and `addr1`. The address translation logic takes the upper bits from the root port inbound PCIe to AXI address translation registers and takes the lower bits from the inbound PCIe address to form the AXI address. The `addr0[5:0]`

+ 1 number of lower bits are passed from the inbound PCIe address to AXI address. That is, the number of bits taken from inbound PCIe address is given by the `addr0[5:0]` + 1 value.

Figure 17: Root Port Inbound PCIe to AXI Address Translation

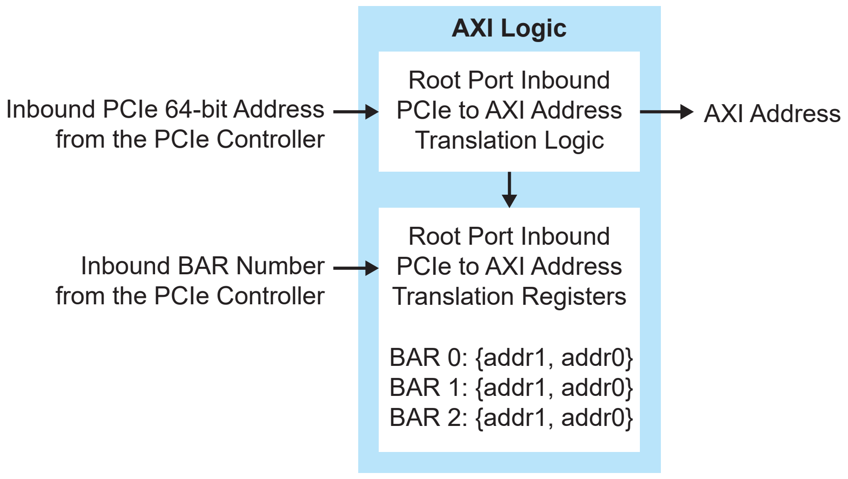

AXI Logic
Inbound PCIe 64-bit Address from the PCIe Controller
Root Port Inbound PCIe to AXI Address Translation Logic
AXI Address
Inbound BAR Number from the PCIe Controller
Root Port Inbound PCIe to AXI Address Translation Registers
BAR 0: {addr1, addr0}
BAR 1: {addr1, addr0}
BAR 2: {addr1, addr0}

Table 17: Root Port Inbound PCIe to AXI Address Translation Registers for 1 BAR

Where *BAR* is bar0, bar1 or bar7

Register Name	Bits	Description	Default Value
ib_rp_[*BAR*]_addr1	31:0	Upper [63:32] bits of the AXI address.	32'd0
ib_rp_[*BAR*]_addr0	31:8	Lower [31:8] bits of the AXI address.	24'd0
	7:6	Reserved	2'd0
	5:0	Number of address bits passed from PCIe to AXI. The PCIe Controller passes the programmed value + 1 bits from PCIe to AXI. The minimum value to be programmed into this field is 7 because the lower eight bits of the base address programmed in these registers (AXI) are replaced by zeros by the address translation logic.	6'd0

Inbound PCIe to AXI Address Translation (Endpoint)

The PCIe Controller performs end point inbound PCIe to AXI address translation on memory and I/O TLPs. The PCIe Controller chooses which address translation registers to use for translation based on the BAR match of the incoming TLP. There are seven BARs per function in endpoint mode; therefore, there are seven sets of registers per function with each BAR having two 32-bit registers (`addr0` and `addr1`). The address translation logic takes the upper bits from the endpoint inbound PCIe to AXI address translation registers and takes the lower bits from the inbound PCIe address to form the AXI address. The inbound BAR aperture determines the number of bits to pass from the inbound PCIe address to AXI.

Figure 18: Endpoint Inbound PCIe to AXI Address Translation

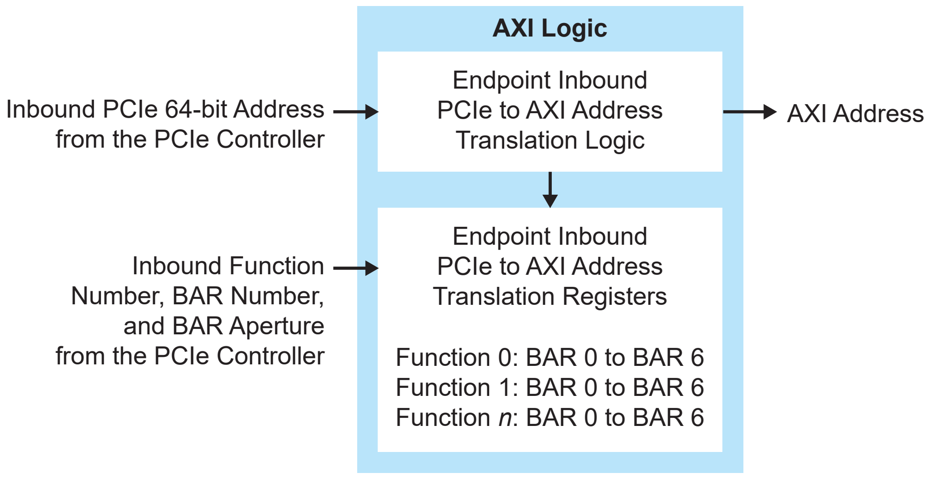

AXI Logic
Inbound PCIe 64-bit Address from the PCIe Controller
Endpoint Inbound PCIe to AXI Address Translation Logic
AXI Address
Inbound Function Number, BAR Number, and BAR Aperture from the PCIe Controller
Endpoint Inbound PCIe to AXI Address Translation Registers
Function 0: BAR 0 to BAR 6
Function 1: BAR 0 to BAR 6
Function n: BAR 0 to BAR 6

Table 18: Endpoint Inbound PCIe to AXI Address Translation Registers for 1 BAR

Where *BAR* is bar0, bar1, bar2, … bar7 and *PF* is pf0, pf1, pf2, ... pf21

Register Name	Bits	Allocation	Default Value
[*PF*]_ib_ep_[*BAR*]_addr1	31:0	Upper [63:32] bits of the AXI address.	32'd0
[*PF*]_ib_ep_[*BAR*]_addr0	31:0	Lower [31:8] bits of the AXI address.	32'd0

AXI Slave Interface

The AXI slave interface enables a client endpoint to initiate PCI transactions as a bus master across the PCIe link to the host memory. For root ports, this interface initiates I/O and configuration requests. For endpoints, this interface must be connected to client logic only when the client has bus master capability. Endpoints and root ports can also use the slave interface to send messages on the PCIe link. The transactions on this bus are similar to those on the AXI master bus, except that the roles of the PCIe Controller and the client are reversed.

The client must check the following conditions before making a request on the AXI slave interface:

- Only root ports can initiate I/O or configuration requests.
- An endpoint can initiate a memory read or write request only when the Bus Master Enable bit of the PCI Command Register associated with the requesting function is set. These bits are accessible on the PCIe Controller's `FUNCTION_STATUS` output (see **Status and Error Indicator Signals** on page 102).
- An endpoint can only send requests when the originating function's power state is D0-Active. The function power state is available on the `FUNCTION_POWER_STATE` output (see **Status and Error Indicator Signals** on page 102).
- The originating function is not currently processing a function-level reset (FLR).

Note: All AXI slave interface signals have the prefix `MASTER_AXI` (see **AXI Slave Interface Signals** on page 97).

Unsupported Request Handling During Enumeration (Rootport)

If an unsupported request (UR) or configuration request retry status (CRS) is received for a configuration request, the PCIe Controller does not assert the `SLVERR` if the AXI features control register's `SLVERRCTRL` bit is set to 1. The returned data is:

- UR—32'hFFFF_FFFF
- CRS—32'hFFFF_0001

AXI Slave Ordering

If non-posted writes and posted writes are issued to the AXI slave with the same ID, posted write requests are only sent to the link when the previous non-posted write completions are received from the link. This process follows the AXI ordering rules for same ID requests.

Completion Error Handling

When the PCIe Controller receives a completion TLP from the link, it matches the TLP against the outstanding requests in the split completion table to determine the corresponding request and compares the fields in its header against the expected values to detect any error conditions. The PCIe Controller then signals the error conditions on `MASTER_AXI_RRESP` and sets `SLVERR` (2'b10) or `DECERR`(2'b11). The PCIe Controller asserts this signal as well as the `MASTER_AXI_RRESP` and `MASTER_AXI_RVALID` signals. When the client receives a read response with slave error as the response code, it should discard the data sent by the PCIe Controller and it should either discard or retry the corresponding request.

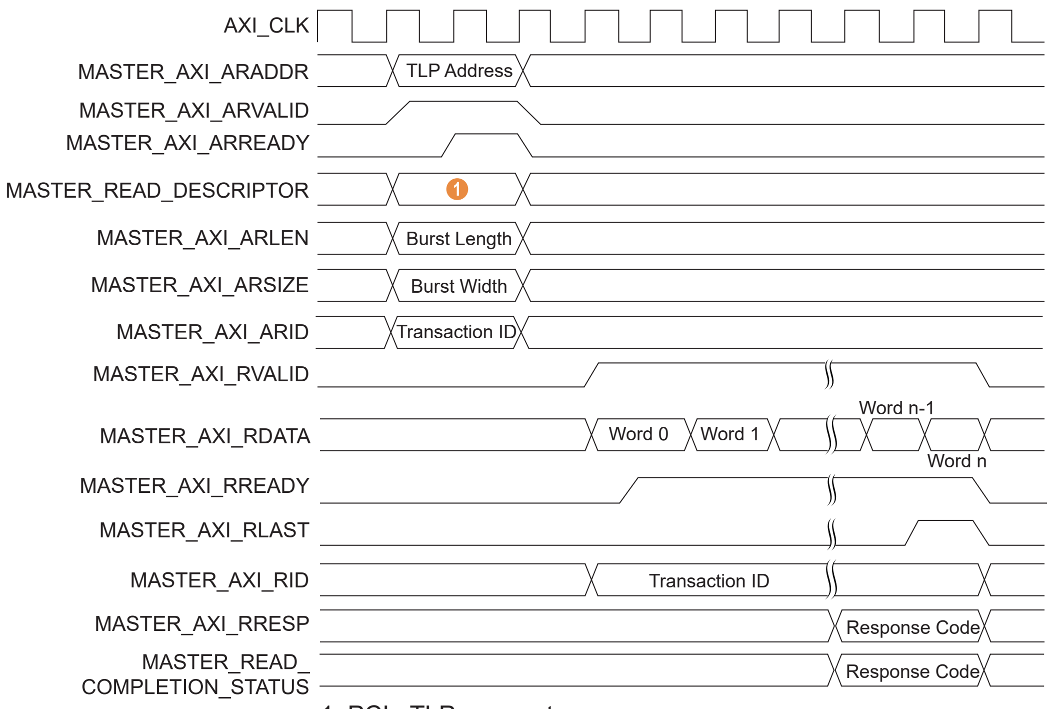
The PCIe error conditions that can lead to a slave error are:

- The completion TLP received from the link was poisoned.
- Request terminated by a completion TLP with UR, CA, or CRS status.
- Read request terminated by a completion TLP with incorrect byte count.
- The current completion being delivered has the same tag as an outstanding request, but its requester ID, TC, or Attr fields did not match the parameters of the outstanding request.
- Error in starting address.
- Request terminated by a completion timeout, or by a function-level reset (FLR) targeting the function that generated the request.

AXI Slave Read Operation

An AXI read is a split transaction with independent address and data on the corresponding channels. The client must follow the master protocol for the read address and data channel as described in the AXI Specification v1.0. The PCIe Controller follows the slave protocol for the read address and data channel as described in the AXI specification v1.0.

Figure 19: AXI Slave Read Interface Waveform



1: PCIe TLP parameters.

When `MASTER_AXI_ARLEN` is not zero, `MASTER_AXI_ARSIZE` must be the maximum value (5).

The client starts a memory read operation by placing the read request parameters on the AXI slave read address channel and asserting the `MASTER_AXI_ARVALID` signal. The PCIe Controller responds to the request by asserting the `MASTER_AXI_ARREADY` signal for one clock cycle. The PCIe Controller might not be able to accept the request if it does not have adequate credit to transmit the request TLP on the link or if the split completion table is full.

When the data for a read request becomes available, the PCIe Controller transfers the data on the AXI slave read data channel. The PCIe Controller begins the transfer by placing the data word on the `MASTER_AXI_RDATA` bus and asserting the `MASTER_AXI_RVALID` signal. The completion is delivered as a single burst for each read request. In the first data transfer cycle, the PCIe Controller returns data on `MASTER_AXI_RDATA` bus aligned to the request address. It indicates the last data transfer cycle by asserting the `MASTER_AXI_RLAST` signal. The client can pace the data transfer by controlling the `MASTER_AXI_RREADY` input to the PCIe Controller. The PCIe Controller keeps each data word on the `MASTER_AXI_RDATA` bus until it samples the ready input high on a positive edge of the clock. The PCIe Controller will not terminate a burst for a read request on the AXI slave interface. It always satisfies the complete read request as indicated by the `MASTER_AXI_ARLEN` signal.

In root port, the AXI Slave interface initiates configuration and I/O read requests, which function in the same way as memory reads. These requests are distinguished from memory requests by the transaction-type field in the master read descriptor bus. The data returned in response to these requests is always four bytes long and is delivered aligned to the request address.

Tag Management for Non-Posted Transactions

The AXI slave maintains the state of all pending, client-initiated non-posted transactions (e.g., memory reads, I/O reads and writes, configuration reads and writes) so that the completions returned by the targets can be matched to the corresponding requests. The AXI slave has a split completion table that stores the state of each outstanding transaction; the table has a capacity of four non-posted transactions. The returning completions are matched with the pending requests using an 8-bit tag. The PCIe Controller allocates the tag for each non-posted request initiated from the AXI slave. The PCIe Controller maintains a list of free tags and assigns one to each request when the client initiates a non-posted transaction. The PCIe Controller checks whether the split completion table is full, and only accepts an AXI request from the client if the number of outstanding non-posted requests is less than four.

Error Handling

The PCIe Controller drives the `MASTER_AXI_RRESP` signal when it sends read data out for a request with `MASTER_AXI_RVALID`. It can signal an error response to the client any time during the data transfer cycles by putting an error response of `SLVERR` or `DECERR` on the `MASTER_AXI_RRESP` output.

AXI ID Management

Read requests are split transactions; that is, the client may make additional read and write requests while the completion for a read request is pending. The client can issue each outstanding read request with the same `MASTER_AXI_ARID` or unique ones. The PCIe Controller can receive a maximum of 256 outstanding read requests. It internally maps the `MASTER_AXI_ARID` to an internally generated PCIe tag. The PCIe Controller looks up this mapping to translate an incoming completion to a corresponding `MASTER_AXI_RID` value sent with the read data channel.

Note: The read interleaving depth is one; that is, the PCIe Controller transfers complete data for a particular read request before sending data out for another read request.

Completion Data Ordering

AXI specifications mandate that if there are outstanding read requests with same `MASTER_AXI_ARID`, read data should be returned by the AXI slave interface in order. However, because each of these outstanding read requests are assigned unique tags on the PCIe side, the PCIe ordering rules permit the read completions to come back in any order. The PCIe Controller re-orders the out-of-order completions and issues them out on the AXI slave read data channel in the correct order. If the client issues read requests with unique `MASTER_AXI_ARID`, the AXI ordering rules permit the read data to come back out of order and the PCIe Controller issues the read data in the order it comes back from the PCIe link.

Error and Decode Errors

The following table lists the different status codes and their causes. The `decode_err` indicates a usage error and points to incorrect user programming of the AXI outbound address. This error is fatal; the behavior of the PCIe Controller after this error is not deterministic.

The user application should:

- Fix the programming error causing the decode error.
- Reset the PCIe Controller to recover from this error.

Table 19: AXI Slave Error and Decode Error Cases

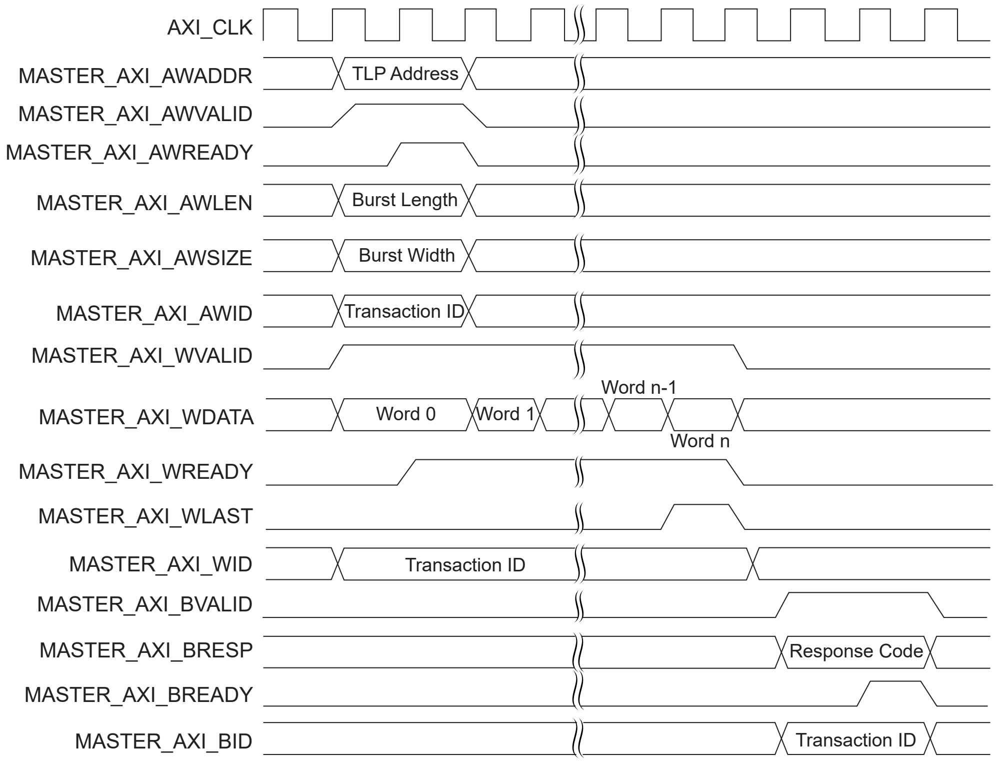
Cases	Indication Register	MASTER_AXI_ RUSER_STATUS	Description	AXI Response
Normal completion.	N/A	5'b000	The completion returned by the link partner has no errors.	OK
The completion TLP received from the link was poisoned.	Poisoned TLP status in AER Uncorrectable Status Register Detected parity error bit in Command Status Register	5'b001	The completion received from the link was poisoned.	SLVERR
Request terminated by a completion TLP with UR, CA, or CRS status.	Received target abort status bit in Command and Status Register only for CA	5'b010	Request terminated by a completion TLP with UR, CA, or CRS status by the link partner.	SLVERR
Read request terminated by a completion TLP with incorrect byte count.	N/A	5'b011	The returned completion did not match the request stored locally for byte count.	SLVERR
The current completion being delivered has the same tag as an outstanding request, but its requester ID, TC, or Attr fields did not match the parameters of the outstanding request.	N/A	5'b100	The returned completion did not match the request stored locally for requester ID, TC, or Attr fields.	SLVERR
Error in start address.	N/A	5'b101	The completion start address bits [6:0] did not match the request start address.	SLVERR
Request terminated by a completion timeout, or by a function-level reset (FLR) targeting the function that generated the request.	Completion timeout status in AER Uncorrectable Error Status Register and Local Error Status Register FLR_IN_PROGRESS pin is asserted to indicate FLR	5'b111	A completion timeout or FLR terminated the request.	SLVERR
Link down reset indication bit set.	Link down indication bit in the AXI configuration registers is set	N/A	N/A	SLVERR
AXI slave read/write addresses did not match any of the AXI base addresses programmed in the outbound regions.	N/A	5'b10000	The AXI slave read or write address did not match any of the AXI base addresses programmed in the outbound region.	DECERR
Internal error in the PCIe completion path buffers.	Uncorrectable error in the AXI reorder RAM or completion RAM	5'b01000	The ECC/parity decoder flagged an uncorrectable error while reading from any of the completion path buffers.	SLVERR

AXI Slave Write Operation

An AXI write is a split transaction with independent address, data, and response phases associated with the corresponding channels. The client must follow the master protocol for the write address, data, and response channel as described in the AXI specification v1.0. The

PCIe Controller follows the slave protocol for the write address, data, and response channel as described in the AXI specification v1.0.

Figure 20: AXI Slave Write Interface Waveform



When `MASTER_AXI_ARLEN` is not zero, `MASTER_AXI_ARSIZE` must be the maximum value (5).

The client starts a memory write operation by placing the write request parameters on the AXI slave write address channel and asserting the `MASTER_AXI_AWVALID` signal. Additionally, the client must place the write request PCIe TLP attributes on the master write descriptor. The PCIe Controller responds to the request by asserting the `MASTER_AXI_AWREADY` signal for one clock cycle. The PCIe Controller might not be able to accept the request if it does not have adequate credit to transmit the request TLP on the link or the split completion table is full (for a non-posted write).

The client begins the data transfer by placing data on the AXI slave write data channel signals and asserting the `MASTER_AXI_WVALID` signal. The PCIe Controller can pace the data transfer by controlling the `MASTER_AXI_WREADY` output. The client must keep each data word on the bus until the ready signal is sampled high. The `MASTER_AXI_WSTRB` inputs indicates the valid bytes in the data cycle for the first and the last data transfer. The transfer may start and finish at any byte position in the data bus, depending on the starting address alignment of the data block being written to memory. The client should assert `MASTER_AXI_WLAST` for the last cycle of data transfer. The client must not terminate the AXI write burst early; it should issue all write data cycles as indicated by the `MASTER_AXI_AWLEN` signal.

The PCIe Controller expects the byte valids to be contiguous, even for writes of a single DWORD or two DWORDs (with start address aligned to even DWORD boundary).

The PCIe Controller issues a response back to the client on the AXI slave write response channel by asserting `MASTER_AXI_BVALID` when a memory write transaction has been accepted by the PCIe Controller's transaction layer.

Configuration and I/O (non-posted) writes are handled in a similar manner, except that the data payload is only one DWORD long. The PCIe Controller only issues a response for I/O and configuration writes when it receives the completion back from the PCIe link.

Error Handling

For configuratrion and I/O (non-posted) writes, the PCIe Controller might receive a completion with error status from the PCIe link. In this case, the PCIe Controller issues an error response (i.e., `SLVERR`) on the AXI slave write response channel's `MASTER_AXI_BRESP` output by asserting `MASTER_AXI_BVALID`.

AXI ID Management

AXI slave write requests are split transactions; that is, the client can make additional read and write requests while the response for a write request is pending. The client can issue each outstanding write requests with the same `MASTER_AXI_AWID` or unique ones. The PCIe Controller can receive a maximum of 32 outstanding write requests. For non-posted writes, the PCIe Controller internally maps the `MASTER_AXI_AWID` to an internally generated PCIe tag. It looks up this mapping to translate an incoming completion to a corresponding `MASTER_AXI_BID` value sent with the write response channel.

Note: The write data interleaving depth is one; that is, client logic must send the complete data for a particular write request before sending data for another write request.

Zero-length Writes

The AXI slave can initiate a zero-length memory write transaction in the same way as a 1-byte memory write transaction, except the byte valid bits in `MASTER_AXI_WSTRB` are all set to zero during the data cycle. The PCIe Controller sends a memory write request on the PCIe link with the length field set to one double word, and the byte-enable fields set to all zeroes.

Write Transaction Ordering

On the outbound direction, the PCIe Controller is in cut-through mode. That is, there are no store and forward buffers and TLPs flow out to the link strictly in the order they were received from the AXI slave interface. TLPs go out on the link in the same order they were accepted on the AXI slave write interface, whether the writes are posted or non-posted or whether they have the same `MASTER_AXI_AWID` or different. This process ensures PCIe ordering rule compliance.

AXI Configuration and Status Registers

The PCIe Controller uses the concept of regions to send different types of outbound TLPs (memory, message, etc.). This arrangement allows the static information in the TLPs to be pre-programmed in the AXI configuration region registers. The PCIe Controller uses decoding logic to compare the incoming AXI address to the preprogrammed AXI address in the AXI configuration registers to determine which region it belongs to. You can program the region registers using the APB interface. When the PCIe Controller access a region using the corresponding `MASTER_AXI_AW` or `MASTER_AXI_RADDR` region, the address is selected and is used to supply the static pre-programmed TLP information.

Each region has a set of descriptor registers to form the TLP: AXI to PCIe address translation (PCIe address registers) and AXI region decoding (AXI address registers). AXI configuration registers also include link down indication bit. Refer to the **Titanium PCIe Controller Registers User Guide** for details.

The AXI configuration register set also includes inbound address translation registers. For endpoints, the function number and the bar number is used to find the correct address translation register.

PCIe Controller Outbound Accesses

There are two ways to perform an outbound access:

- Static method, which is region based
- Dynamic method, which uses the sideband descriptor

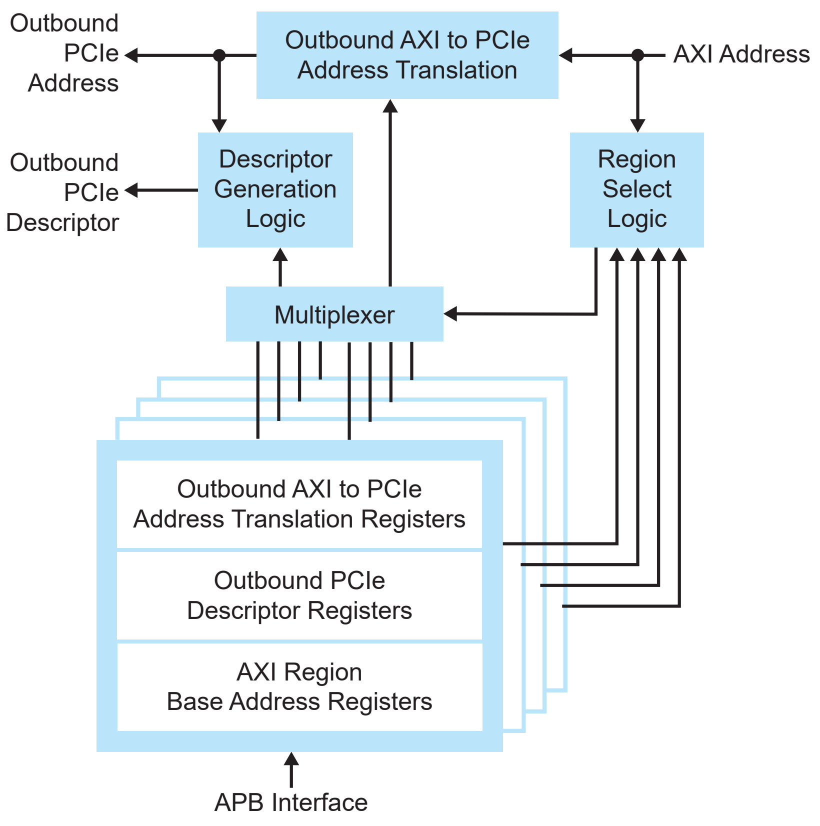
The static method is useful when you only have the TLP type (read or write) and want to send pre-programmed TLP information through the outbound PCIe interface. You use the APB interface to change the TLP information stored in regions, and each region must be re-programmed.

The dynamic method is useful when you already have the TLP information stored outside of the PCIe Controller and you want to send it through the outbound PCIe interface. This method bypasses the AXI address translation logic. If you already have the TLP information, this feature saves re-programming time if you need to change the TLP information.

Outbound Access Using Regions

A maximum of 32 outbound regions can be active at the same time. Each region has registers that control its operation.

Figure 21: AXI Outbound Access Block Diagram


Outbound PCIe Address
Outbound AXI to PCIe Address Translation
AXI Address
Outbound PCIe Descriptor
Descriptor Generation Logic
Region Select Logic
Multiplexer
Outbound AXI to PCIe Address Translation Registers
Outbound PCIe Descriptor Registers
AXI Region Base Address Registers
APB Interface

The outbound access is a two-step process:

1. *Region Setup*—Before accessing the region registers, you program them using the APB interface.
2. *Region Access*—You access the region registers through the outbound AXI interface.

Table 20: Outbound Registers for 1 Region

Registers	Description
Outbound AXI to PCIe address translation registers	Performs address translation from the AXI address to the PCIe address.
Outbound PCIe descriptor registers	Holds the TLP information to be sent outbound.
AXI region base address registers	Holds the AXI region base address and the region size of the corresponding region. The PCIe Controller uses this register set to decode the region by matching it to the incoming AXI address.

Outbound AXI-to-PCIe Address Translation Registers

The PCIe Controller uses these registers (`ob_addr0` and `ob_addr1`) to perform outbound AXI to PCIe address translation for memory and I/O TLPs. For the configuration TLPs, the bus number, device number, and the function number can be programmed in the outbound AXI to PCIe address translation registers if the pass bits are programmed to pass fewer than specific bits.

Table 21: ob_addr1 Outbound AXI-to-PCIe Address Translation Registers

Bits	Memory and I/O TLPs	Vendor Defined Messages	Normal Messages	Default value
31:0	Upper [63:32] bits of the PCIe address.	Vendor-defined message header [127:96]	Reserved for normal messages	32'd0

Table 22: ob_addr0 Outbound AXI-to-PCIe Address Translation Registers

Bits	Memory and I/O TLPs	Vendor Defined Messages	Normal Messages	Default value
31:28	Lower [31:28] bits of the PCIe address.	Vendor-defined message header [95:92]	Reserved for normal messages	4'd0
27:20	Lower [27:20] bits of the PCIe address.	Vendor-defined message header [91:84]	Reserved for normal messages	8'd0
19:15	Lower [19:15] bits of the PCIe address.	Vendor-defined message header [83:79]	Reserved for normal messages	5'd0
14:12	Lower [14:12] bits of the PCIe address.	Vendor-defined message header [78:76]	Reserved for normal messages	3'd0
11:8	Lower [11:8] bits of the PCIe address.	Vendor-defined message header [75:72]	Reserved for normal messages	4'd0
7:6	Reserved.	Reserved	Reserved	2'd0
5:0	Number of address bits passed from AXI to PCIe. The PCIe Controller passes the programmed value + 1 bits from AXI to PCIe. The minimum value to be programmed into this field is 7 because the translation logic replaces the PCIe address's lower 8 bits with zeros.	Reserved	Reserved	6'd0

Some of the AXI slave outbound sddress (`MASTER_AXI_AWADDR`) bits convey the `MSG_ROUTING` and `MSG_CODE` fields of the message TLPs.

Table 23: MSG_ROUTING and MSG_CODE Fields

MASTER_ AXI_ AWADDR bits	Vendor-Defined Message	Normal Message
16	0: MSG with data 1: MSG without data	0: MSG with data 1: MSG without data
15	0: MSG_CODE = 0x7E 1: MSG_CODE = 0x7F	MSG_CODE[7]
14:12	MSG_ROUTING	MSG_CODE[6:4]
11:8	Reserved	MSG_CODE[3:0]
7:5	Reserved	MSG_ROUTING

Outbound PCIe Descriptor Registers

These registers hold the static information in the TLP (e.g., function number, requester ID, etc.).

Table 24: desc0: Outbound PCIe Descriptor Register for Different TLP Accesses

Bits	Memory and I/O TLPs	Configuration TLP	Vendor-Defined Message TLP	Normal Message TLP
3:0	Transaction type 0010: Memory I/O 0110: I/O	Transaction type 1010: Type 0 configuration 1011: Type1 configuration	Transaction type 1101: Vendor-defined message	Transaction type 1100: Normal message
6:4	PCIe attributes [6] ID-based ordering [5] Relaxed ordering [4] No snoop	Same as Memory I/O TLP	Same as Memory I/O TLP	Same as Memory I/O TLP

Bits	Memory and I/O TLPs	Configuration TLP	Vendor-Defined Message TLP	Normal Message TLP
8:7	ATS[1:0]	Reserved	Bit 8: Carries bit [64] of the vendor defined message header. Bit 7: Reserved.	Reserved for normal messages. Bit 8: Carries bit [64] of the PRI message header.
15:9	Reserved	Reserved	Carries [71:65] of the vendor defined message header.	Reserved for normal messages.
16	If desc0 [8:7] bits are set to 2'b01 (i.e., it is an ATS translation request) or if it is a memory read request, this bit is used as no write (NW) flag. In this case the address must be aligned (i.e., address bits 11:0 must be reserved as per the PCIe protocol specification).	Reserved	Reserved	Reserved
19:17	PCIe traffic class	PCIe traffic class	PCIe traffic class	PCIe traffic class
20	When the request is a memory write transaction, setting this bit causes the PCIe Controller to poison the memory write TLP being sent. This bit has no effect for other transactions.	Reserved	Reserved	Reserved
21	Force ECRC insertion. Setting this bit to 1 forces the PCIe Controller to append a TLP digest containing ECRC to the TLP, even when ECRC is not enabled for the function generating the request.	Same as Memory I/O TLP	Same as Memory I/ O TLP	Same as Memory I/ O TLP
22	Reserved	Reserved	Reserved	Reserved
23	Enables the client to provide the bus and device numbers to be used in the requester ID. 0: The PCIe Controller uses the captured values of the bus and device numbers to form the Requester ID. 1: The PCIe Controller uses the bus and device numbers supplied by the client on desc1[7:0] and desc0[31:27] to form the requester ID. This bit must always be set while originating requests in root port mode, and the corresponding bus and device numbers must be placed on desc1[7:0] and desc0[31:27].	1: The PCIe Controller uses the bus and device numbers supplied by the client on addr0[27:20] and addr0[19:15] to form the completer ID. This bit must always be set while originating requests in root port mode, and the corresponding bus and device numbers must be placed on addr0[27:20] and addr0[19:15].	Same as Memory I/ O TLP	Same as Memory I/ O TLP
31:24	PCI function number associated with the request. ARI mode: All 8 bits are used to indicate the requesting function. Non-ARI mode: Bits [26:24] represent the function number. The client must always specify the function number regardless of the bit [23] setting. Bits [31:27] specify the device number to be used within the requester ID, when bit [23] is set.	Reserved	Same as Memory I/ O TLP	Same as Memory I/ O TLP

Table 25: desc1: Outbound PCIe Descriptor Register for Different TLP Accesses

Bits	Memory and I/O TLPs	Configuration TLP	Vendor-Defined Message TLP	Normal Message TLP
7:0	When desc0[23] is set, this field must specify the bus number to be used for the requester ID. Otherwise, this field is ignored.	Reserved	Same as Memory I/O TLP	Same as Memory I/O TLP

Bits	Memory and I/O TLPs	Configuration TLP	Vendor-Defined Message TLP	Normal Message TLP
31:8	Reserved	Reserved	Reserved	Reserved

Table 26: desc2: Outbound PCIe Descriptor Register for Different TLP Accesses

Bits	Memory and I/O TLPs	Configuration TLP	Vendor-Defined Message TLP	Normal Message TLP
7:0	If index bit is 0 this value is taken as the TLP steering tag for the hint. If index bit is 1, this value [7:0] is used as a pointer to the table holding the steering tag values.	Reserved	Reserved	Reserved
8	Index bit	Reserved	Reserved	Reserved
10:9	Value of PH [1:0] associated with the hint.	Reserved	Reserved	Reserved
11	TPH length	Reserved	Reserved	Reserved
12	Set when the request has a transaction processing hint associated with it.	Reserved	Reserved	Reserved
20:13	Reserved	Reserved	Reserved	Reserved

Table 27: desc3: Outbound PCIe Descriptor Register for Different TLP Accesses

Bits	Memory and I/O TLPs	Configuration TLP	Vendor-Defined Message TLP	Normal Message TLP
0	PASID present bit	1'b0	1'b0	1'b0
20:1	PASID value	20'd0	20'd0	20'd0
21	Privilege mode access requested	1'b0	1'b0	1'b0
22	Execute mode access requested	1'b0	1'b0	1'b0
31:23	Reserved	Reserved	Reserved	Reserved

AXI Region Base Address Registers

The PCIe Controller uses region select logic to match the outbound AXI address and the pre-programmed AXI address (in the AXI region base address registers for each region) to determine the region to which it belongs. The matching is done from region 0 to <*max regions*> - 1. The comparator selects the first matching region as the region number used to pick the static TLP information (PCIe descriptor) as well as the PCIe address (for address translation). The AXI region sizes and region base addresses are programmable.

Note: Overlapping regions are not supported.

Table 28: AXI Region Base Address Registers

Register	Bits	Allocation	Default
axi_addr1	31:0	Upper [63:32] bits of the AXI region base address.	32'd0
axi_addr0	31:8	Lower [31:8] bits of the AXI region base address.	24'd0
	7:6	Reserved	2'd0
	5:0	The programmed value in this field + 1 gives the region size. The minimum value to be programmed into this field is 7 because the lower 8 bits of the AXI region base address are replaced by zeros by the region select logic. The minimum region size is 256 bytes.	6'd0

All AXI regions has their start address aligned to the region size, which is programmed through the AXI Region Base Address Register `axi_addr0 [5:0]`. If the select logic does not find a match, the PCIe Controller responds with a DEC ERR over the AXI interface.

Outbound Access through the Sideband Descriptor

This topic describes how to send an AXI outbound address packet directly—without doing any address translation. This method is useful for dynamic address translations; that is, the client does not have enough time to program one of the 32 region registers.

In this method, the PCIe Controller drives the translated outbound PCIe address (PCIe descriptor) directly on `MASTER_AXI_AWADDR` and `MASTER_AXI_ARADDR`. It does not perform address translation performed on the AXI address. A sideband access enable bit in `MASTER_AXI_AWUSER` and `MASTER_AXI_ARUSER` gives the sideband access priority over the region access.

Table 29: AXI Slave Sideband Signal Description (MASTER_AXI_AWUSER and MASTER_AXI_ARUSER)

Bit	Memory or I/O TLP	Configuration TLP	Message TLP
3:0	Transaction type: 0000: Memory read 0010: Memory write 0110: I/O write 0100: I/O read All other values are reserved.	Transaction type: 1010: Type 0 configuration write 1000: Type 0 configuration read 1011: Type 1 configuration write 1001: Type 1 configuration read All other values are reserved	Transaction type: 1100: Normal message 1101: Vendor-defined message All other values are reserved
6:4	PCIe attributes associated with the request. 4: No Snoop 5: Relaxed Ordering 6: IDO	Same as Memory or I/O TLP	Same as Memory or I/O TLP
7	ATS bit 0	Reserved	Reserved
15:8	8: ATS bit 1 15:9: Reserved	Reserved	For vendor defined messages, this field carries bits [71:64] of the message header. For all other requests, this field is reserved.
16	If bits [8:7] are set to 2'b01 (i.e., ATS translation request) and if it is a memory read request, bit [16] is used as a no write (NW) flag. In this case the address must be aligned; that is, address bits [11:0] must be reserved as per the PCIe protocol specification.	Reserved	Reserved
19:17	PCIe Traffic Class (TC) associated with the request.	PCIe Traffic Class (TC) associated with the request.	PCIe Traffic Class (TC) associated with the request.
20	When the request is a memory write transaction, setting this bit causes the PCIe Controller to poison the memory write TLP being sent. This bit has no effect for other transaction types.	Reserved.	Reserved
21	Force ECRC insertion. Setting this bit to 1 forces the PCIe Controller to append a TLP digest containing an ECRC to the TLP, even when ECRC is not enabled for the function generating the request.	Same as Memory or I/O TLP	Same as Memory or I/O TLP

Bit	Memory or I/O TLP	Configuration TLP	Message TLP
22	Enables the client to provide the bus and device numbers to be used in the requester ID. 0: The PCIe Controller uses the captured values of the bus and device numbers to form the Requester ID. 1: The PCIe Controller uses the bus and device numbers supplied by the client on bits [38:31] and [30:26] to form the requester ID. This bit must always be set while originating requests in root port mode, and the corresponding bus and device numbers must be placed on bits [38:31].	Same as Memory or I/O TLP	Same as Memory or I/O TLP
30:23	PCI function number associated with the request. ARI mode: All 8 bits are used to indicate the requesting function. Legacy mode: Only bits [25:23] are used, and bits [30:26] are used to specify the device number to be used within the requester ID, when bit [22] is set.	Same as Memory or I/O TLP	Same as Memory or I/O TLP
38:31	When bit [22] is set, this field must specify the bus number to be used for the requester ID. Otherwise, this field is ignored.	Same as Memory or I/O TLP	Same as Memory or I/O TLP
46:39	Reserved	Reserved	MSG CODE, 0x7E or 0x7F for vendor-defined messages.
49:47	Reserved	Reserved	MSG routing
57:50	TPH ST TAG [7:0]	TPH ST TAG [7:0]	TPH ST TAG [7:0]
58	TPH INDEX	TPH INDEX	TPH INDEX.
60:59	TPH TYPE [1:0]	TPH TYPE [1:0]	TPH TYPE [1:0]
61	TPH length	TPH length	TPH length
62	TPH present	TPH present	TPH present
63	PASID present	1'b0	1'b0
83:64	PASID value	20'd0	20'd0
84	Privilege mode requested	1'b0	1'b0
85	Execute mode requested	1'b0	1'b0
86	Reserved	Reserved	Zero data message
87	MASTER_AXI_AWUSER Only Valid bit to validate sideband access. 1: Descriptor is taken from [87:0] 0: Descriptor is taken from the region registers.	Same as Memory or I/O TLP	Same as Memory or I/O TLP
	MASTER_AXI_ARUSER Only Valid bit to enable sideband access.	Valid bit to enable sideband access.	Valid bit to enable sideband access.

Table 30: Generating Page Request Messages

Register	Bits	Field Name	Description	Default
addr1	31:0	Page Address	Page address upper bits.	32'd0
addr0	11:8	[8:5] Page request group index	Page request group index.	4'd0

Register	Bits	Field Name	Description	Default
desc0	31:12	[31:12] Page address	Page address. Contains the untranslated page address to be loaded. For pages larger than 4,096 bytes, the least significant bits are ignored. For example, for an 8,096 byte page, the least significant bits are ignored.	20'd0
	8	R	Read access requested. Set: The requesting function seeks read access to the associated page. Clear: The requesting function will not read the associated page.	1'b0
	9	W	Write access requested. Set: The requesting function seeks write access and/or zero-length read access to the associated page. Clear: The requesting function will not write to the associated page.	1'b0
	10	L	Write access requested. Set: The requesting function seeks write access and/or zero-length read access to the associated page. Clear: The requesting function will not write to the associated page.	1'b0
	15:11	[4:0] Page request group index	Page request group index. Contains a function provided identifier for the associated page request. A function does not need ot use all available PRG index values. A host shall never respond with a PRG index that has not been previously issued by the function and that is not currently an outstanding request PRG index (except when issuing a response failure, in which case the host need not preserve the associated request's PRG index value in the error response).	5'b0

Table 31: Generating Page Response Messages (addr1 Register)

Bits	Field Name	Description	Default
8:0	PRG index	Page request group index. This field contains a function provided index to which the root port is responding. A given PRG index will receive exactly one response per instance of PRG (with the possible exception of a response failure).	9'd0
15:12	Response code	Contains the response type of the associated PRG. 0000b: Success 0001b: Invalid request 1110b: 0010b: Unused 1111b: Response failure A detailed description of each response code is available in PCIe Specification.	4'd0
31:16	Destination ID	Destination device ID.	16'd0

Table 32: Stop Marker Message (desc0 Register)

Bits	Field Name	Description	Default
8	R	Read access requested. Must be 1'b0.	1'b0
9	W	Write access requested. Must be 1'b0.	1'b0
10	L	Last request in PRG. 1'b1	1'b0
15:11	Marker type	4'b0000	4'd0

Table 33: Invalid Request Messages

Register	Bits	Field Name	Description	Default
addr1	31:16	Device ID	Destination device ID	32'd0

Register	Bits	Field Name	Description	Default
Data 256, 128, 64 bit bus	7:0	Untranslated address [63:56]	-	-
	15:8	Untranslated address [55:48]		1'b0
	23:16	Untranslated address [47:40]	-	-
	31:24	Untranslated address [39:32]	-	-
	39:32	Untranslated address [31:24]	-	
	47:40	Untranslated address [23:16]	-	
	55:52	Untranslated address [15:12]	-	
	51	S	Indicates if the range being invalidated is greater than 4,096 bytes. Its meaning is the same as for the translation completion.	
	56	GI	Global invalidate. Indicates that the invalidation request message affects all PASID values. The ATC ignores this bit if the global invalidate supported bit is clear. This field is reserved if PASID is not support by configuration.	
desc1	28:24	ITAG	Constrained to the values 0 to 31. Used by the TA to uniquely identify requests it issues. A TA must ensure that once an ITag is used, it is not reused until either released by the corresponding invalidate completions or by a vendor-specific timeout mechanism.	5'd0

Table 34: Invalidation Completion Message

Register	Bits	Field Name	Description	Default
addr1	2:0	CC	Completion Count. Indicates the number of individual invalidate completion messages that must be sent for the associated invalidate request. Setting the CC field to 0 indicates that eight responses must be sent. The TA is responsible for collecting all responses associated with a given tag before considering the corresponding invalidate request to be complete.	3'd0
	31:16	Device ID	Set to the TA's requester ID.	16'd0
desc0	15:8	ITAG vector [7:0]	Indicate which invalidate request has been completed. Bit 0 corresponds to the ITag field value of 0.	8'd0
addr0	31:8	ITAG vector [31:8]	-	24'd0

MSI Memory Writes

In endpoint mode, the client can request interrupt service by initiating message signaled interrupts (MSI). MSI uses memory write requests (using the memory write request format) to represent interrupt messages. The client generates the MSI memory write. The typical procedure for initiating an MSI request is:

1. Host initializes the MSI capabilities of each function in the endpoint via configuration writes:
   a. Host configures lower 32 address bits in the MSI Message Address Low Register.
   b. Host configures upper 32 address bits in the MSI Message Address High Register.
   c. Host configures data in MSI Message Data Register.
   d. Host configures per-vector mask in MSI Mask Register.
   e. Host enables MSI by configuring the MSI Control Register.
2. Check for MSI enable by sampling the `MSI_ENABLE` output. When a function's `MSI_ENABLE` is 1, the function can generate an MSI.
3. Wait for a new outbound MSI request for a function or an `MSI_MASK` cleared event for a pending MSI.
   a. New MSI request, go to step 4.
   b. Pending MSI request (mask has been cleared by the host), go to step 5.

4. New outbound MSI request for a function:
    a. Read the MSI address and data registers from the function's MSI Capability registers.
    b. Check whether that the MSI vector is not masked by sampling `MSI_MASK`.
    c. If masked, set the corresponding bit in the MSI Pending Bits Register using one of these methods:
        i. *Use APB (default)*—You can set or clear the MSI Pending Status Register bits by writing to them through the APB interface.
        ii. *Set directly*—You can set or clear the MSI Pending Status Register bits directly using `MSI_PENDING_STATUS_IN`.

        Note: You select the mode by programming bit [9] (MSI Pending Status In Mode Select) in the Debug Mux Control 2 Register local management register.

        iii. Go to step 2.
    d. If not masked:
        i. Allocate an MSI region on the AXI interface by programming the AXI region base address registers.
        ii. Program the AXI to PCIe address translation registers for the allocated MSI region with the MSI address.
        iii. Program the PCIe descriptor registers for the allocated MSI region with required values mentioned in the Memory Write column of the PCIe descriptor registers table.
        iv. Generate an outbound write to the MSI region. The AXI write data is <*MSI vector number*> + <*MSI data register value*>.
        v. Go to step 2.
5. MSI mask is cleared for a pending MSI vector for a function:
    a. Read the MSI address and data registers from the function's MSI capability registers.
    b. Transmit the MSI vector:
        i. Allocate an MSI region on the AXI interface by programming the AXI region base address registers.
        ii. Program the AXI to PCIe address translation registers for the allocated MSI region with the MSI address.
        iii. Program the PCIe descriptor registers for the allocated MSI region with required values mentioned in the Memory Write column of the PCIe descriptor registers table.
        iv. Generate an outbound write to the MSI region. The AXI write data is <*MSI vector number*> + <*MSI data register value*>.
    c. Clear the corresponding bit in the MSI Pending Bits Register by writing to it through the APB local management interface.
    d. Go to step 2.

MSI-X Memory Writes

In endpoint mode, the client can request interrupt service by initiating MSI-X. MSI-X uses memory write requests (using the memory write request format) to represent interrupt messages. The client generates the the memory write. The typical procedure for initiating an MSI-X request is:

1. Client sets up the location of the MSI-X table and MSI-X pending bit array in the endpoint function's memory space:
    a. Program the MSI-X table location in the MSI-X Table Offset Register.
    b. Program the MSI-X pending bit array in the MSI-X Pending Interrupt Register.
2. Host initializes the function's MSI-X capabilities in the endpoint:
    a. Host reads the MSI-X table location from the MSI-X Table Offset Register.
    b. Host reads the MSI-X pending bit array location from the MSI-X Pending Interrupt Register.
    c. Host initializes the MSI-X vectors by writing to each of the MSI-X table locations.

   d. Host enables MSI-X in each function by configuring the MSI-X Control Register.
3. Check whether MSI-X is enabled by sampling `MSIX_ENABLE`.
4. When a function's `MSIX_ENABLE` is 1, the function can generate an MSI-X with the following steps:
   a. Read the MSI-X table entry for each vector to get the MSI-X address, data, and mask settings for that MSI-X vector.
   b. Check whether the MSI-X vector is not masked.
   c. If masked, set the corresponding bit in the MSI-X pending bit array by writing the corresponding bit in the pending bit array memory location.
   d. If not masked:
      i. Allocate an MSI-X region on the AXI interface by programming the AXI Region Base Address Registers.
      ii. Program the AXI to PCIe address translation registers for the allocated MSI-X region with the MSI-X address.
      iii. Program the PCIe descriptor registers for the allocated MSI-X region with required values mentioned in the Memory Write column of the PCIe descriptor registers table.
      iv. Generate an outbound write to the MSI-X region. The AXI write data is the MSI-X vector data.

Outstanding Non-Posted Requests

The AXI slave read and write interfaces (I/O and configuration requests in root port mode). can send non-posted requests. The client should ensure that the sum of outstanding non-posted requests over the two AXI slave interfaces is less than the maximum number of outstanding non-posted requests that can be handled by the PCIe Controller's split completion table.

Note: The PCIe Controller back pressures the AXI slave interface in case the number of non-posted requests exceeds the maximum that it can handle, which affects performance.

Ordering between AXI Slave Write and Read Channels

On the outbound direction, the PCIe Controller is in cut-through mode. The AXI logic has the store and forward buffers. The PCIe Controller arbitrates outbound (read and write channels) transactions it receives on the AXI slave interface after the asynchronous FIFO buffer for the AXI slave write and read channels. If a read and a write request are both placed on the arbiter in the same cycle, is a programmable priority bit in the local management space indicates which request should be sent out on the link first.

Outbound Ordering (Endpoint)

Table 35: Endpoint Outbound Ordering

Row Pass Column?		Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
			Read Request (Col 3)	NPR with Data (Col 4)	
Posted request		No	Yes	N/A	Yes
Non-posted request	Read request	Order presented on AXI[4]	No	N/A	Yes/No
	NPR with data	N/A	N/A	N/A	N/A
Completion		Order presented on AXI[5]	Yes	N/A	No

[4] To ensure that non-posted are not cross posted, your application should wait for a posted response before issuing non-posted requests.
[5] To ensure that completions are not cross posted, your application should wait for a posted response before issuing completions to the AXI master.

Outbound Ordering (Root Port)

Table 36: Root Port Outbound Ordering

Row Pass Column?		Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
			Read Request (Col 3)	NPR with Data (Col 4)	
Posted request		No	Yes[6]	No[7]	Yes
Non-posted request	Read request	Order presented on AXI[8]	No	Yes	Yes/No
	NPR with data	No	Yes/No	No	Yes/No
Completion		Order presented on AXI[9]	Yes	Yes	No

If read and write transactions are serviceable (i.e., ready to be sent to the link) in the same clock cycle, the Enable AXI Bridge Write Priority bit in Debug Mux Control register dictates whether the write or the read is serviced.

- A write transaction is deemed serviceable from the asynchronous FIFO when all data has reached the FIFO.
- A read transaction is deemed serviceable when the read transaction sits in the asynchronous FIFO.

The completions from the AXI master interface are not ordered with respect to the read and write transactions on the AXI slave interface. All three transactions constitute TLPs flowing outbound.

Completion Error Codes

Table 37: Completion Error Codes

Error Code	Description
3'b000	Normal termination (all data received).
3'b001	The completion TLP is poisoned.
3'b010	Request terminated by a completion with UR, CA, or CRS status.
3'b011	Request terminated by a completion with incorrect byte count.
3'b100	The current completion being delivered has the same tag as an outstanding request; however, its requester ID, TC, or Attr fields did not match the parameters of the outstanding request.
3'b101	Starting address error. The low address bits in the completion TLP header did not match the starting address of the next expected byte for the request.
3'b110	Invalid tag. This completion does not match the tags of any outstanding request.
3'b111	Request terminated by a completion timeout or by an FLR targeted at the function that generated the request.

Completion Status Codes

Table 38: Completion Status Codes

Status Code	Description
00	Good.
01	Unsupported request (UR).
10	Completer abort.
11	Retry status.

[6] When a non-posted request is blocked, a posted request can pass it.
[7] The posted packets are blocked if non-posted packets are stuck in the pipeline.
[8] If the write is blocked, the read can go ahead of the write. If there is an address overlap between the write and read, your application can wait for the write response before giving the read.
[9] If the posted request is blocked, the completion can pass the posted request.

AXI Master and Slave Read/Write Length Limitations

For outbound transfers:

- Reads are limited by minimums (`AXI_SLAVE_MAX_RD_TRANSFER_SIZE`, `MAX_READ_REQUEST_SIZE`). The PCIe Controller AXI bridge cannot handle read requests if the outbound read request length is greater than `MAX_READ_REQUEST_SIZE`.
- Writes are limited by `AXI_SLAVE_MAX_WR_TRANSFER_SIZE`. If `AXI_SLAVE_MAX_WR_TRANSFER_SIZE` is greater than `MAX_PAYLOAD _SIZE`—and the outbound write request length is greater than `MAX_PAYLOAD _SIZE`—the write requests are split at `MAX_PAYLOAD_SIZE` boundary.

For inbound transfers:

- Writes are limited by `MAX_PAYLOAD_SIZE`. If `AXI_MASTER_MAX_WR_TRANSFER_SIZE` is less than `MAX_PAYLOAD_SIZE` and the inbound packet length is greater than `AXI_MASTER_MAX_WR_TRANSFER_SIZE`, requests are split into `AXI_MASTER_MAX_WR_TRANSFER_SIZE` packets. The same AXI ID is assigned to all split packets.
- Reads are limited by `MAX_READ_REQUEST_SIZE`. If `AXI_MASTER_MAX_RD_TRANSFER_SIZE` is less than `MAX_READ_REQUEST_SIZE` and the inbound packet length is greater than `AXI_MASTER_MAX_RD_TRANSFER_SIZE`, requests are split into *min*(`AXI_MASTER_MAX_RD_TRANSFER_SIZE`, `MAX_PAYLOAD_SIZE`) packets. The same AXI ID is assigned to all split packets. The splits measure a length of *min*(`AXI_MASTER_MAX_RD_TRANSFER_SIZE`, `MAX_PAYLOAD_SIZE`). The start address is aligned to an RCB boundary and the next address is calculated by adding *min*(`AXI_MASTER_MAX_RD_TRANSFER_SIZE`, `MAX_PAYLOAD_SIZE`).

Note: `MAX_PAYLOAD_SIZE` and `MAX_READ_REQUEST_SIZE` are host-configured values in the device control register.

`AXI_MASTER_MAX_RD_TRANSFER_SIZE`, `AXI_MASTER_MAX_WR_TRANSFER_SIZE` are the maximum transfer sizes for the AXI master.

`AXI_SLAVE_MAX_RD_TRANSFER_SIZE`, `AXI_SLAVE_MAX_WR_TRANSFER_SIZE` are the maximum transfer sizes for the AXI slave.

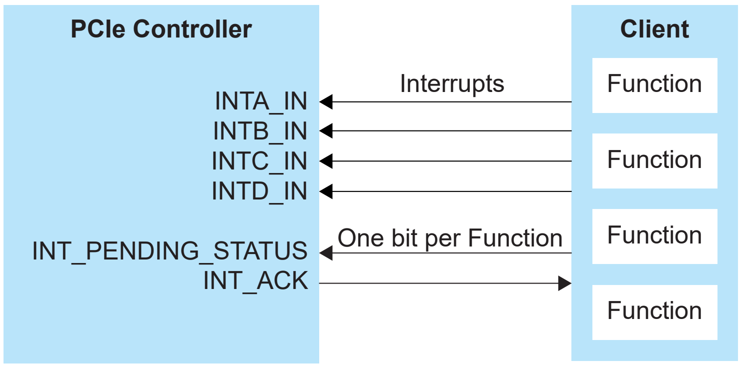
Interrupt Interface

The interrupt interface has a master interface (root port mode) and a target interface (endpoint mode). The master interrupt interface communicates the interrupt events signaled by downstream endpoints to a local interrupt controller. The target interrupt interface allows endpoint clients to signal their interrupt state to a remote root port.

Legacy Interrupt Operation

In legacy mode, the PCIe Controller emulates the four PCI interrupt pins (`INTA_IN`, `INTB_IN`, `INTC_IN`, and `INTD_IN`). Multiple functions can share the same interrupt pin. On the endpoint side, the client signals interrupt conditions to the PCIe Controller using four distinct interrupt inputs.

Figure 22: Legacy Endpoint Interrupt Interface



The PCIe Controller communicates the state of each interrupt input by sending `Assert_INT`*x* or `Deassert_INT`*x* messages on the PCIe link. It sends an assert message when the corresponding interrupt input transitions from low to high, and sends a de-assert message when the input transitions back to low. The high-to-low transition usually occurs when the interrupt has been serviced.

After signaling each transition, the client must wait for the PCIe Controller to assert `INT_ACK` before signaling another transition on the same interrupt pin.

You can modify the default interrupt assignments by writing to the Interrupt Pin register through the local management bus.

The client must provide the interrupt pending status of each of its functions to the PCIe Controller through the PCIe Controller's `INT_PENDING_STATUS` input, so that the status can be read from the PCIe link through the function's PCI Status Register. The client must set `INT_PENDING_STATUS` high when there is an interrupt pending from the function, and set it low when the interrupt has been serviced.

Note: You cannot use MSI or MSI-X interrupts when using legacy interrupts. The client must also check the state of the `INT`*x* disable bits in the associated function's PCI Command Register before generating a legacy interrupt. The `INT`*x* disable bit states are available in the PCIe Controller's `FUNCTION_STATUS` output.

MSI and MSI-X Interrupt Modes

As a root port, the PCIe Controller receives MSI or MSI-X messages from downstream endpoints. It processes the messages like a normal memory write request received from the link. The PCIe Controller transfers the address and data associated with the (message) memory write request over the same target memory write interface used to transfer normal memory write requests. Software running in the root port is responsible for monitoring the writes to the MSI-assigned area in memory and servicing the interrupts.

MSI Interrupts

In this mode, the interrupt conditions are communicated from the endpoint to the root port via messages. When an interrupt condition occurs, the endpoint sends a message with information that identifies the interrupt's origin. Each message has an address and a data value to be written. Each PCI function supported by a device can be assigned a separate memory address, thus providing separate virtual channels for each function that generates interrupts. Additionally, MSI allows (and the PCIe Controller supports) a maximum of 32 distinct data patterns in the messages generated by each PCI function, and each pattern can be assigned to an interrupt condition within the function.

On the endpoint side, the client signals an interrupt condition via the AXI slave interface. The client constructs an AXI write transaction with the configured address and data value in the MSI capability structure. The client should assign the address to a region register that translates the AXI write transaction into a PCIe memory write TLP. The PCIe Controller then forwards the memory write TLP to the link. The PCIe Controller also supports 32 mask bits and pending interrupt bits for each function. When an interrupt condition occurs, the client should check that the corresponding mask bit is not set before sending the AXI write transaction. The client should read the MSI address data values from the MSI capability structure after the enumeration to construct MSI write TLPs to be sent on the AXI slave interface.

On the root port side, the PCIe Controller decodes MSI messages received from the link and passes them to the client through the AXI master interface as normal write requests.

MSI-X Interrupts

This mode is similar to MSI, except MSI-X allows a much larger number of distinct interrupt conditions to be communicated—as many as 2,048 per function—and lets you define a distinct address for each conditions. MSI-X requires the endpoint's memory to store two tables:

- The MSI-X table contains the address and data patterns to be used for each interrupt condition as well as individual enable/mask bits.
- The pending bit array (PBA) table stores the status of each interrupt condition.

Interrupt conditions are communicated from the endpoint to the root port via messages (write requests) like MSI mode. The PCIe Controller supports MSI-X interrupts by providing a dedicated interface to the client on the endpoint side to send MSI-X messages. The MSI-X table and PBA must be stored in client memory. When an MSI-X message is to be sent, the client communicates the message's address and data information to the PCIe Controller via an AXI slave write transaction. The client should assign the address to a region register that translates the AXI write transaction into a PCIe memory write TLP. The PCIe Controller then forwards the memory write TLP to the link.

On the root port side, the operation is similar to MSI mode. The PCIe Controller decodes MSI-X messages received from the link and passes them to the client through the AXI master interface as normal write requests.

Interrupt Sideband Signals

These signals let you generate custom interrupt signals. You can OR required signals from this vector to form interrupt signals. These signals are already masked internally using the corresponding mask bits given in the local management space for each kind of error.

Table 39: Interrupt Sideband Signals

Bit	Description
0	AXI slave reorder SRAM ECC uncorrectable error.
1	AXI slave WFIFO SRAM ECC uncorrectable error.
2	AXI master RFIFO SRAM ECC uncorrectable error.
3	Replay RAM parity error.
4	PNP RX FIFO parity error.
5	Completion RX FIFO parity error.
6	PNP RX FIFO pverflow.
7	Completion RX FIFO pverflow.
8	Replay timeout.
9	Replay timer rollover.
10	PHY error.
11	Malformed TLP received.
12	Unexpected completion received.
13	Flow control error.
14	Completion timeout.
15	This bit is set when the host toggles the Hardware Autonomous Width Change bit in the Link Control Register through a configuration write.
16	Unmapped TC error.
17	Set when the MSI mask register value in the MSI capability register changes value in any of the PCIe Controller's functions.

Bit	Description
18	Set whenever the MSI-X function mask register value in the MSI-X capability register changes in any of the PCIe Controller's functions.
19	Set whenever any bit in the MSI mask register is cleared in any of the PCIe Controller's functions.
20	Set whenever any bit in the MSI mask register is set in any of the PCIe Controller's functions.
21	Set whenever the MSI-X function mask register is cleared in any of the PCIe Controller's functions.
22	Set whenever the MSI-X function mask register is set in any of the PCIe Controller's functions.
23	Set when a NFTS timeout occurs during Rx_L0s exit.
24	Uncorrectable error detected in SC table state RAM protect module.
25	Uncorrectable error detected in SC table timer RAM protect module.
26	Uncorrectable error detected in SC table byte count RAM protect module.
27	Link equalization requests interrupt. Endpoint. Indicates that the PCIe Controller has detected a problem with equalization and automatically requests an equalization retry at the end of equalization. Root port: Reserved.

Clock Sources

The main PCIe Controller clock is derived directly from the PMA PLL; the clock frequency is configuration dependent. For example, the PCIe Gen4 configuration requires the PCIe Controller to run at 500 MHz.

The PCIe Controller clock domain is transparent to the user application.

Table 40: Clock Sources

Clock	Direction	Frequency (MHz)	Descriptions
AXI_CLK	Input	125 - 250	AXI interface clock.
USER_APB_CLK	Input	20 - 200	APB interface clock.
PM_CLK	Output	40	Free-running clock used for low power state transitions.

The `AXI_CLK` clock can be derived from a PLL output. However, you need to ensure that the `AXI_CLK` frequency complies with the PCIe link total bandwidth. For example, four lanes of Gen4 run at a 64 Gbps link bandwidth. To fully utilize the link bandwidth, `AXI_CLK` must operate at 250 MHz in your application.

The `USER_APB_CLK` clock can also be derived from a PLL output.

The `PM_CLK` is used for power management. The PCIe Controller outputs `PM_CLK` so you can utilize the same clock resource.

All clocks are asynchronous; the PCIe Controller handles the clock synchronization internally.

In the Efinity Interface Designer the PCIe block has an option **Reference clock from on-board crystal**. Elitestek recommends that you turn this option off when using a reference clock slot from a PCIe slot to ensure that stable reference clock is present during PHY configuration. When this option is disabled, you need to create a PLL instance with a specific PLL resource and settings as the temporary PCIe reference clock while the PHY is configuring. When the PHY completes configuration, the reference clock reverts to the edge card connector.

If you turn this option off, use these settings:

- Create a PLL block with a resource of BR0 or BR1.
- Enable:
  - `CLKOUT4` signal if you are using QUAD 0.
  - `CLKOUT2` signal if you are using QUAD 2.
-
- Configure the PLL in local feedback mode.

You can use the `CLKOUT4` signal as a clock source for core logic after the PHY completes configuration.

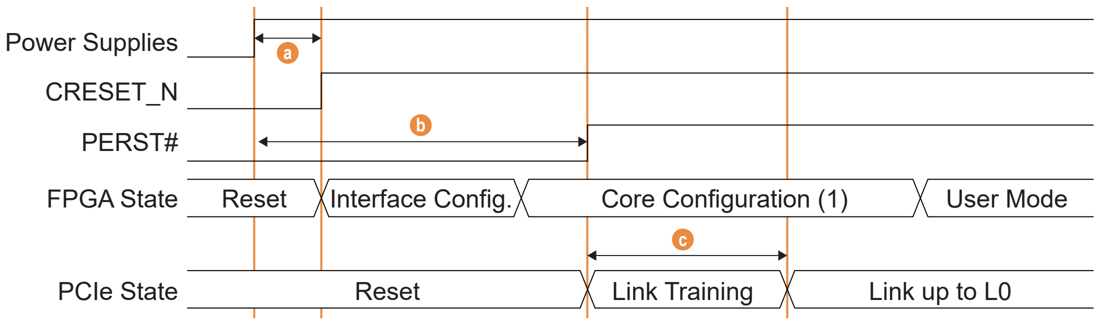
Link Control

The following topics describe the process for link up, link down, and reset.

Link Up

Upon power on reset, the PCIe Controller is ready for link training in 100 ms. After the `PERST#` signal is deasserted, the PCIe link goes through training and achieves link active (L0) state in another 100 ms.

Figure 23: PCIe Controller Link Up Mechanism



Parameter	Min.	Typ.	Max.	Units	Description
a	10	-	500	µs	CRESET_N release time after power supplies are stable.
b (2)	100	-	-	ms	Minimum PERST# signal active time from the PCIe host.
c	-	-	100	ms	Maximum time required for the PCIe device to enter the L0 state after PERST# is released.

1. The core configuration length depends on the configuration mode. In some modes the FPGA enters user mode before the PCIe link up.
2. To meet b, Efinix recommends you use SPI passive programming or SPI active programming with the following configuration:

Programing Mode	External Clock Frequency (MHz)	Internal Oscillator CLock Divider
SPI active x1	8.04	DIV4, DIV2, DIV1
SPI active x2	4.04	DIV8, DIV4, DIV2, DIV1
SPI active x4	2.04	DIV8, DIV4, DIV2, DIV1
SPI active x8	1.04	DIV8, DIV4, DIV2, DIV1

Link Down and Reset

When the link goes down or is disabled or upon hot reset, the PCIe Controller internally generates a link down reset, which clears all of its internal state machines, timers, and control registers. In the PCIe defined configuration register space, all registers, except those that are sticky, are also reset upon link down reset.

The PCIe Controller's AXI interface handles the link down reset as follows:

- When the PCIe Controller detects a link down reset, it seets the Link Down Indication Bit in the AXI register space.
- The AXI interface responds to the client with a `SLVERR` response while the Link Down Indication Bit is set.
- All write requests from the application (via the AXI slave interface) are consumed by the AXI interface and return a `SLVERR` response.

- All read requests are completed with a generated zero data pattern and return `SLVERR` response.

Additionally, the PCIe Controller aserrts the `LINK_DOWN_RESET_OUT` output signal upon a link down event. Your user application can monitor this signal to know when there is a link down event. For example, the client may have to reset its own FIFO buffers, registers, or state machines when the link is down. Firmware should clear the Link Down Indication Bit to restart any valid traffic after the negative edge of `LINK_DOWN_RESET_OUT`.

The AXI address translation registers are not cleared upon link down reset. These registers hold their programmed values and you do not need to re-program them.

Note: Refer to "AXI Configuration Registers" in the **TJ-Series PCIe Controller Registers User Guide** for the register descriptions.

Reset Types

There are three reset types: cold reset, warm reset, and hot reset. These resets cause link down conditions.

For warm and hot reset, the PCIe Controller, except sticky registers, goes into reset. The rest of the FPGA design remains operational. For a cold reset, the entire FPGA is reset.

Cold Reset

When the FPGA is reset, for example by power cycling, it triggers a cold reset for the PCIe Controller.

Warm Reset

When `PERST_N` is asserted, it triggers a warm reset for the PCIe Controller. The PCIe Controller, except sticky registers, undergoes a reset. The rest of the FPGA remains operational.

During a warm reset, the PCIe Controller issues a reset request to soft logic and IP cores connected to it. Your user application must observe the reset handshake, and return a relevant `RESET_ACK` to allow the PCIe Controller to complete the warm reset procedure.

Hot Reset

During a hot reset, the PCIe Controller, except sticky registers, undergoes a reset. The rest of the FPGA remains operational.

- In root port mode, the `HOT_RESET_IN` input initiates a hot reset sequence on the PCIe link. The root port application can assert and hold `HOT_RESET_IN` high until the `LINK_DOWN_RESET_OUT` output goes high.
- In endpoint mode, asserting `LINK_DOWN_RESET_OUT` triggers a hot reset.

When you trigger a hot reset, the PCIe Controller issues a reset request to soft logic and IP cores connected to it. Your user application must observe the reset handshake, and return a relevant `RESET_ACK` to allow the PCIe Controller to complete the hot reset procedure.

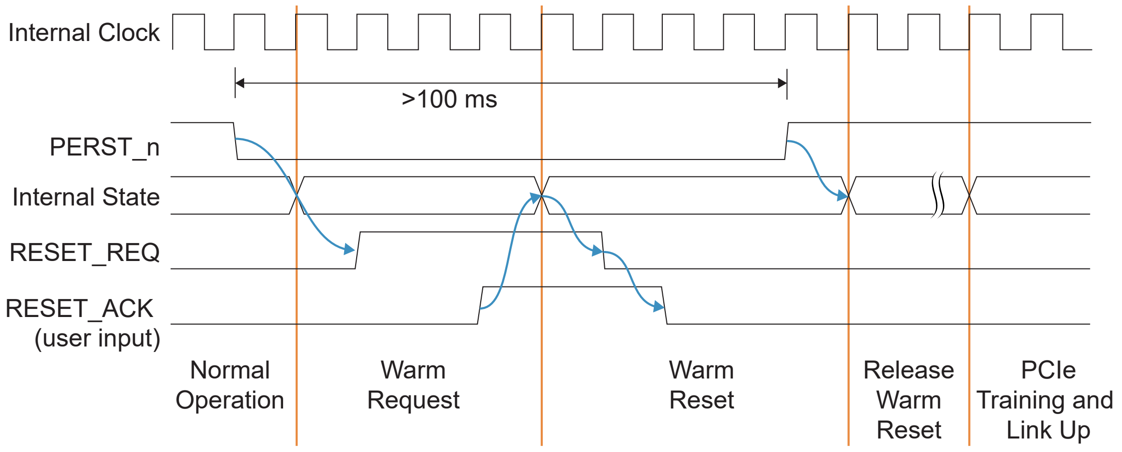
Reset Handshake

When a warm or hot reset occurs, the PCIe Controller sends a reset request by driving the `RESET_REQ` output high. When `RESET_REQ` goes high, the user application can delay the assertion of `RESET_ACK`, for example, to provide sufficient time to power down the application. However, the user application must indicate its readiness for reset by asserting `RESET_REQ`, which tells the PCIe Controller to proceed with the warm or hot reset.

Once the reset event is served, the PCIe Controller de-asserts `RESET_REQ`, at which point the user application must deassert `RESET_ACK`.

De-asserting `PERST_N` triggers the PCIe Controller's internal sequence to exit from reset and subsequently perform PCIe training and link up.

Figure 24: Reset Handshake



When `RESET_REQ` is asserted, the reset request is sent to all affected IP cores (including soft IP) and the core fabric.

- When the soft IP receives the reset request, it should evaluate its readiness and assert `RESET_ACK`.
- The user appliication should hold `RESET_ACK` asserted until the PCIe Controller de-asserts `RESET_REQ`.
- When `RESET_REQ` deasserts, the user application should deassert `RESET_ACK`.

When all affected soft IP cores assert `RESET_ACK`, the actual reset event occurs and the affected reset domains are reset.

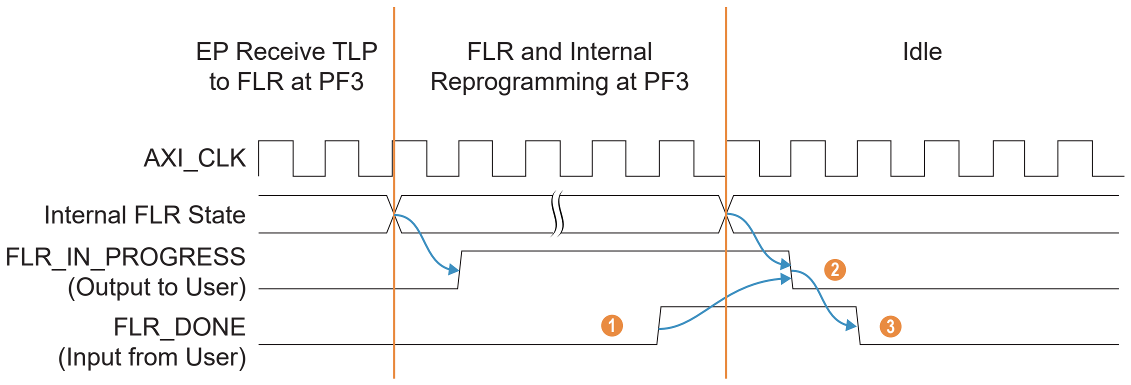
Function-Level Reset (FLR)

FLR enables the user application to reset a specific function on the endpoint; the FLR only affects the targeted function. When a specific VF is reset, only that VF's resources are reset. When a PF is reset, all of the PF's resources, including those of its associated VFs, are reset. The link state is not affected by an FLR.

Note: Refer to **Function-Level Reset Signals** on page 105 for a detailed description of the signals.

The following figure shows the handshake of `FLR_DONE` and `FLR_IN_PROGRESS` in PF3, and is applicable to any PFs/VFs undergoing FLR.

Figure 25: FLR Handshake


EP Receive TLP to FLR at PF3
FLR and Internal Reprogramming at PF3
Idle
AXI_CLK
Internal FLR State
FLR_IN_PROGRESS (Output to User)
FLR_DONE (Input from User)
1
2
3

1. User clears any pending transactions associated with the reset and then asserts FLR_DONE.
2. When internal PF3 reprogramming completes, the PCIe Controller de-asserts FLR_IN_PROGRESS.
3. De-assert FLR_DONE after FLR_IN_PROGRESS has finished de-asserting.

When the host sets the FLR bit in an endpoint function's device control register with a `CfgWr`, the PCIe Controller first responds with the completion. Then it initiates the FLR. The affected function's configuration registers are reset as described in the PCIe specification.

The PCIe Controller asserts the `FLR_IN_PROGRESS` and `VF_FLR_IN_PROGRESS` outputs to the user application, indicating the PFs and VFs that received a FLR.

When the PCIe Controller asserts `FLR_IN_PROGRESS`[*n*], the user application must clear any pending transactions associated with the function (PF/VF) being reset. Then, the user application must assert `FLR_DONE`[*n*] and hold `FLR_DONE`[*n*] high until the de-assertion of `FLR_IN_PROGRESS`[*n*]. At the same time, the associated PF/VF undergoes an internal re-programming and retrieves the original user configurations. Upon the assertion of `FLR_DONE`[*n*] from both ends (user and internal reprogramming), the PCIe Controller de-asserts `FLR_IN_PROGRESS`[*n*], at which time the user application must de-assert `FLR_DONE`[*n*].

Note: The client must complete the FLR within 100 ms, as required by the PCIe specification.

While the `FLR_IN_PROGRESS` and `VF_FLR_IN_PROGRESS` outputs are high, any configuration or memory or I/O requests the function receives are silently discarded as permitted by the PCIe specification.

Each function's Bus Master Enable bit in the Command Register is also reset upon FLR. The client can only restart outbound traffic after the host sets the Bus Master Enable bit through a `CfgWr`.

In the event of an FLR, requests on the AXI interface are handled as follows:

- In progress write requests complete normally. The client should not initiate any additional write requests from the function under FLR.
- Read requests from the affected function return `SLVERR` on `RRESP`.
- The AXI address translation registers are not affected by FLR for any function.

Concurrent FLR Request in Multiple PFs/VFs

If an FLR is triggered concurrently in multiple PFs/VFs, the PCIe Controller asserts the associated `FLR_IN_PROGRESS`[*n*].

For example, if an FLR is triggered concurrently in PF1 and PF3:

1. The PCIe Controller asserts `FLR_IN_PROGRESS[1]` and `FLR_IN_PROGRESS[1]`.
2. The user application must clear pending transactions associated with PF1 and PF3.
3. The user application must assert `FLR_DONE[1]` and `FLR_DONE[3]`, and hold them high until `FLR_IN_PROGRESS[1]` and `FLR_IN_PROGRESS[3]` de-assert, respectively.

Reset During an FLR

If a warm or hot reset occurs during an FLR, the reset takes precedence and *all* PFs undergo reset. In this event, the user application must monitor for the de-assertion of `FLR_IN_PROGRESS`[*n*], and de-assert `FLR_DONE`[*n*].

Power Management

The PCIe Controller supports several power management techniques, as described in the following topics. Refer to the PCIe specification mandated features and their usage in a PCI system.

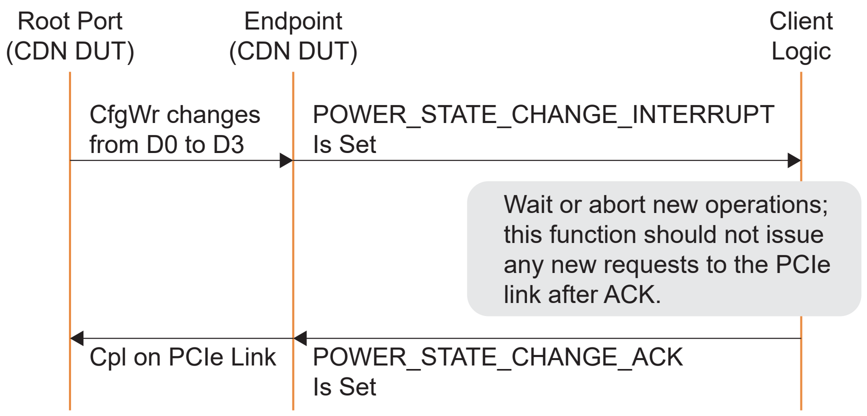
Function Power States

The PCIe Controller supports the PCI function power states D0 (uninitialized and active), D1, and D3hot.

By default, the PCIe Controller sets the Power Management Control Register's No Soft Reset bit for all enabled functions to 1. This setting means that the function's state is not lost when it is in the D3hot power state, and its registers do not need to be re-configured when the function is goes back to D0. The PCIe specifications recommend setting this bit for all functions.

When a root port changes an endpoint's power state form D0 to a non-D0 state, the PCIe Controller asserts `POWER_STATE_CHANGE_INTERRUPT`. Before asserting `POWER_STATE_CHANGE_ACK`, the client must ensure that no new request are issued to the PCIe link after the acknowledge is asserted. This functionality is required per the PCIe Specification.

Figure 26: Asserting POWER_STATE_CHANGE_INTERRUPT


Root Port (CDN DUT)
Endpoint (CDN DUT)
Client Logic
CfgWr changes from D0 to D3
POWER_STATE_CHANGE_INTERRUPT Is Set
Wait or abort new operations; this function should not issue any new requests to the PCIe link after ACK.
Cpl on PCIe Link
POWER_STATE_CHANGE_ACK Is Set

L0s Power State

L0s entry and exit is an autonomous process, and has very low exit latency compared to the other link power states. In this state, the PCIe Controller automatically initiates entry into ASPM L0s if the TX is idle (i.e., no TLPs and no DLLPs to be transmitted) for a programmable time period. For the transition to occur:

- *Endpoints*—ASPM L0s must be enabled in the Link Control Register of the configuration spaces of all enabled functions.
- *Root port*—ASPM L0s must be enabled in the Link Control Register of the PCIe Controller root port register set.

Note: You can enable L0s in the Interface Designer (**PCI Express block** > **Pins tab** > **Power Management sub-tab** > **Enable Power Management**).

During operation, you can update the setting using the APB interface. Enable or disable active-state power management (ASPM) L0s by setting the Active State Power Management Control (bit [0]) in the Link Control and Status Register.

Important: You cannot enable active state power management (ASPM) if SRIS is enabled.

You can program the L0s entry timeout using the L0s Timeout Limit Register local management register. The transition from L0 to L0s happens after a time period programmed in the L0s Timeout Limit Register elapses with no TLP or DLLP being transmitted. Setting the L0s Timeout Limit Register to 0 disables the transition to L0s state.

L1 Power State

The L1 link power state savess more power at the expense of more latency compared to the L0s state.

Entering L1 via ASPM

ASPM L1 is an autonomous process; entry and exit happens without any user handshaking. You can enable or disable ASPM L1 by setting the Active State Power Management Control (bit [1]) in the Link Control and Status Register. The PCIe Controller automatically initiates entry into ASPM L1 if the TX side is idle (i.e., no TLPs from the client and no replay TLPs pending) for a programmable time period. SPM L1 entry operates as follows:

1. When the link TX is idle, the endpoint PCIe Controller begins incrementing the ASPM L1 entry timer internally. If the client requests to transmit a TLP, the timer is immediately cleared.
2. When the ASPM L1 entry timer reaches the programmed value in the ASPM L1 Entry Timeout Delay Register, the PCIe Controller checks whether sufficient credits are accumulated.
3. The PCIe Controller blocks new TLPs and initiates ASPM L1 entry by sending PM_Active_State_Request_L1 DLLPs to its transmit lanes.
4. The PCIe Controller continuously transmits the PM_Active_State_Request_L1 DLLP until it receives a response from the upstream device.
5. The upstream device must immediately respond to the request with an acceptance (PM_Request_Ack) or rejection (PM_Active_State_Nak).
6. If the upstream device rejects with a 'PM_Active_State_Nak message, the PCIe Controller aborts the ASPM L1 entry and continues to send TLPs normally.
7. If the upstream device accepts with a PM_Request_Ack message, the PCIe Controller puts its TX into electrical idle and enters ASPM L1.
8. The upstream device detects the electrical idle and puts its TX into electrical idle as well.

For the transition to ASPM L1 to occur:

- *Endpoints*—You must enable ASPM L1 in the Link Control Register of all the enabled functions.
- *Endpoints*—You must enable the L1 power state in the Link Control Register of the PCIe Controller's root port register set.

ASPM L1 entry timeout is programmable through the ASPM L1 Entry Timeout Delay Register local management register. The L1 Timeout[19:0] field contains the timeout value (in 16 ns units) for transitioning to the ASPM L1 power state. Setting it to 0 disables the transition to the ASPM L1 power state.

Entering L1 via PCI-PM

When the PCIe Controller is configured as an endpoint, the PCI power management operation is as follows:

1. The PCIe Controller operates normally with all functions in the D0 active state.

2. The remote root port writes to the Power Management Control Register for all enabled functions, which transitions the function(s) to the non-D0 power state.
3. When all functions are in the non-D0 state, the PCIe Controller initiates a link power state transition to L1 by transmitting PM_Enter_L1 DLLPs.
4. After the data link layer handshake, the link transitions to the L1 state.

While the link is in L1 state and the PCIe Controller's functions are in D1 or D3hot, the link partner can transition the link from L1 to L0 at any time. The PCIe Controller can then optionally initiate a re-entry back to L1 if the link has been idle for a set interval and the PCIe Controllers functions are still in D3hot. The re-entry to L1 is controlled by the delay programmed in the L1 State Re-entry Delay Register in the local management space. Setting this register to a non-zero value causes the PCIe Controller to initiate entry back to L1 when a delay equal to the number of clock cycles programmed in this register has elapsed with no link activity. Setting this register to 0 prevents re-entry to L1. The initial transition to L1 (step 3 above) is not affected by this register setting.

L1 Exit Triggers

The following events trigger an L1 exit:

- Electrical idle exit detection.
- New requests at the AXI interface.
- Assertion of side-band signal `CLIENT_REQ_EXIT_L1`.
- The root port initiates the retraining request using the link control register).

ASPM Exit

Any L1 exit triggers an L1 exit process and there is no handshake required from the client.

PCI-PM Exit

Any L1 exit triggers change the link power state to L0. However, an additional root port to endpoint handshake is required for normal operation after reaching L0.

For a root port initiated PM L1 exit:

1. Root port initiates a configuration write to change the endpoint function's power state to D0.
2. This configuration write triggers an L1 exit on the link.
3. The endpoint device also exits from L1 and responds with CPL.
4. Normal data transfer can happen on the PCIe link.

For an endpoint initiated PM L1 exit:

1. The endpoint initiates a PM_PME message to the root port to request a power state change.
2. The message triggers an L1 exit on the link.
3. After receiving this message, the root port initiates a configuration write to change the endpoint function's power state to D0.
4. The endpoint device responds with CPL.
5. Normal data transfer can happen on the PCIe link.

L1 Register Programming

The following registers contain information about the L1 state:

- *Entering L1*—The Low Power Debug and Control Register 0's L1 Entry Mode field in the local management space gives information about the last L1 entry mode. You can determine whether PM or ASPM was used to enter L1. This field is reset before each L1 entry operation.
- *Exit trigger for L1 (or L1 substate)*—The Low Power Debug and Control Register 0's L1, L1.x Exit Reason field in the local management space gives information about the last L1 or L1 substate exit triggers. This field is reset before each L1 entry operation.

Blocking L1 Explicit Client Exit or Endpoint Entry

The client can trigger an explicit L1 exit by asserting `CLIENT_REQ_EXIT_L1`. This signal triggers an exit to L0 from L1 or L1-substates. A new request at the AXI interface also triggers an L1 exit event internally.

You can also use this signal to block L1 entry in when the PCIe Controller is in endpoint mode. Blocking L1 entry is useful when a client wants to disable L1 entry to reduce latency for its outstanding operations.

Refer to **L1 Interface Signals** on page 109 for the signal description.

Figure 27: Exiting from L1 with CLIENT_REQ_L1_EXIT

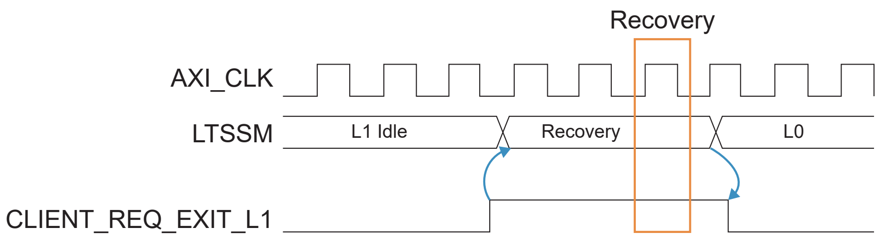
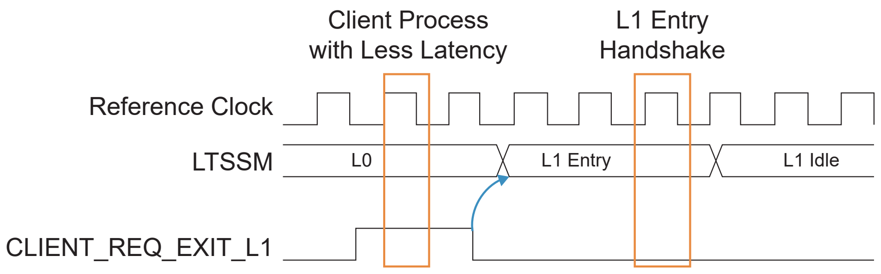


Figure 28: Blocking L1 Entry with CLIENT_REQ_L1_EXIT



L1 Power Substates

The PCI-SIG L1 Power Management Substates ECN defines an optional mechanism to reduce idle power in the L1 link state by defining L1 substates to facilitate removing power from the PHY, and clocks to the PCIe Controller. The L1 PM substates are enabled when the link enters L1 due to PCI power management or ASPM.

There are two L1 substates:

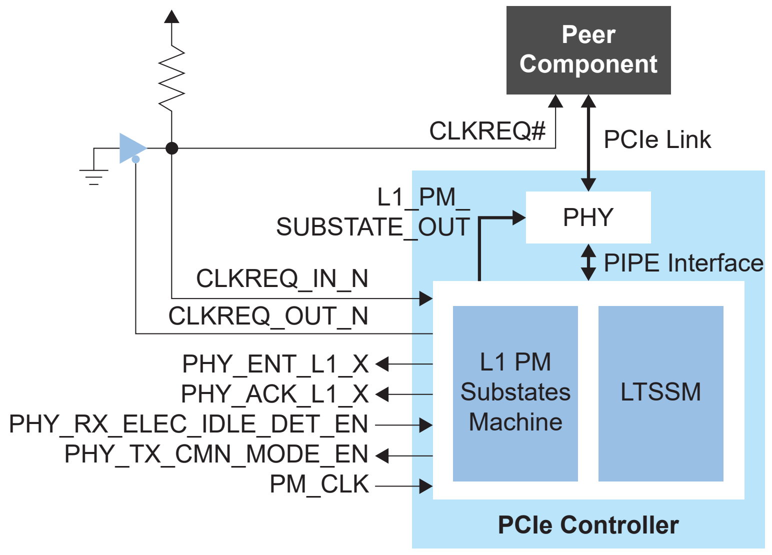
- The L1.1 substate allows you to turn off clocks and most of the PHY power, but it requires the PHY to maintain common-mode voltages on the TX side.
- The L1.2 substate enables further reduction in idle power by not requiring common-mode voltages to be maintained.

Both L1.1 and L1.2 states allow you to turn off the PHY's electrical idle detection circuitry.

The L1 PM substates use the `CLKREQ#` sideband signal to control the clocks. The `CLKREQ#` signal is an open-drain, active-low signal shared by the upstream and downstream ports, and either side can assert it by driving it low. This signal enables the clock generator. The core clock is turned off when both sides de-assert their `CLKREQ#` outputs.

The PCIe Controller has the `CLKREQ_IN_N` input and a `CLKREQ_OUT_N` output to implement the tri-state `CLKREQ#` pin. The `CLKREQ_OUT_N` output, when low, enables the tri-state driver driving the `CLKREQ#` pin, causing assertion of the shared signal. The port on the other side can also assert CLKREQ# by driving it low. The PCIe Controller monitors the state of this shared signal through the CLKREQ_IN_N input, as shown in the following figure.

Figure 29: L1 PM Substates Block Diagram



Because the core clock is turned off in the L1.1 and L1.2 substates, a separate power management clock (PM_CLK) drives the L1 PM substates state machine. This clock must always be active, regardless of the link's power state. There is no requirement on the relative phase of this clock with respect to the other PCIe Controller clocks.

The L1 PM substates state machine also provides the handshake signals PHY_ENT_L1_X and PHY_ACK_L1_X to prepare the local PHY for the removal of the reference clock. The state machine asserts the PHY_ENT_L1_X output in the L1.0 substate when it has determined that the conditions for transition to the L1.1 or L1.2 substates are met. It then waits for the PHY to assert PHY_ACK_L1_X before de-asserting CLKREQ_OUT_N and entering L1.1 or L1.2 substates. During L1.1 or L1.2 exit, the PCIe Controller de-asserts PHY_ENT_L1_X to the PHY and waits for the corresponding de-assertion of PHY_ACK_L1_X before transitioning to the L1.0 state. This step is required to ensure that the PHY is fully operational and the clocks are stable before entering L1.0. For the case of L1.2, the PHY handshake is performed while in the L1.2 exit substate.

Note: De-asserting PHY_ENT_L1_X changes the PHY state from the L1 substate to L1.0. Make sure that the PHY has a stable reference clock during the exit process.

The L1 PM substates state machine provides an output signal PHY_RX_ELEC_IDLE_DET_EN to inform the PHY when to enable its electrical idle detection circuits on the RX side. The PCIe Controller asserts this output in all states except when the L1 PM substates state machine is in the L1.1, L1.2.Entry, and L1.2.Idle substates.

The L1 PM substates state machine also provides an output signal PHY_TX_CMN_MODE_EN to enable common mode on the PHY TX. The PCIe Controller de-asserts this output when the L1 PM substates state machine is in the L2.Idle substate, and asserts it at all other times.

Entering L1 Substate

L1 substate entry is initiated when the link is in L1 and CLKREQ# from the upstream and downstream components are de-asserted. The PCIe Controller enters L1.1 or L1.2 depending on which substate is enabled in the L1 PM Substate Control registers.

CLKREQ_OUT_N is de-asserted in both L1 substates. When the remote device also de-asserts CLKREQ#, the core clock is turned off by the clock controller in the user domain.

Exiting L1 Substate

Either side can initiate a transition out of the L1 substate. The remote side initiates an L1 PM substate exit by asserting its CLKREQ# output. This assertion turns on the core clock and asserts the PCIe Controller's `CLKREQ_IN_N` input, causing its L1 PM substate to change to L1.0 and enabling the transition of the LTSSM from L1 into recovery. The client can also initiate the L1 exit.

The following events trigger an L1 substate exit:

- Remote device initiated exit triggers:
  - Assert `CLKREQ#`.
  - Detect an electrical idle exit (only during entry into L1.1 or L1.2 substates before `PHY_RX_ELEC_IDLE_DET_EN` is deasserted).
- Locally initiated exit triggers:
  - New requests at the AXI interface.
  - New register access requests.
  - Assertion of sideband signals `CLIENT_REQ_EXIT_L1_SUBSTATE` or `CLIENT_REQ_EXIT_L1`.

The L1 substate exit triggers change the link from L1.1 or L1.2 state to L1.0 and then to L0.

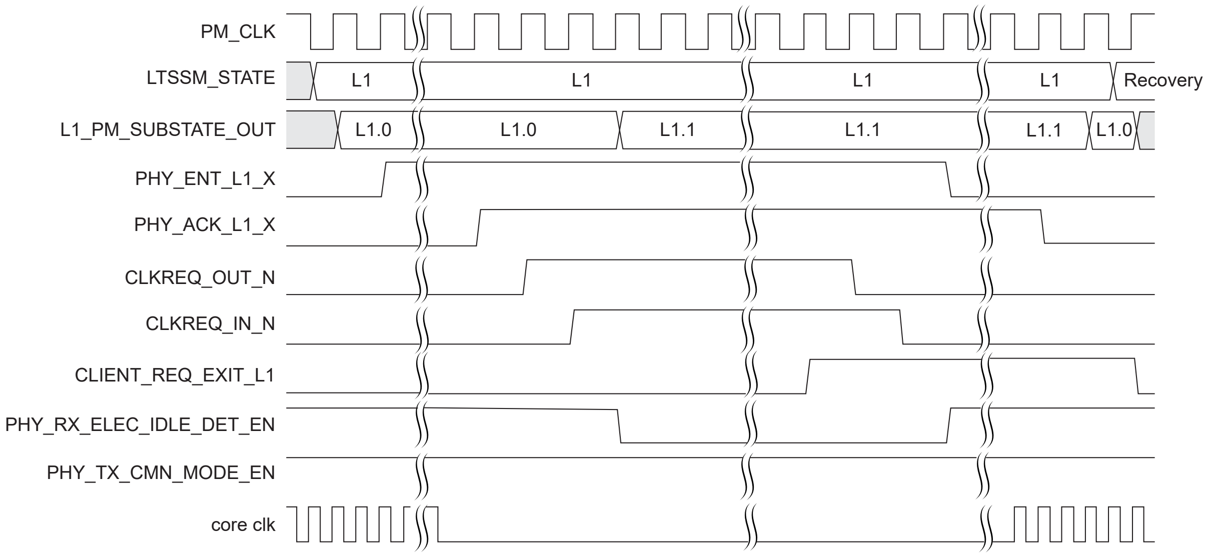
L1.1 Operation

The PCIe Controller enters L1.1 when L1.1 entry conditions are true and L1.2 entry conditions are false. The following diagram illustrates L1.1 entry and locally initiated exit process. `CLIENT_REQ_EXIT_L1` represents all local exit triggers.

Download: You can enable L1.1 in the Interface Designer (**PCI Express block** > **Pins tab** > **Power Management sub-tab** > **PM L1.1 Substate Enable**).

It is possible to enable/disable L1.1 with the APB interface, however, you **must** follow the rules described in **L1 Substate Register Programming** on page 75 for the power transitions to work correctly.

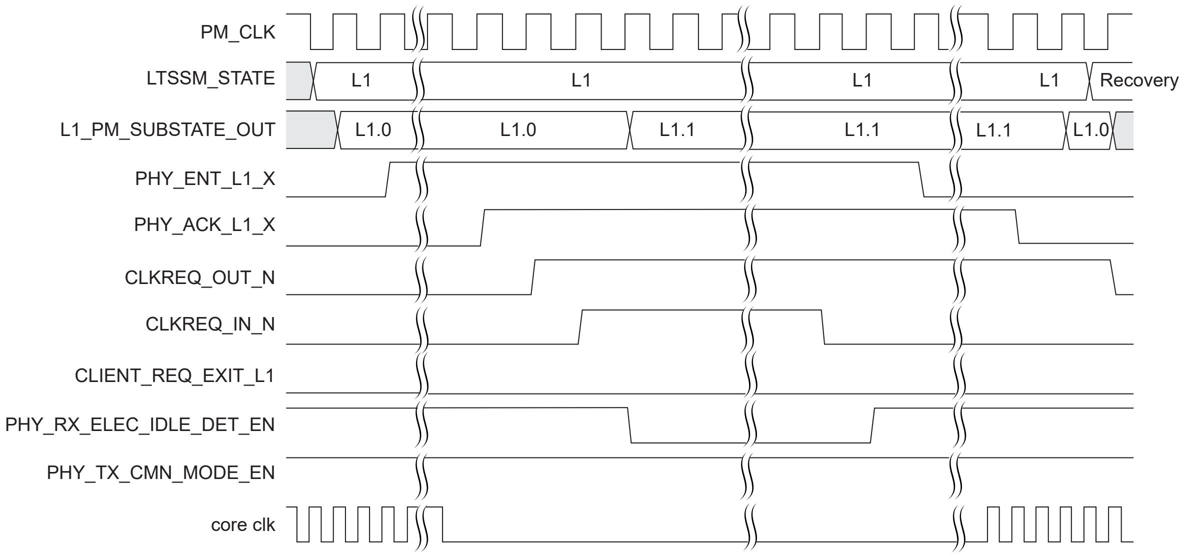
Figure 30: L1.1 Entry and Locally Initiated Exit



When the conditions for entering the L1.1 substate are met, the L1 PM substates state machine first performs a handshake with the PHY using the `PHY_ENT_L1_X` and `PHY_ACK_L1_X` signals to prepare the PHY for the removal of the reference clock. Once the PHY has asserted `PHY_ACK_L1_X`, the PCIe Controller de-asserts `CLKREQ_OUT_N`. If the link partner also de-asserts its `CLKREQ#` output, the core clock becomes inactive and the PCIe Controller's `CLKREQ_IN_N` input is de-asserted. The L1 PM substates state machine transitions to L1.1 when `CLKREQ_IN_N` goes high.

Any local L1 substate exit triggers bring the PCIe Controller back to the L1.0 state. During exit, the PCIe Controller asserts `CLKREQ_OUT_N` to turn on the core clock, thereby asserting `CLKREQ_IN_N`. When `CLKREQ_IN_N` goes low, the L1 PM substates state machine performs another handshake with the PHY by de-asserting `PHY_ENT_L1_X` and waiting for the PHY to respond by de-asserting `PHY_ACK_L1_X`. This handshake is necessary to prepare the PHY for the re-activation of the reference clock. Once this handshake has been completed, the PHY transitions back to the L1.0 substate.

Figure 31: L1.1 Entry and Exit Initiated by Link Partner



The previous figure illustrates the operation when the link partner initiates the exit from L1. The PCIe Controller enters L1.1 from L1.0 when the entry conditions for L1.2 are not satisfied and the entry conditions for L1.1 are satisfied. After completing the `PHY_ENT_L1_X`/ `PHY_ACK_L1_X` handshake with the PHY, the PCIe Controller de-asserts `CLKREQ_OUT_N`. If the link partner also de-asserts its `CLKREQ#` output, the core clock becomes inactive and the PCIe Controller's `CLKREQ_IN_N` input is de-asserted, causing the L1 PM substates state machine to enter the L1.1 state.

The link partner initiates the transition of the link from the L1 state by asserting its `CLKREQ#` outputt, resulting in the assertion the PCIe Controller's `CLKREQ_IN_N` input. When `CLKREQ_IN_N` goes low, the L1 PM substates state machine prepares the PHY for exit from L1.1 by de-asserting `PHY_ENT_L1_X` and waiting for the PHY to de-assert `PHY_ACK_L1_X`. When this handshake is completed, the L1 PM substates state machine transitions to the L1.0 substate. Meanwhile, the assertion of `CLKREQ#` results in the core clock becoming active, which enables the LTSSM to move out of L1 into recovery.

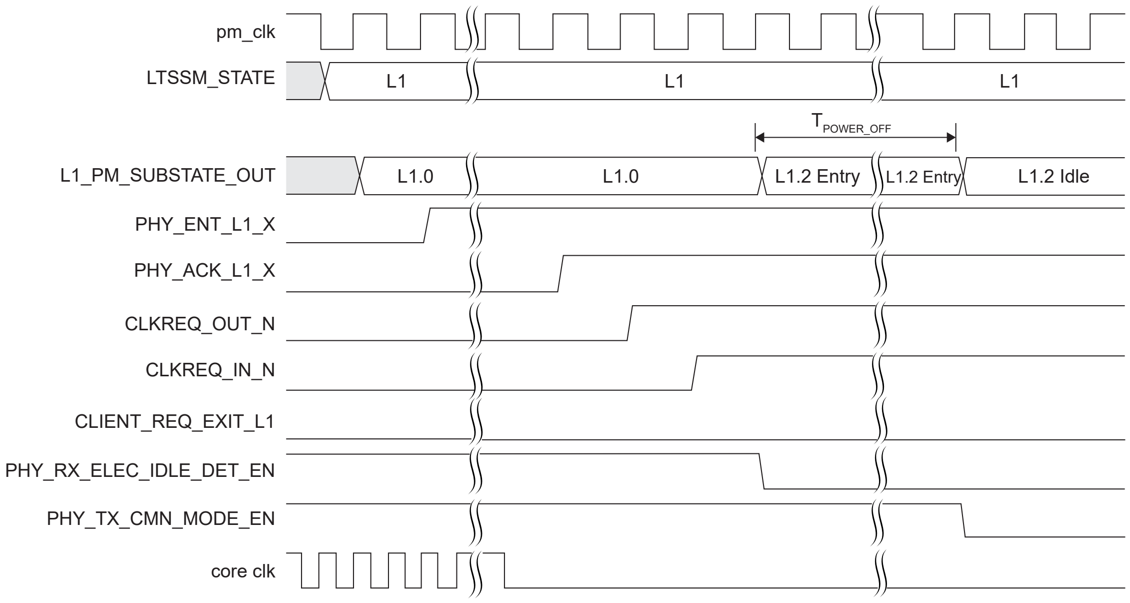
L1.2 Operation

The following figure illustrates the sequence for the L1 PM substates state machine to enter the L1.2 substate.

Download: You can enable L1.1 in the Interface Designer (**PCI Express block** > **Pins tab** > **Power Management sub-tab** > **PM L1.2 Substate Enable**).

It is possible to enable/disable L1.2 with the APB interface, however, you **must** follow the rules described in **L1 Substate Register Programming** on page 75 for the power transitions to work correctly.

Figure 32: L1.2 Substate Entry

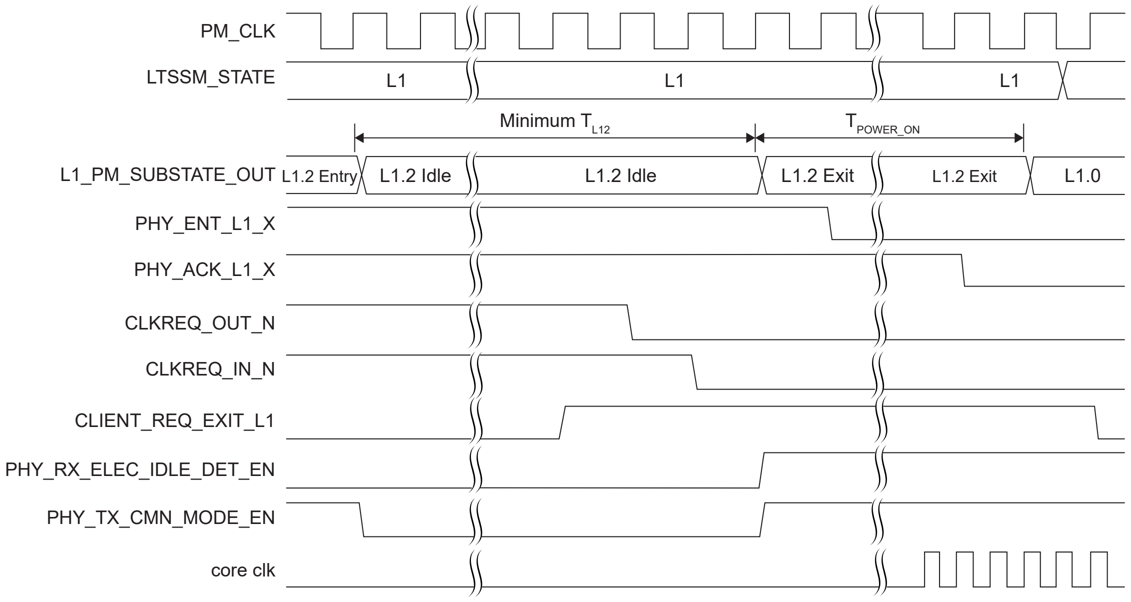


If the entry conditions for L1.2 are satisfied, it first performs the handshake with the PHY using the `PHY_ENT_L1_X` and `PHY_ACK_L1_X` signals to prepare the PHY for the removal of the reference clock. Once the PHY has asserted `PHY_ACK_L1_X`, the PCIe Controller de-asserts `CLKREQ_OUT_N`. If the link partner also de-asserts its `CLKREQ#` output, the core clock becomes inactive and the PCIe Controller's `CLKREQ_IN_N` input is de-asserted. The L1 PM substates state machine transitions to L1.2.Entry when the `CLKREQ_IN_N` input goes high. While the L1 PM substates state machine is in the L1.2.Entry substate, it monitors the `CLKREQ_IN_N` input and transitions back to L1.0 if it is asserted. If `CLKREQ_IN_N` remains de-asserted, the state machine stays in the L1.2.Entry substate for an interval T_{POWER_OFF} (2 ms) and then transitions to L1.2.Idle.

When the L1 PM substates state machine is in L1.2.Idle, the client or the link partner can initiate a transition of the link out of the L1-substate. The following figure shows the operation of the

L1.2 substates when there is an exit trigger from the client. `CLIENT_REQ_EXIT_L1` represents all local exit triggers.

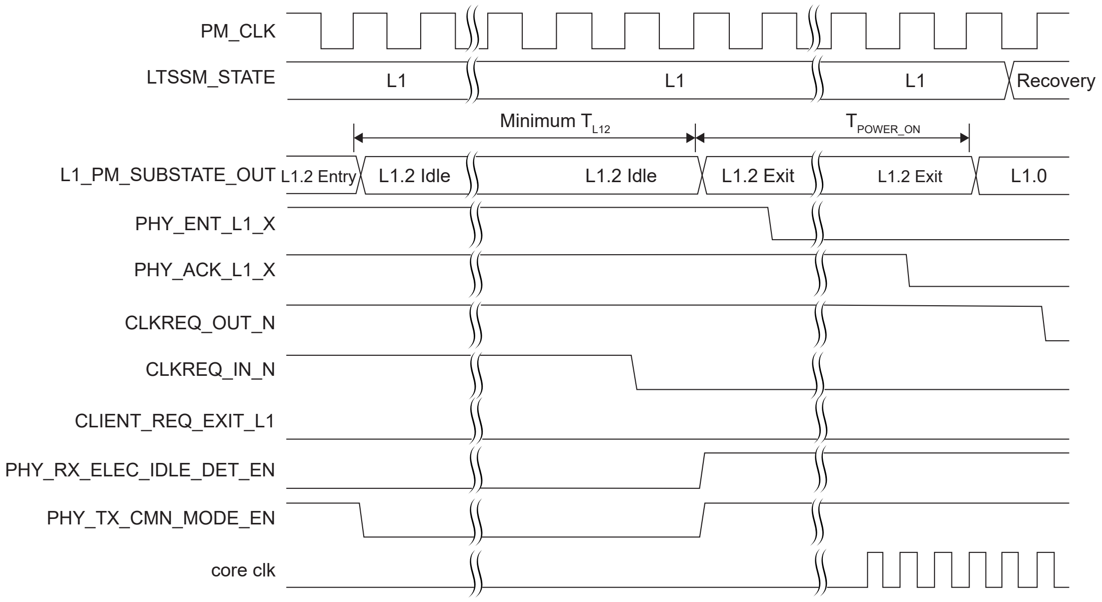
Figure 33: L1.2 Substate Locally Initiated Exit



Any one of the local exit triggers initiates the L1.2 exit process. The PCIe Controller first asserts `CLKREQ_OUT_N` to turn on the core clock, resulting in `CLKREQ_IN_N` becoming asserted. When `CLKREQ_IN_N` goes low, the L1 PM substates state machine transitions to L1.2.Exit.

While in L1.2.Exit, the L1 PM substates state machine performs the `PHY_ENT_L1_X`/ `PHY_ACK_L1_X` handshake with the PHY to prepare the PHY for the re-introduction of the clocks, and subsequently transitions back to L1.0. The L1 PM substates state machine must stay in L1.2.Exit for a minimum interval of T_{POWER_ON}. The duration of this interval is determined by the setting of the T_{POWER_ON} value and scale parameters in the L1 PM Substates Control 2 Register. The interval can vary from 0 to 3,100 ms.

Figure 34: L1.2 Substate Exit Initiated by Link Partner



The previous figure illustrates the operation when the link partner initiates the exit from L1. When in L1.2.Idle, the link partner initiates the transition of the link from L1 by asserting its `CLKREQ#` output, resulting in the assertion of the PCIe Controller's `CLKREQ_IN_N` input. When `CLKREQ_IN_N` goes low, the L1 PM substates state machine transitions to L1.2.Exit (after satisfying the minimum 4 μs stay in L1.2.Idle). After completing the handshake with the PHY for re-enabling its clocks and staying in L1.2.Exit for a minimum interval of T_{POWER_ON}, the L1 PM substates state machine then transitions to L1.0.

L1 Substate Register Programming

The following table provides guidance to program specification-defined registers. Refer to the **TJ-Series PCIe Registers User Guide** for the full list of registers. Registers with prefix *Port* and capability registers are read-only when accessed from the PCIe link. These registers are writable through the local management interface. You need to initialize these registers to match PHY and system electrical characteristics. These registers are used by the standard system initialization software to program RW control registers in the endpoint and root port L1 substate capability space.

Table 41: Root Port Inbound PCIe to AXI Address Translation Registers for 1 BAR

Register Name	Guide to Select the Value	Who Updates	Issues with Wrong Values
Port T_POWER_ON Value and Scale	Port's PHY T_POWER_ON value. For example, TP1.2_to_P1 value in the PHY.	Client's controller and PHY initialization firmware	PCIe system initialization software uses this value to program the T_power_on register.
T_POWER_ON Value and Scale	Maximum of (endpoint Port_T_power_on, root port Port_T_power_on).	Host's PCIe system initialization software	If the value is lower than required, one device drives into an unpowered remote device, which can result in a link down event. The LTSSM moves to detect the state.
Port Common Mode Restore Time	Time required for the port's PHY to establish common mode actively during the transmission of TS1s. For example, in the PHY this value is t_common_mode.	Local client's controller and PHY initialization firmware.	Initialization software uses this value to program the Common Mode Restore Time register.
Common Mode restore Time	Maximum of (root port port common mode restore time, endpoint port common mode restore time).	PCIe system initialization software	If the value is lower than required, one device drives into an unpowered remote device, which can result in a link down event. The LTSSM moves to detect the state.
LTR_L1_2 _threshold Value and Scale	This register is used if LTR and ASPM L1.2 are enabled. It is the worst-case latency a request would face to get completed in the presence of L1.2. Refer the LTR section of the PCIe specification for the LTR usage. An example calculation is: LTR_L1_2_threshold = <*service request latency*> + 2 * (TL1.2 + TL1O_REFCLK_ON + TP1_to_P0 + TCOMMONMODE + 2 µs) Where: <*service request latency*> is the worst case delay for the root port to respond to a read request or to accept a WR request; *TL1.2* is the minimum time to stay in L1.2 (4 µs) *TL1O_REFCLK_ON* is the CLKREQ# assertion to reference clock valid when exiting L1.2. This is TPOWER_ON + any additional time the system needs to enable the reference clock. *TP1_to_P0* is the time the PHY needs to change the power state from P1 to P0. *TCOMMONMODE* is the same as the Port Common Mode Restore Time Register. The accountable margin for latency in the PCIe Controllerand handshake is 2 µs. The equation multiplies by two to account for the L1.2 exit that a request and/or completion might require.	PCIe system initialization software	ASPM L1.2 entry happens only when the endpoint's LTR requirement is larger than this threshold. Incorrect programming of the threshold can cause unexpected latency for the endpoint requests.

Delayed Entry

You can delay entry to L1 substates with the Low Power Debug and Control Register 0/ L1 Substate Entry Delay field in the local management space. The PCIe Controller responds to an L1 exit event while waiting for this delay to expire.

Wait for Outstanding Completions before Entry

The PCIe Controller can wait for outstanding completions by using the Low Power Debug and Control Register 1/ Enable Outstanding CPL Check field in the local management space. The PCIe Controller waits for outstanding packets from the client and PCIe link. The PCIe Controller can respond to normal L1 exit triggers while waiting for outstanding completions. This field is normally not required unless you wants to fine tune latency for completions.

Wait for Empty Receive Buffers before Entry

The PCIe Controller can wait for receive buffers to be empty before entering an L1 substate. You use the Low Power Debug and Control Register 1/ Enable RX Path Check field in the local management space. The PCIe Controller can respond to normal L1 exit triggers while waiting for receive buffers. This field is normally not required unless you wants to fine tune latency for incoming packets from the PCIe link. You do not have to set this field because packets resume draining from receive buffers once the clocks resume during an L1 substsate exit.

Prevent Exit During Register Access

The core clock turns off during L1 substates. The register interface needs the core clock to respond to register access requests. Therefore, an L1- substate exit is triggered by default while accessing registers. You can change this behavior with the Low Power Debug and Control Register 1/ Disable Autonomous L1.x Exit upon Reg Access field in the local management space. This field is mostly helpful for debugging.

Explicit Client Exit or Entry Block

The client can trigger an explicit L1 substate exit by asserting `CLIENT_REQ_EXIT_L1_SUBSTATE`. This signal triggers an exit from L1 substates to L0 if the PCIe Controller is already in an L1 substate. The PCIe Controller waits in L1 for this signal to become de-asserted before entering an L1 substate. The PCIe Controller responds to normal L1 exit triggers while it waits for de-assertion .

You need to use this signal if the L1 substate shuts off clocks that drive new requests to the AXI interface. Elitestek highly recommends that you implement client hardware logic to assert this signal before initiating a new TLP or register access from the client. Hardware logic reduces overhead on firmware or software drivers. This signal is also useful if you want to block L1 substate entry to reduce latency for any pending outstanding operations.

Refer to **Table 74: L1 Substrate Interface** on page 109 for the signal description.

The following figures explain the use of this signal. L1.x means L1.1 or L1.2.

Figure 35: L1.2 Using CLIENT_REQ_L1_EXIT_SUBSTATE to exit from L1.x

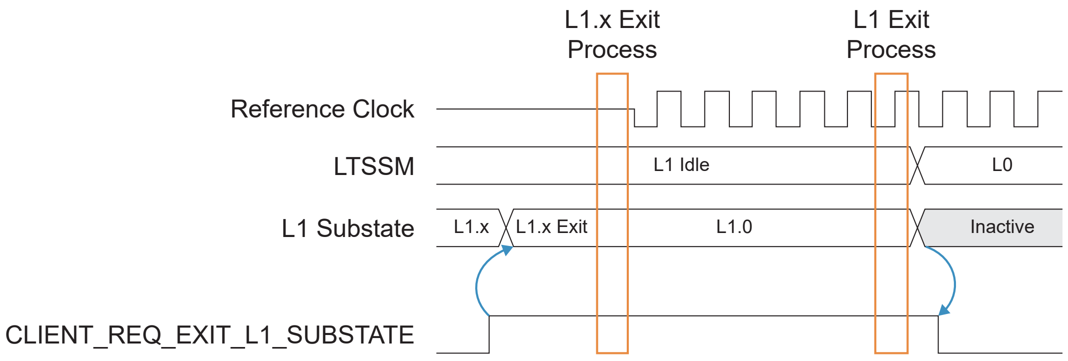
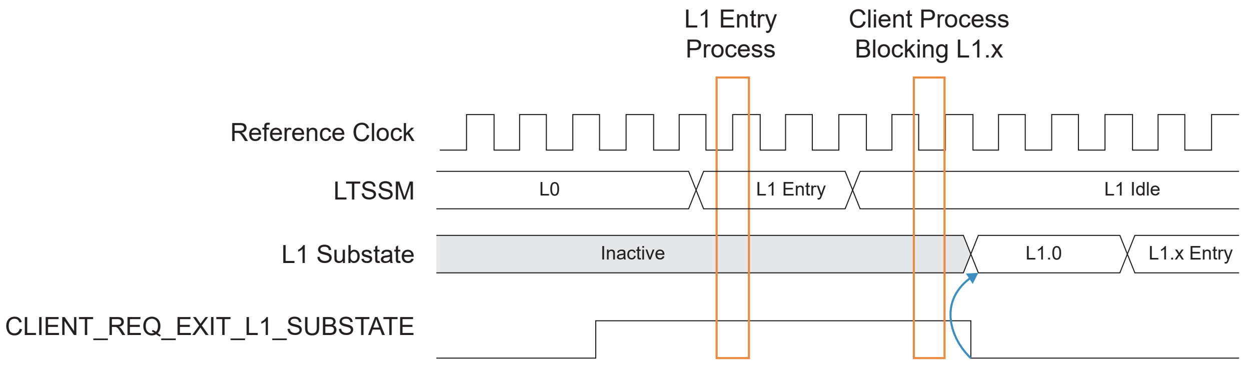


Figure 36: Using CLIENT_REQ_L1_EXIT_SUBSTATE to block L1.x entry



Integration Details

You can turn off the core clock in an L1 substate. To operate, the L1 substate requires `PM_CLK`. Client firmware needs to program `PM_CLK`. Use the Frequency Register/PM_CLK Frequency Select field in the local management register to change the `PM_CLK` frequency. You can only change `PM_CLK` frequency when LTSSM is not in L1.

By default, if there is a register access when the PCIe Controller is in the L1 substate, the PCIe Controller exits the L1 substate and responds to the register access request. The PCIe Controller moves to L0 and after servicing the register access request it goes back into the L1 substate. If the user firmware is performing polling and does not want this behavior, you can disable this feature by using the Low Power Debug and Control Register 1/ Disable Autonomous L1.x Exit upon Reg Access field in the local management space. If this field is set, the PCIe Controller gives an APB error response upon register access when it is in an L1 substate. The only reason for an error response for valid addresses at the APB interface is if a clock is not available. An error response is only available with the APB interface.

L2 Power State

For the PCIe Controller, L2 is a power saving state. It is a pseudo-L2 state because you cannot completely remove power from the PCIe Controller. You can suppress transition to the L2 state by holding `REQ_PM_TRANSITION_L23_READY` low.

Entering L2

Entering L2 from a non-D0 state is cleaner and the PCIe Controller can automate the handshake process with the host. The PCIe specification has provisions to enter L2 from D0. D0 is a normal operating state. L2 entry while a function's power state is D0 requires the client to respond to the host.

The following steps illustrates L2 entry from all allowed function power states.

1. The remote root port sends a PME_Turn_Off message to the PCIe Controller.
2. The PCIe Controller delivers the PME_Turn_Off message to the client through the AXI message interface (for AXI configurations) or target-request interface (for non-AXI configurations).
3. When ready, the client transmits the PME_TO_Ack message to the root port via the client master interface with the following steps:
   a. Wait for the client target request interface to receive a PME_Turn_Off message.
   b. Read the function's power state from the Power Management Control/Status Register configuration register.
   c. Check the programmed value of PME Turnoff Ack Delay[15:0] in the local management register space.
   d. If the Function Power State == 'D0' or if the PME Turnoff Ack Delay == 0x0000, the PCIe Controller does not transmit the PME_TO_Ack message (see following note).
      - Client firmware should ensure that there are no PCIe transfers active in the PCIe subsystem.
      - Client sends a PME_TO_ACK message over the client master request interface.
   e. If the condition in step (d) is not true, the PCIe Controller automatically transmits a PME_TO_Ack message after the PME Turnoff Ack Delay time. The client must not send PME_TO_ACK.
4. Optionally, the client can now change the PCIe Controller's power state to L23_Ready by asserting `REQ_PM_TRANSITION_L23_READY`. This assertion causes the PCIe Controller's LTSSM to transition to L2 and enables the client to power down the PCIe Controller completely. The client has to assert `REQ_PM_TRANSITION_L23_READY` until the LTSSM moves into L2.

Note: If any enabled PF is in the D0 power state, there may be PCIe transfers outstanding in the system for that PF. In this case, the PCIe Controller does not automatically transmit PME_TO_Ack.

Wake Up or Exiting L2

The PCIe Controller supports systems that use a wake-up mechanism. The client shall capture the PCIe Controller's PME context before sending the PME_TurnOff Acknowledge message to the root port. The client must store the PCIe Controllers bus number and device number from local management before acknowledging PME_TurnOff. The client must specify the Requester ID of the PME message sent when power is re-applied to the PCIe Controller and the link reaches L0. The following sequence describes the process of supporting `WAKE#`.

1. Assume the PCIe Controller is in L2. The client is maintaining the PME context and requester ID using a client that is powered by V_{AUX} or full power.
2. The client can decide to wake up the PCIe Controller. The client drives the `WAKE#` out-of-band signal to tell the power management controller that the PCIe Controller requires power to be re-applied.

   Note: The `WAKE#`signal is external to the PCIe Controller and is not used by it.
3. Assert `PERST_N` to bring the PCIe Controller into warm reset. Then, de-assert `PERST_N` to reapply the internal power in the PCIe Controller. Follow the reset sequence shown in **Figure 24: Reset Handshake** on page 64.
4. The client restores any PME context to the PCIe Controller registers via the local management interface.
5. Prevent the PCIe Controller from transmitting the PM_PME message automatically by programming bit [20] (Disable PME Message on PM Status) to 1 in the Local Management Register PME Service Timeout Delay Register.

   Note: Set the Disable PME Message on PM Status bit to 1 before setting the PME Status bit to 1.
6. The client sends a PM_PME message over the client interface using the requester ID that was captured before entering L2.

7. The root complex can perform a configuration write to the PCIe Controller to move the device from the D3hot state.

Configuring Registers with the APB Interface

The PCIe Controller has a 32-bit APB bus, which is accessible by the user application. The protocol is per the APB v1.0 specifications.

You configure the PCIe Controller's interfaces and features with the Efinity Interface Designer, which sets the corresponding bits in the configuration registers. The PCIe Controller uses these settings during power up or cold reset. If you want to change the setting during operation, you can set the register bits through the APB interface.

You enable the APB interface in the Interface Designer (**PCI Express block** > **Pins tab** > **APB sub-tab** > **Enable Advanced Peripheral Bus**).

Figure 37: Configuration and Management Registers

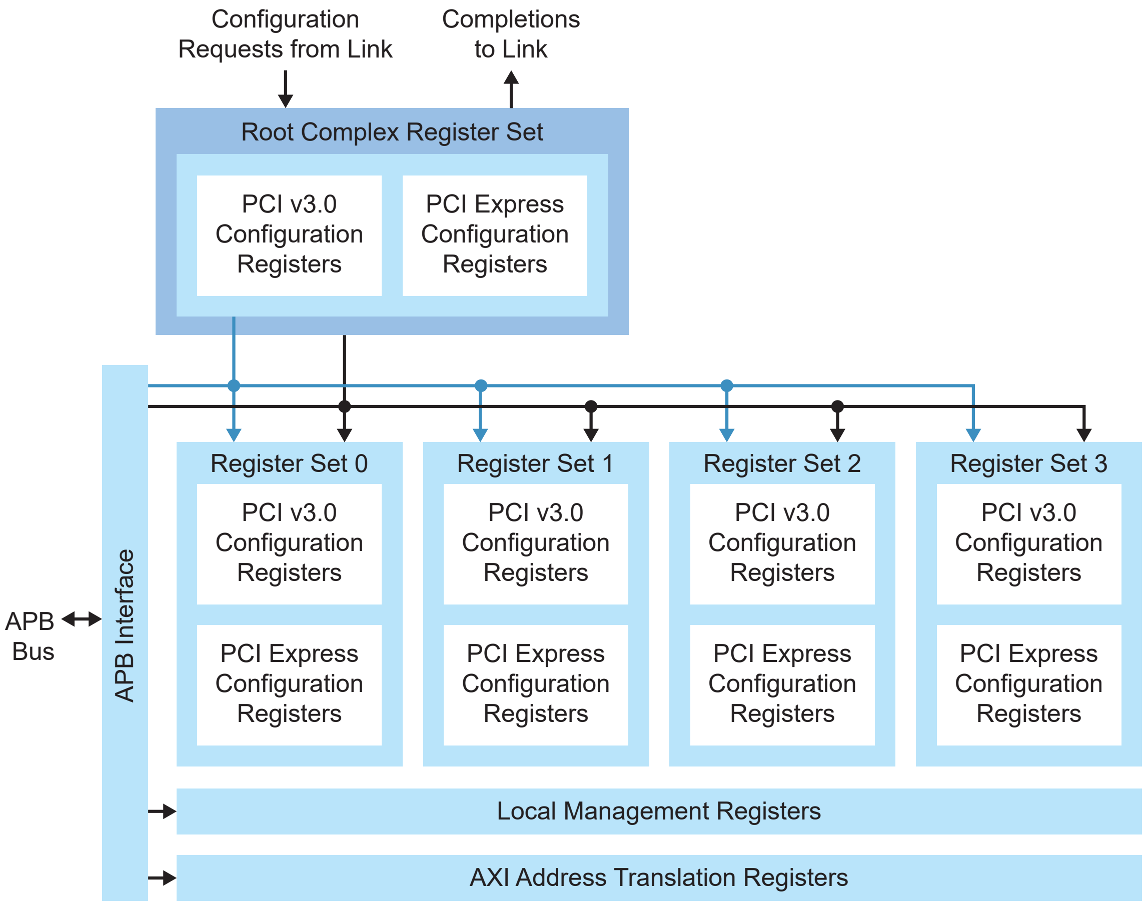

Configuration Requests from Link
Completions to Link
Root Complex Register Set
PCI v3.0 Configuration Registers
PCI Express Configuration Registers
APB Bus
APB Interface
Register Set 0
PCI v3.0 Configuration Registers
PCI Express Configuration Registers
Register Set 1
PCI v3.0 Configuration Registers
PCI Express Configuration Registers
Register Set 2
PCI v3.0 Configuration Registers
PCI Express Configuration Registers
Register Set 3
PCI v3.0 Configuration Registers
PCI Express Configuration Registers
Local Management Registers
AXI Address Translation Registers

Table 42: Global Address Map for Local Management Bus (apb_paddr)

[23]	[22]	[21]	[20]	[19:12]	[11:0]
0	0	0	0	0	PCIe Physical Function 0 Registers
0	0	0	0	1	PCIe Physical Function 1 Registers
0	0	0	0	2	PCIe Physical Function 2 Registers
0	0	0	0	3	PCIe Physical Function 3 Registers
0	0	0	0	4 67	PCIe Virtual Function 0-63 Registers
0	0	0	0	68 255	Reserved
0	0	0	1	0	PCIe Local Management Registers

[23]	[22]	[21]	[20]	[19:12]	[11:0]
0	1	0	0	x	PCIe AXI Configuration Registers
1	0	0	0	0	PCIe Root Port Registers
1	0	1	0	0	PCIe Root Port Registers. In this mode, certain RO fields in the configuration space can be written. **Please see documentation of the RC mode registers below for more information.**

These register addresses are DWORD addresses. In write operations, individual bytes can be addressed with byte-enable bits. Any addresses not defined are reserved. A configuration access from the link to a reserved address causes the PCIe Controller to return a completion packet with a UR (unsupported request) completion code. A read from the local management bus to a reserved address returns all zeros, and a write to a reserved address does not modify any of the registers. All registers (with the exception of reserved or hardwired fields) are writable from the local management bus.

Note: Refer to the **TJ-Series PCIe Controller Registers User Guide** for a detailed description of the registers.

Configuration Snoop Interface

The PCIe Controller has configuration space registers for each function as defined in the PCIe specifications. It supports PCI-compatible configuration space as well as PCIe extended space registers. The PCIe Controller automatically builds a linked list of capability structures, depending on the features you choose when configuring the PCI Express block in the Efinity Interface Designer.

Learn more: Refer to the **TJ-Series Interfaces User Guide** for detail on configuring the PCI Express block in the Interface Designer.

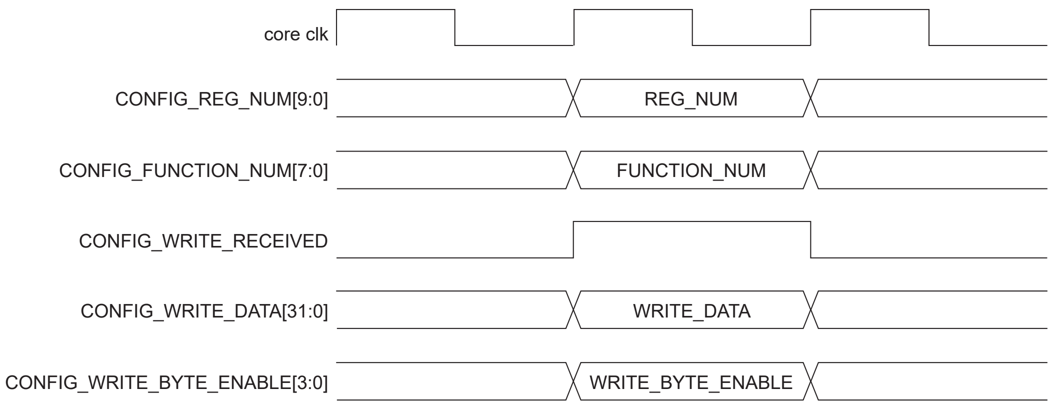
During the enumeration process, the host can traverse through the PCIe Controller's linked list of structures to find out which features are supported. Root port software uses the same capability structures for control and status information. The PCIe configuration read/write TLPs are used for this purpose. The PCIe Controller maps an incoming configuration read/write TLP to a read/write access to the internal configuration space registers. The control fields of various capability structures route to different PCIe Controller layers so that the logic can use the control information as expected.

The PCIe Controller also has an optional configuration snoop interface that lets you implement your own register set in the PCIe configuration space. The interface snoops incoming configuration read/write TLPs received from the link and places them on a simple configuration snoop read/write interface. You can enable the configuration snoop interface in the Interface Designer.

Note: You obtain the actual configuration register address by multiplying the `CONFIG_REG_NUM` value by 4. For example, a configuration space address of 0xa00 equates to a value on `CONFIG_REG_NUM` of 0x280.

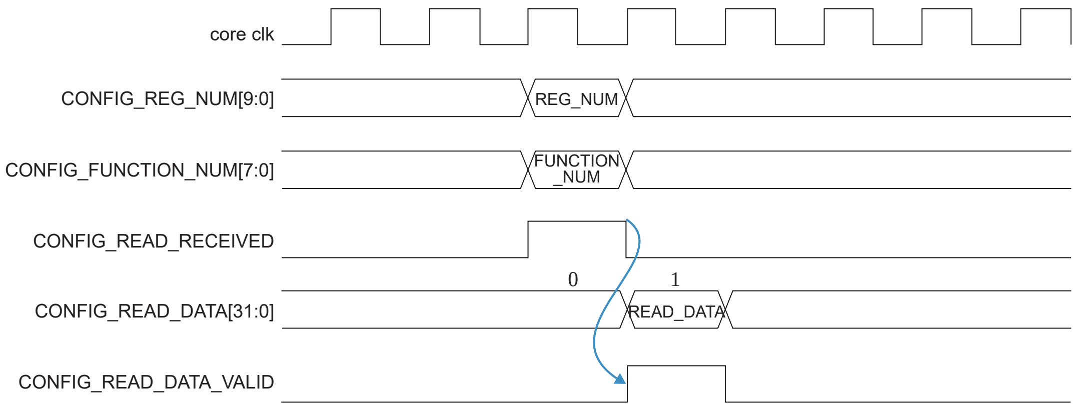
For a configuration write transaction, the interface places the address and data on the configuration snoop interface and asserts the `CONFIG_WRITE_RECEIVED` signal for one clock cycle. If the PCIe Controller implements the register being accessed, the write data is updated internally.

Figure 38: Configuration Snoop Interface Write Waveform



For configuration read transactions, the interface places the address on the configuration snoop interface and asserts the `CONFIG_READ_RECEIVED` signal for one clock cycle. If you want to provide the read data externally for this address, your user application must place the data on the `CONFIG_READ_DATA[31:0]` read data bus and assert `CONFIG_READ_DATA_VALID` on the first clock cycle after `CONFIG_READ_RECEIVED` is sampled high. The externally-provided data is sent back in the completion TLP. If the `CONFIG_READ_RECEIVED` input is not asserted by the user application, the PCIe Controller returns the data from its registers.

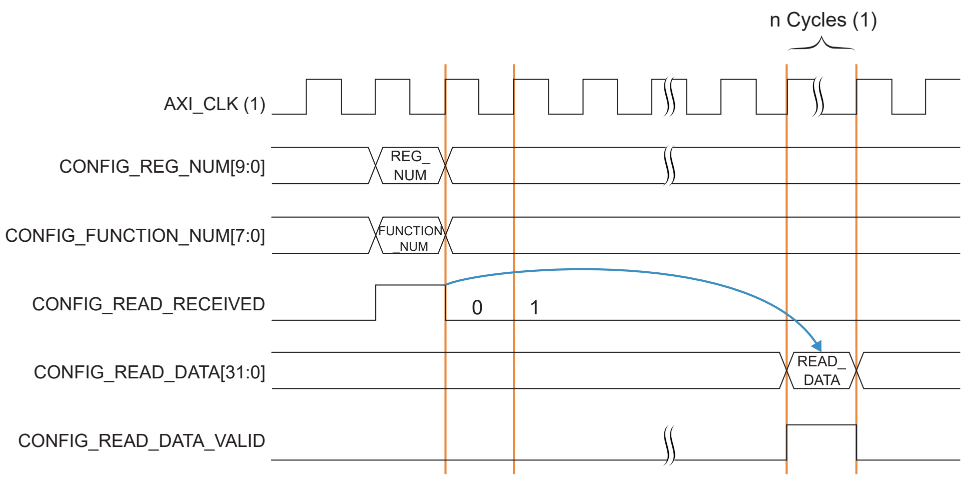
Figure 39: Configuration Snoop Interface Read Waveform



If you need more clock cycles for the configuration snoop read, you set a local management register bit. Set the Enable Extended Config Snoop Read bit in the Debug Mux Control 2 Register to `1` to change the read interface timing as shown in the following figure. If you want to provide the read data externally for this address, the user application must place the data on the `CONFIG_READ_DATA[31:0]` read data bus and assert `CONFIG_READ_DATA_VALID` within the specified number of clock cycles after `CONFIG_READ_RECEIVED` is sampled. If

`CONFIG_READ_DATA_VALID` is not asserted within the window, the PCIe Controller sends the data placed in the internal registers to the completion TLP.

Figure 40: Extended Configuration Snoop Interface Read Waveform



Note:
1. The number of clock cycles depends on the AXI_CLK speed.

AXI_CLK (MHz)	Clock Cycles
125	0 - 2
160	0 - 5
200	0 - 7
250	0 - 10

Note: This mechanism is not intended to completely replace a capability structure. A capability structure has many control fields that the PCIe Controller uses throughout its layers for specification-mandated operation.

The control fields are routed from the internally implemented registers to the various parts of the PCIe Controller. Therefore, replacing a capability structure entirely outside the PCIe Controller is not possible.

Vendor-Specific Extended Capability (VSEC)

The PCIe specification strongly recommends that PCIe devices do not place in the configuration space any registers other than those that are architected by the PCIe specification.

Device-specific registers that need to be placed in configuration space (e.g., they need to be accessible before memory space is allocated) can be placed in the Vendor-Specific Extended Capability (VSEC) structure in PCIe extended configuration space. This structure lets you use the extended capability mechanism to expose vendor-specific registers; for example, vendor-specific features that are used in a series of components from that vendor. A VSEC structure can tell vendor-specific software which features a particular component supports, including components developed after the software was released.

The PCIe Controller implements the VSEC register structure with 8 bytes of vendor-specific registers. Additionally, it provides a custom interface to help you access the vendor-specific registers directly, including:

- PCI Express Extendeed Capability Register
- Vendor-Specific Header
- Vendor-Specific Registers

Per the PCIe specification, the vendor-specific register structure and definition is determined by the vendor, indicated by the vendor ID field located at byte offset 00h in the PCI-compatible configuration space. The number of bytes of vendor-specific registers are advertised in the VSEC Length field.

The PCIe Controller implemetns 8 bytes of vendor-specific registers in:

- i_vendor_specific_control_reg, address @0x408

- i_vendor_specific_data_reg0, address @0x40c

Table 43: i_vendor_specific_control_reg Fields

Bits	Attributes	Descriptions
7:0	RO	These bits are read only for the host. The client can control these bits with the F0_VSEC_CONTROL_IN[7:0] input. The host's vendor-specific software can access these bits with a configuration read.
8	RW	These bits are read/write for the host. The host's vendor-specific software can access these bits with a configuration read or write. Bit 8 drives the PCIe Controller's F0_VSEC_INTERRUPT_OUT output.
31:9	RW	These bits are read/write for the host. The host's vendor-specific software can access these bits with a configuration read or write. Bits [31:9] drive the PCIe Controller' F0_VSEC_CONTROL_OUT[22:0] output.

Table 44: i_vendor_specific_data_reg0 Fields

Bits	Attributes	Descriptions
31:0	RW	These bits are read/write for the host. The host's vendor-specific software can access these bits with a configuration read or write.

- `F0_VSEC_CONTROL_IN`—This input drives bits [7:0] of the vendor-specific registers. The value to be driven on this input and its applications are specific to the user application.
- `F0_VSEC_INTERRUPT_OUT`—This output is driven by bit [8] of the vendor-specific registers in the PF0 vendor-specific capability structure. The signal is available for the user application; for example, the host can use it to signal a software-driven interrupt to the client application outside the PCIe Controller.
- `F0_VSEC_CONTROL_OUT`—This output is driven by bits [31:9] of the vendor-specific registers in the PF0 vendor-specific capability structure. The signal is available for the user application; for example, the host can ujse this signal to indicate vendor specific control data to the client application outside the PCIe Controller.

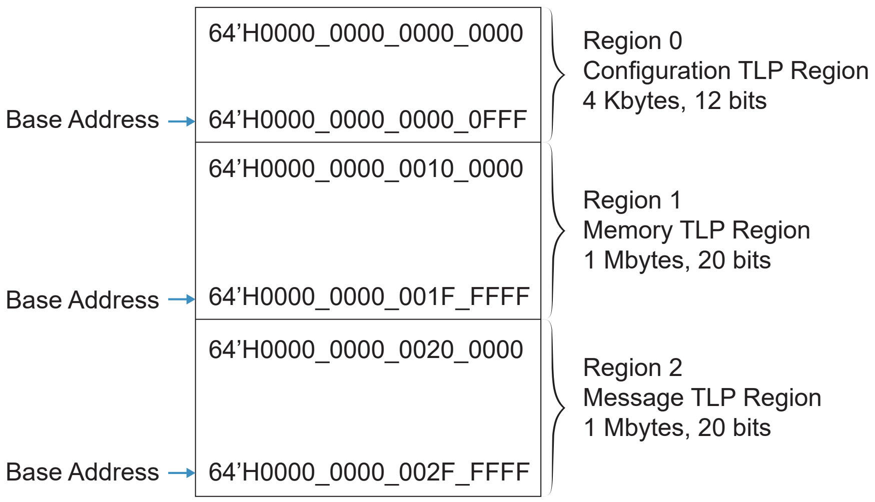
Configuration Guide

The following topics provide example of how to access the memory space.

AXI Outbound Access Example

The following figure shows the outbound AXI memory map with three region partitions and their corresponding base address and region size.

Figure 41: Outbound AXI Address Map for TLP Accesses



The actual region type is assigned after you program the Outbound PCIe Descriptor Registers with correct values for each TLP type as described in **Table 24: desc0: Outbound PCIe Descriptor Register for Different TLP Accesses** on page 47. You program the base address and region size into the **Table 28: AXI Region Base Address Registers** on page 49 for the corresponding region number.

For this example, the values to program in the AXI Region Base Address Registers are shown in the following table.

Table 45: AXI Region Base Address Register Values

Region Number	TLP Type desc0[3:0]	AXI_ADDR0[5:0]	AXI_ADDR0[31:8]	AXI_ADDR1[31:0]
0 (Configuration TLP)	b'1010	6'd11	24'h00_0000	32'h0000_0000
1 (Memory or I/O TLP)	b'10	6'd19	24'h00_1000	32'h0000_0000
2 (Message TLP)	b'1100	6'd19	24'h00_2000	32'h0000_0000

Accessing the Configuration TLP

There are two methods to convey the BDF (completer ID) information for a configuration TLP. Configuration TLPs are always routed to the endpoint's PCIe configuration space.

Method 1

This method uses the bus number, device number, and function number and the registers described in **Outbound AXI to PCIe Address Translation Registers**. You can drive the configuration register address on the `MASTER_AXI_AW/RADDR[11:0]` bits. Program the pass bits or `addr0 [5:0]` of the **Table 21: ob_addr1 Outbound AXI-to-PCIe Address Translation Registers** on page 47 as `6'd11` so that the lower 12 bits are taken from `MASTER_AXI_AW/`

`RADDR`. In this case, the BDF information is programmed in the corresponding region's (region 0 in this example).

Table 46: BDF Value Programming for Configuration TLPs through Outbound AXI to PCIe Address Translation Register

BDF values to be programmed in the addr0 register for legacy and ARI mode.

Legacy Mode		ARI Mode	
addr0 bits	**Value**	**addr0 bits**	**Value**
27:20	Bus number	27:20	Bus number
19:15	Device number	19:12	Function number
14:12	Function number		

Note: With this method, the AXI region size should be at least 4K bytes. Program `addr0[31:28]` and `addr1[31:0]` to zero.

Method 2

This method uses the bus number, device number, and function number from the `MASTER_AXI_AW/RADDR`. Program `addr0[5:0]` of the **Table 21: ob_addr1 Outbound AXI-to-PCIe Address Translation Registers** on page 47 to `6'd27` so that the lower 28 bits of the AXI address are passed to the PCIe Controller directly. These bits include the BDF values as indicated in the following table.

Table 47: BDF Values Programming for Configuration TLPs through the MASTER_AXI_AW/RADDR

Legacy Mode		ARI Mode	
MASTER_AXI_AW/ RADDR bits	**Value**	**MASTER_AXI_AW/ RADDR bits**	**Value**
27:20	Bus number	27:20	Bus number
19:15	Device number	19:12	Function number
14:12	Function number		

Note: With this method, the AXI region size should be at least 256 Mbytes (28 bits) because the BDF information is passed through the bits [27:12] of the `MASTER_AXI_AW/RADDR` or AXI address. Program `addr0[31:28]` and `addr1[31:0]` to zero.

Programming the Outbound PCIe Descriptor Register

When you program the outbound PCIe Descriptor Register, you assign a specific TLP type to the region number. For this example, follow the values to be programmed in the region 0 `desc0`, `desc1`, `desc2`, and `desc3` registers given in the **Table 24: desc0: Outbound PCIe Descriptor Register for Different TLP Accesses** on page 47 Configuration TLPs column.

Address Translation

Consider a different example where the AXI configuration TLP address map does not start at 64'h0000_0000_0000_0000. Use the PCIe Controller's address translation feature to translate the addresses to the requester id and configuration register number.

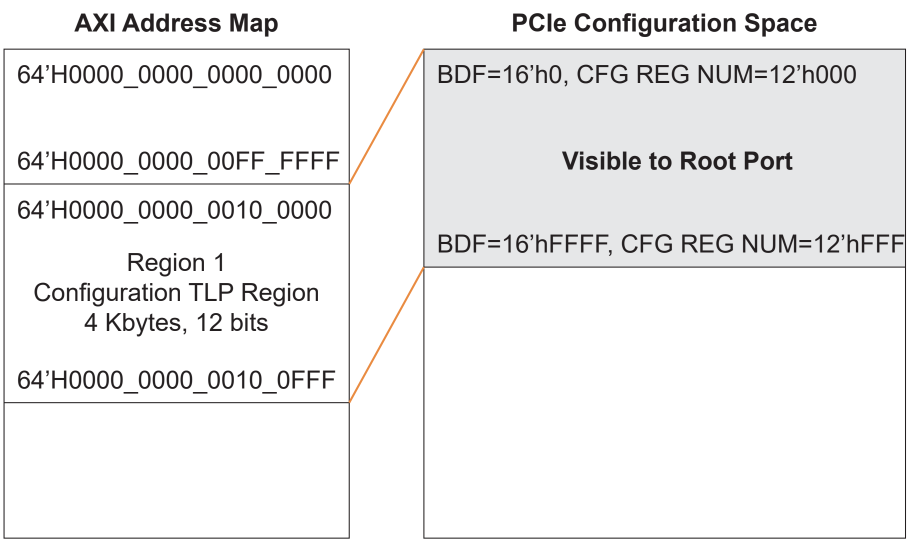
Table 48: PCIe Configuration Write and I/O Write Requests

Signal	Example 1	Example 2
MASTER_AXI_ADDR	0x4	0x0
MASTER_AXI_WSTRB	0xF0	0x0F

Example 1

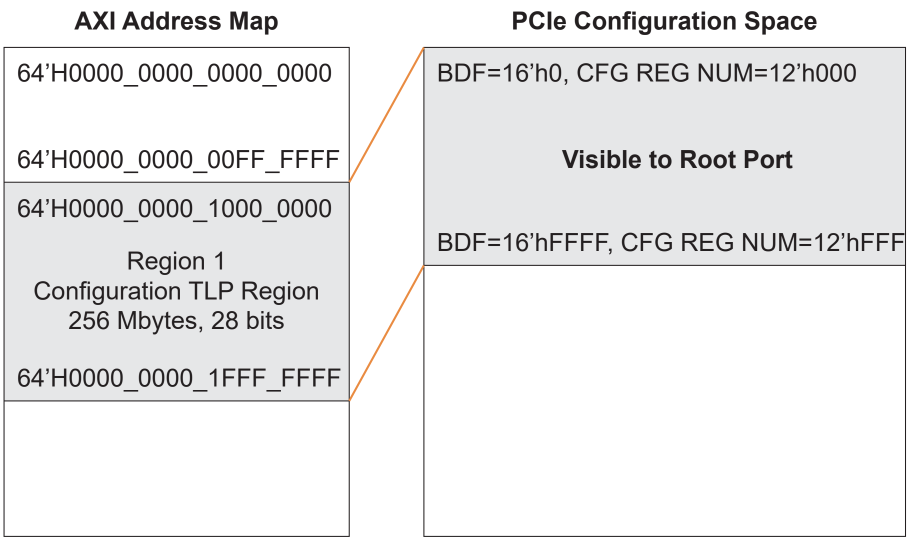
In this method the AXI region has a region size of 4K bytes (lower 12 bits of AXI address). The BDF information is captured from the region registers. This method is best for smaller BDFs; that is, the PCIe configuration space is enough to handle it.

Figure 42: Outbound AXI to PCIe Configuration Space Address Translation for Configuration TLPs



Example 2

Figure 43: Outbound AXI to PCIe Configuration Space Address Translation for Configuration TLPs



The PCIe configuration write and I/O write requests have a payload size of one DWORD. The AXI address should be aligned to the first data byte enable that is provided. The data/byte enable starts from the first address location given. One DWORD of data starting from the above address is used for the configuration and I/O write payload.

Memory or I/O TLP Access

For this example, assume that the one of the PCIe Controller's memory regions has an AXI region base address of 64'h0000_0000_0010_0000 (region 1 as shown in the following figure) with a region size of 1 Mbytes. Assume you want to write to the 64'h0000_0000_3000_0000 base address.

When you program the Outbound PCIe Descriptor Register you assign a specific TLP type to the region number. Follow the values to be programmed in the `desc0`, `desc1`, `desc2`, and `desc3` registers given in **Outbound PCIe Descriptor Registers** on page 47 Memory or I/O TLPs column.

Figure 44: Outbound AXI to PCIe Address Translation Register Programming

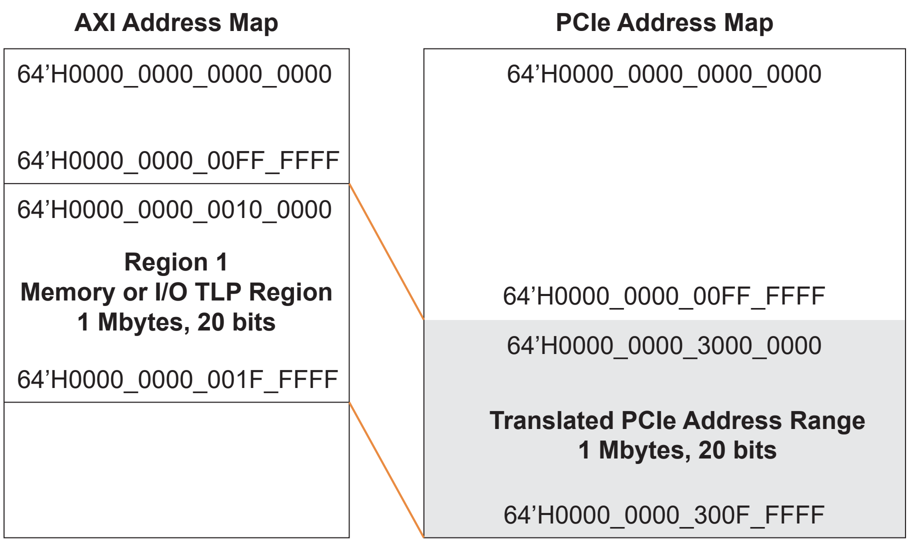


Table 49: AXI Region Base Address Register and Outbound AXI to PCIe Address Translation Register Values

Register	Value	Register	Value
AXI_ADDR0[5:0]	6'd19	AXI_ADDR0[5:0]	6'd19
AXI_ADDR0[31:8]	24'h00_1000	AXI_ADDR0[31:8]	24'h30_0000
AXI_ADDR1[31:0]	32'h0000_0000	AXI_ADDR1[31:0]	32'h0000_0000

Message TLP Access

When an outbound message access is made to an AXI region, the PCIe header fields are driven from the region register values. Only message code and message routing fields are driven through the AXI address (`MASTER_AXI_AWADDR`).

This example uses region two as a message region. The messages region is decoded after an access is made with an AXI address that falls between 64'h0000_0000_0020_0000 and 64'h0000_0000_002F_FFFF. The Outbound PCIe Descriptor Registers for region two must be programmed with values for message TLPs.

Note: Messages should have a minimum region size of 128 Kbytes. A bit for the message code, message routing, and message with/without data is driven through the AXI address (refer to **Table 23: MSG_ROUTING and MSG_CODE Fields** on page 47).

For vendor-defined messages, the vendor defined message header bits [127:72] is driven through the Outbound AXI to PCIe Address Translation Registers as explained in **Table 21: ob_addr1 Outbound AXI-to-PCIe Address Translation Registers** on page 47.

Endpoint Autonomous Link Bandwidth Management

The PCIe Controller performs link bandwidth management by enabling the endpoint to change the link speed autonomously (which is permitted by the PCIe specification). However, the PCIe specification does not define the mechanism to initiate the autonomous change. In the PCIe Controller, firmware can initiate the autonomous link speed change.

Upon reset de-assertion, the link initially trains up to Gen4 speed if supported by both link devices. Otherwise, the link initially trains up to Gen1 speed. The endpoint can autonomously initiate a link speed change with the following process:

1. Check the current negotiated link speed by reading the Negotiated Link Speed field of the Link Control and Status Configuration Register (bits [19:16]).
2. Check the host's upper link speed limit by reading the Target Link Speed field in the Link Control 2 Configuration Register.
3. If the endpoint needs to change the link speed within the limit constrained by the host:
   a. Program the required speed in the endpoint's Target Link Speed field of the Local Management Register - Link Width Control Register (bits [25:24]) .
   b. Trigger link retraining by writing a 1 to the endpoint's Link Speed Change Retrain Link bit of the Local Management Register - Linkwidth Control Register.
   c. Wait for the endpoint Link Speed Change Retrain Link bit to clear. The PCIe Controller clears this bit when link retraining is complete.
4. To check if the speed change was successful, read the Negotiated Link Speed field of the Link Control and Status Configuration Register (bits [19:16]).

The host can disable autonomous link speed changes on the endpoint by setting the Hardware Autonomous Speed Disable bit in the Link Control and Status Register 2. If the host sets this bit, any endpoint request to change the link speed will not be successful. The PCIe Controller enters recovery but does **not** initiate a speed change.

The firmware should always wait for a link retraining request to complete before initiating another retraining request.

After initiating a link retrain request, do not write to the Linkwidth Control Register utill the retraining request is completed.

Programming the SR-IOV Registers

The following topics describe how the VF numbers are allocated and how to set up the VF BARs.

VF Function Number Allocation

The VF numbers begin immediately after the PF numbers; there are no gaps in the function number allocation between PFs and VFs. The VF stride is fixed at `0x1`. Therefore, all VFs that belong to the same PF are allocated successive function numbers.

Table 50: VF Function Number allocation

Routing ID	Description
0	PF0
1	PF1
2	PF2
3	PF3
4 to 19	PF0_VF1 to PF0_VF16
20 to 35	PF1_VF1 to PF1_VF16
36 to 51	PF2_VF1 to PF2_VF16
52 to 67	PF3_VF1 to PF3_VF16

If you configure the PCIe Controller to have multiple PFs, you can change the total number of VFs allocated to each PF by programming bits [15:0] in the Total VF Count Register in the PFs' SR-IOV extended capabilities. When you modify the field, the VF function number allocation changes and ensures that all VFs are allocated successive function numbers. Firmware should re-program the First VF Offset field to reflect the new VF function number allocation as described below.

- PF0 First VF Offset[15:0] = Total Number of PFs
- PF1 First VF Offset[15:0] = Total Number of PFs + (PF0 Total VF Count[15:0]) - 1
- PF2 First VF Offset[15:0] = Total Number of PFs + (PF0 Total VF Count[15:0]) + (PF1 Total VF Count[15:0]) - 2
- PF3 First VF Offset[15:0] = Total Number of PFs + (PF0 Total VF Count[15:0]) + (PF1 Total VF Count[15:0]) + (PF2 Total VF Count[15:0]) - 3
- PF4 First VF Offset[15:0] = Total Number of PFs + (PF0 Total VF Count[15:0]) + (PF1 Total VF Count[15:0]) + (PF2 Total VF Count[15:0]) + (PF3 Total VF Count[15:0]) - 4
- PF5 First VF Offset[15:0] = Total Number of PFs + (PF0 Total VF Count[15:0]) + (PF1 Total VF Count[15:0]) + (PF2 Total VF Count[15:0]) + (PF3 Total VF Count[15:0]) + (PF4 Total VF Count[15:0]) - 5
- PF6 First VF Offset[15:0] = Total Number of PFs + (PF0 Total VF Count[15:0]) + (PF1 Total VF Count[15:0]) + (PF2 Total VF Count[15:0]) + (PF3 Total VF Count[15:0]) + (PF4 Total VF Count[15:0]) + (PF5 Total VF Count[15:0]) - 6
- PF7 First VF Offset[15:0] = Total Number of PFs + (PF0 Total VF Count[15:0]) + (PF1 Total VF Count[15:0]) + (PF2 Total VF Count[15:0]) + (PF3 Total VF Count[15:0]) + (PF4 Total VF Count[15:0]) + (PF5 Total VF Count[15:0]) + (PF6 Total VF Count[15:0]) – 7

Setting up the VF BAR Registers

The PCIe Controller provides the following registers in the local management space to allow user control of the VF BAR registers:

- Virtual Function BAR Configuration Register 0
- Virtual Function BAR Configuration Register 1

You can setup the following BAR parameters with these registers:

- VF BAR size—32-bit or 64-bit BAR
- VF BAR type—Prefetchable or non-prefetchable
- VF BAR aperture

Download: You set up the VF BAR registers in the Interface Designer (**PCI Express block** > **Function tab** > **Physical Function n sub-tab** where *n* is 0 - 3).

Efinix **does not** recommend changing these settings via the APB interface.

The VF BAR aperture is also affected by the System Page Size register in the SR-IOV extended capability. Each VF BAR*n* or VF BAR*n* pair should be aligned on a system page size boundary. Additionally, each VF BAR*n* or VF BAR*n* pair defining a non-zero address space should be sized to consume an integer multiple of system page size bytes.

If the VF BAR aperture is not aligned to the system page size, the PCIe Controller internally overrides the user settings to align the aperture to the system page size. For example:

- *VF BAR aperture is not aligned to system page size*—If the VF BAR 0 aperture is 4 KB in Virtual Function BAR Configuration Register 0 and if the host-programmed System Page Size Register is 4 MB in the SR-IOV extended capability, the PCIe Controller internally requests 4 MB in the VF BAR0 register, aligned to the system page size.
- *VF BAR Aperture is aligned to System Page Size*—If the VF BAR 0 aperture is 8 KB in Virtual Function BAR Configuration Register 0 and if the host-programmed System Page Size Register is 4 KB in the SR-IOV extended capability, the PCIe Controller internally requests 8 KB in the VF BAR0 register (as requested by the user).

Managing Outbound NP Outstanding Requests and Completion Responses (Endpoint)

As per the PCIe specification, an endpoint must advertise infinite credits for completion packets. Therefore, the endpoint must be ready to accept all completions it receives for all non-posted requests it initiates. The endpoint can receive responses for NP requests in multiple split completion packets.

The PCIe Controller has two buffers to store the received completion packets:

- *Stage 1 buffer*—Completion FIFO RAM
- *Stage 2 buffer*—AXI re-ordering FIFO RAM

The stage 1 buffer performs posted vs. completion ordering checks. The PCIe Controller stores completion packets in this FIFO until the posted vs. completion ordering checks are cleared. The stage 2 buffer reorders the received split completion packets and merges them to form a single completion response for each request. This process is needed because the AXI interface cannot accept split completions.

Table 51: Outbound NP Request Parameters

Parameter	Value
Maximum number of outstanding NP requests.	128
Maximum size of each NP request (MRRS).	4096B
Maximum number of split completion packets received per NP request assuming each NP request is of MRRS, 64B RCB boundary and addresses are non-64B aligned.	65
Maximum number of split completion packets that can be received.	8320
Maximum number of split completion packets that can be stored in the stage 1 buffer.	256

Although the stage 2 buffer can store the full size of the completion data, the stage 1 buffer is smaller and the maximum possible Completion data exceeds the stage 1 buffer storage capacity. The stage 1 buffer is temporary storage for completion packets and can only store a limited number of them. The stage 1 buffer is is not designed to store the maximum possible completion data. Therefore, the posted data must not block the completion data. The client must drain the posted data at PCIe link rate; otherwise, a completion FIFO RAM overflow can occur.

The following sections describe how the client can prevent completion FIFO overflows.

Drain Received Inbound Completion Packets at PCIe Link Rate

With this method, the client guarantees that the write data received on the PCIe Controller's AXI interface is drained at the PCIe link rate. This method ensures that the posted packets do not block the completion packets, which avoids a completion FIFO overflow.

Disable Independent Posted vs. Completion Ordering Checks

You can only use this option if the received inbound completion and posted data streams are completely independent. In this case, the client programs the local management register Disable Ordering Checks bit [30] in the i_debug_mux_control_reg register. This setting disables posted vs. completion ordering checks and decouples the two inbound data streams, and the posted packets no longer block the completion packets, which avoids a completion FIFO overflow.

Limit Outstanding NP Read Requests to Ensure FIFO Never Overflows

The client limits the total number of outstanding NP requests thereby ensuring that the completion FIFO does not overflow. Limit the number by setting the Maximum NP Outstanding Request Limit[7:0] field in the local management Debug Mux Control 2 Register. Program it based on maximum size of the NP requests and NP request address alignment as shown in the following table.

Table 52: Controlling Maximum NP Outstanding Requests

Maximum NP Request Size (MRRS)	All NP 64B Rquest Addresses Aligned?	Programmed Value in Maximum NP Outstanding Request Limit Register
64 B	Yes	128
64 B	No	128
128 B	Yes	128
128 B	No	85

Maximum NP Request Size (MRRS)	All NP 64B Rquest Addresses Aligned?	Programmed Value in Maximum NP Outstanding Request Limit Register
256 B	Yes	64
256 B	No	51
512 B	Yes	32
512 B	No	28
1024 B	Yes	16
1024 B	No	15
2048 B	Yes	8
2048 B	No	7
4096 B	Yes	4
4096 B	No	3

Interface Signals

The following topics describe the PCIe Controller signal interface. Refer to **Figure 2: PCIe Controller Block Diagram** on page 5 for the block diagram.

Clock Signals

Table 53: Clock Ports

Signal	Direction	Width	Description
AXI_CLK	Input	1	AXI interface clock.
PM_CLK	Output	1	Free-running clock used for low power state transitions and clock control generation.
USER_APB_CLK	Input	1	APB interface clock.

Reset Interface Signals

Table 54: Reset Interface

Signal	Direction	Width	Clock Domain	Description
HOT_RESET_IN	Input	1	AXI_CLK	When this input is asserted in root port mode, the PCIe Controller initiates a hot reset sequence on the PCIe link. The PCIe Controller keeps the PCIe link in hot reset as long as this signal remains asserted. When de-asserted, the PCIe Controller brings the PCIe link out of hot reset and initiates link training.
HOT_RESET_OUT	Output	1	AXI_CLK	The PCIe Controller asserts this output when a hot reset is received from the link in endpoint mode. This signal is an active-high output driven synchronous to AXI_CLK.
LINK_DOWN_RESET_OUT	Output	1	AXI_CLK	The PCIe Controller asserts this output when the LTSSM detects a link-down event (i.e., when the LINK_UP state variable goes to 0). This signal is an active-high output driven synchronous to AXI_CLK. It is asserted high for eight AXI_CLK clock cycles during a link down event.
PERST_N	Input	1	Async	Triggers a warm reset from the I/O pad.
RESET_REQ	Output	1	Async	When this signal is asserted, the PCIe Controller requests to trigger a warm or hot reset. Refer to **Reset Handshake.**
RESET_ACK	Input	1	Async	Assert this signal to indicate readiness and permission for a warm or hot reset. Refer to **Reset Handshake**.

AXI Master Interface Signals

Table 55: AXI Master Write Address Channel

Signal	Direction	Width	Clock Domain	Description
TARGET_AXI_AWREADY	Input	1	AXI_CLK	Ready signals from the client to the PCIe Controller indicating that the application is ready to sample the address and associated parameters from the target write interface. The address and associated parameters are transferred across the interface when TARGET_AXI_AWVALID and TARGET_AXI_AWREADY are both high in a clock cycle.

Signal	Direction	Width	Clock Domain	Description
TARGET_AXI_AWADDR	Output	64	AXI_CLK	The PCIe Controller places the address of the first byte in a burst when initiating a write transaction on the target write interface. The address is valid when TARGET_AXI_AWVALID is asserted. The AXI address is the starting byte-level address of the memory block or I/O location to be read or written. When the transaction is a 32-bit read/write, bits [63:32] are set to zeroes.
TARGET_AXI_AWID	Output	8	AXI_CLK	This output contains an 8-bit tag to identify the write transaction. This output is valid when TARGET_AXI_AWVALID is asserted.
TARGET_AXI_AWLEN	Output	8	AXI_CLK	Indicates the number of beats (data transfer cycles) associated with the current burst (0000 = 1 beat, 0001 = 2 beats, ..., 1111 = 16 beats). This information is valid when TARGET_AXI_AWVALID is asserted. The valid bytes within each beat are identified by the write strobe signal TARGET_AXI_WSTRB[7:0].
TARGET_AXI_AWSIZE	Output	3	AXI_CLK	Indicates the size of the AXI transfer.
TARGET_AXI_AWUSER	Out	88	AXI_CLK	Sideband status information for inbound AXI write transfer. [2:0] Transaction type: 010: Memory write 011: I/O write All other values are reserved.
				[5:3] PCIe attributes associated with the request.
				[21:6]: PCI Requester ID associated with the request. With the legacy interpretation of RIDs, these 16 bits are divided into: • [31:24] an 8-bit bus number • [23:19] 5-bit device number • [18:16] 3-bit function number When ARI is enabled, bits [31:24] carry the 8-bit bus number and [23:16] provide the function number.
				[29:22] Request's PCI tag.
				[32:30] Request's PCIe transaction class (TC).
				[35:33] For memory and I/O requests, these bits identify the matching BAR for the memory or I/O address. For 64-bit transactions, the BAR number is given as the lower address of the matching pair of BARs (i.e., 0, 2 or 4).(10) 000: BAR 0 001: BAR 1 010: BAR 2 011: BAR 3 100: BAR 4 101: BAR 5 110: Expansion ROM access. For message requests, these bits provide the 3-bit Routing field r[2:0] from the message header.
				[43:36] Request's target function determined by the BAR check. When ARI is in use, all 8 bits of this field are valid. Otherwise, only bits [50:48] are valid. This field is valid only for memory and I/O requests and is set to 0 for message requests.(11)
				[51:44] For message requests, these bits provide the message code from the message TLP header. These bits are reserved for all other request types.
				[59:52] 8-bit steering tag for the hint.
				[61:60] Value of PH[1:0] associated with the hint.
				[62] Set when the request has a transaction processing hint associated with it.

(10) This description is also applicable for root ports. If RC BAR check is enabled, 000 = RC BAR0 and 001 = RC BAR2 for the two RC 64-bit BARs.
(11) This signal is applicable to endpoints only. For root ports, these bits are 0.

Signal	Direction	Width	Clock Domain	Description
				[64:63] PCIe AT bits: 00: Untranslated 01: Translation request 10: Translated 11: Reserved
				[65] PASID present.
				[85:66] PASID value, 20 bits maximum. The size depends on the Max PASID Width field in the PASID Capability Register.
				[86] Privilege mode access.
				[87] Execute mode access.
TARGET_AXI_AWVALID	Output	1	AXI_CLK	Valid signal for the address and control information on the AXI slave write interface. The PCIe Controller keeps this valid signal asserted until the client sets the ready input to the PCIe Controller (TARGET_AXI_AWREADY) in response.

Table 56: AXI Master Write Data Channel

Signal	Direction	Width	Clock Domain	Description
TARGET_AXI_WREADY	Input	1	AXI_CLK	Ready for write data from the client to the PCIe Controller. The client must assert this signal when it is ready to receive the next beat from the PCIe Controller.
TARGET_AXI_WDATA	Output	256	AXI_CLK	Data associated with a memory write operation delivered from the PCIe Controller. Data is transferred in little-endian order. For writes, data is transferred aligned. The data on this bus is valid when TARGET_AXI_WVALID is high.
TARGET_AXI_WDATA_PAR	Output	32	AXI_CLK	Contains the end-to-end parity for TARGET_AXI_WDATA. Odd parity is computed for every byte of the data and propagated through the PCIe Controller for end-to-end parity protection.
TARGET_AXI_WLAST	Output	1	AXI_CLK	Asserted in the last beat of the burst to indicate the end of the write transaction.
TARGET_AXI_WSTRB	Output	32	AXI_CLK	Indicates valid bytes in the first and last beat of the data block being transferred. Data is transferred aligned. Indicates valid bytes of the data block being transferred. The AXI interface supports noncontiguous byte enables on any data block. The AXI logic splits the write packets based on the write strobes, followimg the PCIe first/last byte enable as described in the specification.
TARGET_AXI_WSTRB_PAR	Output	4	AXI_CLK	Contains the end-to-end parity for TARGET_AXI_WSTRB.
TARGET_AXI_WVALID	Output	1	AXI_CLK	The PCIe Controller maintains data on the bus until the client has asserted TARGET_AXI_WREADY.

Table 57: AXI Master Write Response Channel

Signal	Direction	Width	Clock Domain	Description
TARGET_AXI_BID	Input	8	AXI_CLK	This output contains an 8-bit tag to identify the response phase of a write transaction. This output is valid when TARGET_AXI_BVALID is asserted.
TARGET_AXI_BID_PAR	Input	1	AXI_CLK	Contains the end-to-end parity for TARGET_AXI_BID.
TARGET_AXI_BRESP	Input	2	AXI_CLK	Indicates the response to the transaction when TARGET_AXI_BVALID is asserted.
TARGET_AXI_BRESP_PAR	Input	1	AXI_CLK	Contains the end-to-end parity for TARGET_AXI_BRESP.
TARGET_AXI_BVALID	Input	1	AXI_CLK	Valid for write response from client to PCIe Controller.

Signal	Direction	Width	Clock Domain	Description
TARGET_AXI_BREADY	Output	1	AXI_CLK	Ready for write response from PCIe Controller to client. Client should hold the response signals and valid until this signal is asserted.

Table 58: AXI Master Read Address Channel

Signal	Direction	Width	Clock Domain	Description
TARGET_AXI_ARREADY	Input	1	AXI_CLK	Ready signals from the client to the PCIe Controller indicating that the application is ready to sample the address and associated parameters from the target read interface. The address and associated parameters are transferred across the interface when TARGET_AXI_ARVALID and TARGET_AXI_READ_ARREADY are both high in a clock cycle.
TARGET_AXI_ARADDR	Output	64	AXI_CLK	Address of the first byte in the read request. The address is valid when TARGET_AXI_ARVALID is asserted. The AXI address is the starting byte-level address of the memory block or I/O location to be read or written. When the transaction is a 32-bit read/write, bits [63:32] are set to zeroes.
TARGET_AXI_ARID	Output	8	AXI_CLK	Read ID tag associated with the target memory read transaction. The client must store this tag and return it on TARGET_AXI_RID while transferring the data associated with the read request. This output is valid when TARGET_AXI_ARVALID is asserted.
TARGET_AXI_ARLEN	Output	8	AXI_CLK	Indicates the number of beats (data transfer cycles) associated with the read burst.
TARGET_AXI_ARSIZE	Output	3	AXI_CLK	Indicates size of the AXI transfer.
TARGET_AXI_ARUSER	Output	88	AXI_CLK	Sideband status information for inbound AXI read transfer. For 64-bit transactions, the BAR number is given as the lower address of the matching pair of BARs (that is, 0, 2 or 4). [2:0] Transaction type: 000: memory read 001: I/O read All other values are reserved.
				[5:3] Request's PCIe attributes associated.
				[21:6] PCI Requester ID associated with the request. With the legacy interpretation of RIDs, these 16 bits are divided into: • [21:14] 8-bit bus number • [13:9] 5-bit device number • [8:6] 3-bit function number When ARI is enabled, bits [21:14] carry the 8-bit bus number and bits [13:6] provide the function number.
				[29:22] PCI Tag associated with the request.
				[32:30] Request's PCIe transaction class (TC).

Signal	Direction	Width	Clock Domain	Description
				[35:33] For memory and I/O requests, these bits identify the matching BAR for the memory or I/O address.[12] 000: BAR 0 001: BAR 1 010: BAR 2 011: BAR 3 100: BAR 4 101: BAR 5 110: Expansion ROM access
				[43:36] Request's target function number as determined by the BAR check. When ARI is in use, all 8 bits of this field are valid. Otherwise, only bits [50:48] are valid. This field is valid only for memory and I/O requests, and is set to 0 for message requests.[13]
				[51:42] These bits are reserved for all read request types.
				[59:52] 8-bit steering tag for the hint.
				[61:60] Value of PH[1:0] associated with the hint.
				[62] Set when the request has a transaction processing hint associated with it.
				[64:63] PCIe AT bits: 00: Untranslated 01: Translation request 10: Translated 11: Reserved
				[65] PASID present.
				[85:66] PASID value, 20 bits maximum. The size depends on the Max PASID Width field in the PASID Capability Register.
				[86] Privilege mode access.
				[87] Execute mode access.
TARGET_AXI_ARVALID	Output	1	AXI_CLK	Valid signal for the address and control information on the AXI slave read interface. The PCIe Controller keeps this valid signal asserted until the client application sets the ready input to the PCIe Controller TARGET_AXI_ARREADY in response.

Table 59: AXI Master Read Data Channel

Signal	Direction	Width	Clock Domain	Description
TARGET_AXI_RDATA	Input	256	AXI_CLK	Data associated with a memory read operation delivered by the client to the PCIe Controller. Data is transferred in little-endian order. For memory reads, data is transferred in aligned fashion. The data on this bus is valid when TARGET_AXI_RVALID is high.
TARGET_AXI_RDATA_PAR	Input	32	AXI_CLK	Contains the end-to-end parity for TARGET_AXI_RDATA.
TARGET_AXI_RID	Input	6	AXI_CLK	When transferring data in response to a read request, the client must place the 6-bit read ID tag associated with the request on this bus. This tag must be valid when TARGET_AXI_READ_RVALID is high.
TARGET_AXI_RID_PAR	Input	1	AXI_CLK	Contains the end-to-end parity for TARGET_AXI_RID.

(12) This description is also applicable to root ports. The BAR check is enabled, 000 = root port BAR0, 001 = root port BAR2 for the 2 root port 64-bit BARs.
(13) This signal is applicable to endpoints only. For root ports, these bits are 0.

Signal	Direction	Width	Clock Domain	Description
TARGET_AXI_RLAST	Input	1	AXI_CLK	The client must assert this signal in the last beat of the burst to indicate the end of the read burst.
TARGET_AXI_RRESP	Input	2	AXI_CLK	Read status from client. Allowed status encoding are: 00: Good completion 10: Slave error Others: Not supported The PCIe Controller responds to the slave error by sending a completion on the link with the completer abort status, instead of a normal completion. The read response status must be valid when TARGET_AXI_RVALID is high.
TARGET_AXI_RRESP_PAR	Input	1	AXI_CLK	Contains the end-to-end parity for TARGET_AXI_RRESP.
TARGET_AXI_RVALID	Input	1	AXI_CLK	Indicates valid data on the TARGET_AXI_RDATA bus. The client must maintain data on the bus until the PCIe Controller has asserted TARGET_AXI_RREADY.
TARGET_AXI_RREADY	Output	1	AXI_CLK	Ready for read data from the PCIe Controller to the client. The PCIe Controller asserts this signal when it is ready to receive the next beat from the client.

Table 60: AXI Master Sideband Signals

Signal	Direction	Width	Clock Domain	Description
TARGET_NON_POSTED_REJ	Input	1	AXI_CLK	Asserted by client when it cannot service a non-posted request. The PCIe Controller does not present any non-posted requests that it receives from the PCIe link. Instead, it will holds them in the PNP FIFO RAM until the signal is de-asserted. If a non-posted TLP has already been queued from the PNP FIFO and this signal is asserted, the PCIe Controller places it on the AXI bridge. The client must accept the non-posted TLP. The in-flight non-posted TLPs in the PCIe Controller from the PNP FIFO cannot be stopped. However, non-posted TLPs that are in the PNP FIFO RAM when this signal is asserted or that come in after the signal is asserted are not forwarded to the AXI interface. The client must assert this signal when it still can process two or three non-posted TLPs. This requirement allows posted TLPs to go past non-posted TLPs at the AXI master write interface due to client not being able to service non-posted TLPs.

AXI Slave Interface Signals

Table 61: AXI Slave Interface Write Address Channel

Signal	Direction	Width	Clock Domain	Description
MASTER_AXI_AWADDR	Input	64	AXI_CLK	Address for the master-side write transaction from the client. The address is valid when MASTER_AXI_AWVALID is asserted. The AXI address is the starting byte-level address of the memory block, config or I/O location to be read or written. When the transaction is a 32-bit read/write, bits [63:32] must be set to zeroes.
MASTER_AXI_AWID	Input	8	AXI_CLK	The client must place a 8-bit identifier for the write transaction on this input. This tag is used to match the write completion status returned by the PCIe Controller with the associated request. This input must be valid when MASTER_AXI_AWVALID is asserted.

Signal	Direction	Width	Clock Domain	Description
MASTER_AXI_AWLEN	Input	8	AXI_CLK	Indicates the number of beats (data transfer cycles) associated with the current burst. This information is valid when MASTER_AXI_AWVALID is asserted. The valid bytes within each beat are identified by the write strobe signal MASTER_AXI_WSTRB.
MASTER_AXI_AWSIZE	Input	3	AXI_CLK	Burst size. This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.
MASTER_AXI_AWUSER	Input	88	AXI_CLK	Sideband descriptor. Refer to **AXI Outbound Access through Sideband Descriptor** for detailed description.
MASTER_AXI_AWVALID	Input	1	AXI_CLK	Valid signal for the address and other parameters associated with the request on the AXI master write interface. The client must keep this valid signal asserted until the PCIe Controller sets the ready output (MASTER_AXI_AWREADY) in response.
MASTER_AXI_AWREADY	Output	1	AXI_CLK	Ready signal from the PCIe Controller to the client, indicating that the PCIe Controller is ready to sample the address and associated parameters from the master write interface. The address and associated parameters are transferred across the interface when MASTER_AXI_AWVALID and MASTER_AXI_AWREADY are both high in a clock cycle.

Table 62: AXI Slave Interface Write Data Channel

Signal	Direction	Width	Clock Domain	Description
MASTER_AXI_WDATA	Input	256	AXI_CLK	Data associated with a memory write operation delivered from the client to the PCIe Controller. Data is transferred in little-endian order. For memory writes, data is transferred in aligned fashion. The data on this bus is valid when MASTER_AXI_WVALID is high.
MASTER_AXI_WDATA_PAR	Input	32	AXI_CLK	Contains the end-to-end parity for MASTER_AXI_WDATA.
MASTER_AXI_WLAST	Input	1	AXI_CLK	Asserted in the last beat of the burst to indicate the end of the write transaction.
MASTER_AXI_WSTRB	Input	32	AXI_CLK	Indicates valid bytes in the first and last beat of the data block being transferred. Data is transferred in aligned fashion. For write transactions with a payload size of 8 bytes or less, the byte strobes may be non-contiguous, as described in the PCIe specification.
MASTER_AXI_WSTRB_PAR	Input	4	AXI_CLK	Indicates valid bytes in the first and last beat of the data block being transferred. Data is transferred aligned. For write transactions with a payload size of 8 bytes or less, the byte strobes may be non-contiguous, as described in the PCIe Specifications.
MASTER_AXI_WVALID	Input	1	AXI_CLK	Indicates valid data on the MASTER_AXI_WDATA bus. The client must maintain data on the bus until the PCIe Controller has asserted MASTER_AXI_WREADY in return.
MASTER_AXI_WREADY	Output	1	AXI_CLK	Ready for write data from the PCIe Controller to the client. The core asserts this signal when it is ready to receive the next beat from the client.

Table 63: AXI Slave Interface Write Response Channel

Signal	Direction	Width	Clock Domain	Description
MASTER_AXI_BREADY	Input	1	AXI_CLK	Ready for write response from the PCIe Controller to the client. The client asserts this signal when it is ready to accept the next write response from the client
MASTER_AXI_BID	Output	8	AXI_CLK	For each write transaction received on the AXI master write interface, the PCIe Controller returns the completion status of the transaction by placing the 8-bit identifier of this transaction and asserting MASTER_AXI_BVALID.

Signal	Direction	Width	Clock Domain	Description
MASTER_AXI_BID_PAR	Output	1	AXI_CLK	Contains the end-to-end parity for MASTER_AXI_BID. Odd parity is computed for every byte of the data and propagated through the PCIe Controller for end-to-end parity protection.
MASTER_AXI_BRESP	Output	2	AXI_CLK	Write completion status from client. Valid status encoding are: 2'b00: Good completion 2'b10: Slave error, Completion error for Non-Posted Writes 2'b11: Decode error
MASTER_AXI_BRESP_PAR	Output	1	AXI_CLK	Contains the end-to-end parity for MASTER_AXI_BRESP. Odd parity is computed for every byte of the data and propagated through the PCIe Controller for end-to-end parity protection.
MASTER_AXI_BUSER	Output	5	AXI_CLK	[2:0] Completion error code. [4:3] Completion status code. Other fields are reserved. See **Appendix B: Error Handling** on page 112.
MASTER_AXI_BUSER_PAR	Output	1	AXI_CLK	Contains the end-to-end parity for MASTER_AXI_BUSER. Odd parity is computed for every byte of the data and propagated through the PCIe Controller for end-to-end parity protection.
MASTER_AXI_BVALID	Output	1	AXI_CLK	The PCIe Controller asserts this output when it has placed the completion status of a master write transaction on MASTER_WRITE_COMPLETION_STATUS. It keeps the output asserted until the client has asserted MASTER_AXI_BREADY.

Table 64: AXI Slave Interface Read Address Channel

Signal	Direction	Width	Clock Domain	Description
MASTER_AXI_ARADDR	Input	64	AXI_CLK	Address for the client's master-side read transaction. The client must places the address of the first byte of the burst on this bus when initiating a read transaction on the master read interface. The address is valid when MASTER_AXI_ARVALID is asserted.
MASTER_AXI_ARID	Input	8	AXI_CLK	Read ID tag associated with the master memory read transaction. This tag is used to match the read completion status returned by the PCIe Controller with the associated request. This output must be valid when MASTER_AXI_ARVALID is asserted.
MASTER_AXI_ARLEN	Input	8	AXI_CLK	Indicates the number of beats (data transfer cycles) associated with the read burst. This information is valid when MASTER_AXI_ARVALID is asserted.
MASTER_AXI_ARSIZE	Input	3	AXI_CLK	Burst size. This signal indicates the size of each transfer in the burst. All bytes in the current transfer size are read.
MASTER_AXI_ARUSER	Input	88	AXI_CLK	Sideband descriptor. **AXI Outbound Access through Sideband Descriptor**" for detailed description.
MASTER_AXI_ARVALID	Input	1	AXI_CLK	Valid signal for the address and associated parameters on the AXI master read interface. The client must keep this valid signal asserted until the PCIe Controller sets the ready input (MASTER_AXI_ARREADY).
MASTER_AXI_ARREADY	Output	1	AXI_CLK	Ready signal from the PCIe Controller to the client, indicating that the PCIe Controller is ready to sample the address and associated parameters from the master read interface. The address and associated parameters are transferred across the interface when MASTER_AXI_ARVALID and MASTER_AXI_ARREADY are both high in a clock cycle.

Table 65: AXI Slave Interface Read Data Channel

Signal	Direction	Width	Clock Domain	Description
MASTER_AXI_RREADY	Input	1	AXI_CLK	Ready for read response status from the client. The client must assert this signal when it ready to accept the read response status from the PCIe Controller. PCIe Controller core keeps MASTER_AXI_RVALID asserted until it samples this ready signal high on a positive edge of the clock.
MASTER_AXI_RDATA	Output	256	AXI_CLK	Data associated with a memory read operation delivered by the PCIe Controller to the client. Data is transferred in little-endian order. Data is transferred in aligned fashion. The data on this bus is valid when MASTER_AXI_RVALID is high.
MASTER_AXI_RDATA_PAR	Output	32	AXI_CLK	Contains the end-to-end parity for MASTER_AXI_RDATA. Odd parity is computed for every byte of the data and propagated through the PCIe Controller for end-to-end parity protection.
MASTER_AXI_RID	Output	8	AXI_CLK	The PCIe Controller places the 4-bit read ID tag associated with the read request when returning data/completion status to the client. The ID on this bus is valid when MASTER_READ_REPONSE_VALID is high.
MASTER_AXI_RID_PAR	Output	1	AXI_CLK	Contains the end-to-end parity for MASTER_AXI_RID. Odd parity is computed for every byte of the data and propagated through the PCIe Controller for end-to-end parity protection.
MASTER_AXI_RLAST	Output	1	AXI_CLK	The PCIe Controller asserts this signal in the last beat of the burst to indicate the end of the read burst.
MASTER_AXI_RRESP	Output	2	AXI_CLK	Read completion status from client. Valid status encoding are: 2'b00: Good completion 2'b10: Slave error, completion error 2'b11: Decode error
MASTER_AXI_RRESP_PAR	Output	1	AXI_CLK	Contains the end-to-end parity for MASTER_AXI_RRESP.. Odd parity is computed for every byte of the data and propagated through the PCIe Controller for end-to-end parity protection.
MASTER_AXI_RUSER	Output	7	AXI_CLK	[2:0] Completion error code. [4:3] Completion status code. [5] If 1, uncorrectable error in the AXI reorder RAM or completion RAM. [6] If 1, AXI slave read/write addresses may not match any of the AXI base address programmed in the outbound region. See **Appendix B: Error Handling** on page 112.
MASTER_AXI_RUSER_PAR	Output	1	AXI_CLK	Contains the end-to-end parity for MASTER_AXI_RUSER. Odd parity is computed for every byte of the data and propagated through the PCIe Controller for end-to-end parity protection.
MASTER_AXI_RVALID	Output	1	AXI_CLK	Valid for read response status from the PCIe Controller to the client. The assertion of this signal indicates that the PCIe Controller is ready to transfer data in response to a read request it received from the client.

Interrupt Interface Signals

Table 66: Interrupt interface

Signal	Direction	Width	Clock Domain	Description
LOCAL_INTERRUPT	Output	1	AXI_CLK	Active-high local error and status register interrupt. Asserted until software clears the Local Error and Status Register.
INTERRUPT_SIDEBAND_ SIGNALS	Output	28	AXI_CLK	Signal that causes local interrupt as sideband. See **Interrupt Sideband Signals**.

Signal	Direction	Width	Clock Domain	Description
INTA_IN	Input	1		When the PCIe Controller is configured as an endpoint, the client uses this input to signal an interrupt from any of its PCI functions to the root port using legacy PCIs interrupts. This input corresponds to the PCI bus INTA. Asserting this signal causes the PCIe Controller to send an Assert_INT*x* message; de-asserting it causes the PCIe Controller to transmit a Deassert_INT*x* message.
INTB_IN	Input	1	AXI_CLK	When the PCIe Controller is configured as an endpoint, the client uses this input to signal an interrupt from any of its PCI functions to the root port using Legacy PCI Express Interrupt Delivery. This input corresponds to the PCI bus INTB. Asserting this signal causes the PCIe Controller to send an Assert_INT*x* message; de-asserting it causes the PCIe Controller to transmit a Deassert_INT*x* message.
INTC_IN	Input	1	AXI_CLK	When the PCIe Controller is configured as an endpoint, the client uses this inputs to signal an interrupt from any of its PCI functions to the root port using Legacy PCI Express Interrupt Delivery. This input corresponds to Ithe PCI bus INT. Asserting this signal causes the PCIe Controller to send an Assert_INT*x* message; de-asserting it causes the PCIe Controller to transmit a Deassert_INT*x* message.
INTD_IN	Input	1	AXI_CLK	When the PCIe Controller is configured as an endpoint, the client uses this inputs to signal an interrupt from any of its PCI functions to the root port using Legacy PCI Express Interrupt Delivery. This input corresponds to the PCI bus INTD. Asserting this signals causes the PCIe Controller to send an Assert_INT*x* message; de-asserting it causes the PCIe Controller to transmit a Deassert_INT*x* message.
INT_PENDING_STATUS	Input	4	AXI_CLK	When using legacy interrupts, this input indicates the PF interrupt pending status. The i input must be set when an interrupt is pending in function i.
MSI_PENDING_STATUS_IN	Input	128	AXI_CLK	These inputs provide the status of the MSI pending interrupts for the PFs from the client to the PCIe Controller. If the MSI Pending Status In Mode Select field is set to 1 in the Debug Mux Control 2 Register in local management space, these pin settings determine the value read from the MSI Pending Bits Register of the corresponding PF. Bits [31:0] belong to PF0 , bits [63:32] to PF1, and so on.
INT_ACK	Output	1	AXI_CLK	A pulse on this output indicates that the PCIe Controller has sent an Assert_INT*x* or Deassert_INT*x* message in response to a change in the state of one of the INT*x* inputs.
INTA_OUT	Output	1	AXI_CLK	When the PCIe Controller is configured as a root port, this output emulates the PCI legacy interrupt INTA. The PCIe Controller asserts an interrupt output when it has received an Assert_INT*x* message from the link, and deasserts it when it receives a Deassert_INT*x* message.
INTB_OUT	Output	1	AXI_CLK	When the PCIe Controller is configured as a root port, this output emulates the PCI legacy interrupt INTB. The PCIe Controller asserts an interrupt output when it has received an Assert_INT*x* message from the link, and deasserts it when it receives a Deassert_INT*x* message.
INTC_OUT	Output	1	AXI_CLK	When the PCIe Controller is configured as a root port, this output emulates the PCI legacy interrupt INTC. The PCIe Controller asserts an interrupt output when it has received an Assert_INT*x* message from the link, and deasserts it when it receives a Deassert_INT*x* message.
INTD_OUT	Output	1	AXI_CLK	When the PCIe Controller is configured as a root port, this output emulates the PCI legacy interrupt INTD. The PCIe Controller asserts an interrupt output when it has received an Assert_INT*x* message from the link, and deasserts it when it receives a Deassert_INT*x* message.

Message Interface Signals

Table 67: Message interface

Signal Name	Direction	Width	Clock Domain	Descriptions
MSG	Output	256	AXI_CLK	Inbound message interface data bus.
MSG_BYTE_EN	Output	32	AXI_CLK	Indicates which bytes of MSG are valid.
MSG_DATA	Output	1	AXI_CLK	Indicates that MSG contains message data.
MSG_END	Output	1	AXI_CLK	Indicates that MSG contains the last stripe of a message.
MSG_PASID	Output	22	AXI_CLK	PASID value.
MSG_PASID_PRESENT	Output	1	AXI_CLK	When asserted with MSG_START, indicates the presence of PASID.
MSG_START	Output	1	AXI_CLK	Indicates that MSG contains the start of a message i.e., a message header.
MSG_VALID	Output	1	AXI_CLK	Indicates that MSG_<*name*> signals are valid.
MSG_VDH	Output	1	AXI_CLK	Indicates that MSG contains a vendor-defined message header.

Status and Error Indicator Signals

Table 68: Status and Error Indicator

Signal Name	Direction	Width	Clock Domain	Descriptions
LTSSM_STATE	Output	6	AXI_CLK	LTSSM state.
REG_ACCESS_CLK_SHUTOFF	Output	1	USER_APB_CLK	Pulse indicating an APB access when the internal core clock was not running.
CORE_CLK_SHUTOFF	Output	1	USER_APB_CLK	Level signal indicating that the core clock is not running.
LINK_STATUS	Output	2	AXI_CLK	Status of the PCIe link. 00: No receivers detected. 01: Link training in progress. 10: Link up, DL initialization in progress. 11: Link up, DL initialization completed.
FUNCTION_STATUS	Output	16	AXI_CLK	These outputs indicate the states of each function's command register bits in the PCI configuration space. Used to enable requests and completions from the host. [0] Function 0 I/O space enable [1] Function 0 memory space enable [2] Function 0 bus master enable [3] Function 0 INT*x* disable [4] Function 1 I/O space enable [5] Function 1 memory space enable [6] Function 1 bus master enable [7] Function 1 INT*x* disable and so on.

Signal Name	Direction	Width	Clock Domain	Descriptions
PCIE_MAX_READ_REQ_SIZE	Output	3	AXI_CLK	The maximum request size field programmed in the PCI Express Device Control Register. When using multiple functions, this output provides the minimum of the maximum read-request field in the PFs' Device Control Registers. The client must limit the size of outgoing read requests to this value. The 3-bit codes are the same as those defined in the PCIe specifications: 000: 128 bytes 001: 256 bytes 010: 512 bytes 011: 1,024 bytes 100: 2,048 bytes 101: 4,096 bytes
PCIE_MAX_PAYLOAD_SIZE	Output	3	AXI_CLK	The maximum payload size field programmed in the PCI Express Device Control Register. When using multiple fuctions, this output provides the minimum of the maximum payload-size field in the PFs' Device Control Registers. The client must limit the size of outputgoing completion payloads to this value. The 3-bit codes are the same as those defined in the PCIe specifications: 000: 128 bytes 001: 256 bytes 010: 512 bytes 011: 1024 bytes 100: 2048 bytes 101: 4096 bytes
ATS_PR_CONTROL_REG_ RESET	Output	4	AXI_CLK	Reset per PF. When the enable field is clear (or is being cleared during the same register update that sets this field) writing a 1b to this field clears the associated implementation dependent page request credit counter and pending request state for the associated page request interface. If this field is written with 0b or if it is written with any value while the enable field is set, no action is taken. Reads of this field return 0b.
ATS_PR_CONTROL_REG_ ENABLE	Output	4	AXI_CLK	Indicates that the page request interface can to make page requests. If this field is clear, the page request interface is not allowed to issue page requests. If this field and the stopped field are both clear, the page request interface does not issue new page requests. Instead, it has outstanding page requests that have been transmitted or are queued for transmission. When the page request interface is transitioned from not-enabled to enabled, its status flags (stopped, response failure, and unexpected response flags) are cleared. Enabling a page request interface that has not successfully stopped has indeterminate results. Default value is 0b.
Q0_PMA_CMN_READY	Output	1	Async	Common ready.
Q0_PIPE_P00_RATE	Output	2	Static	PIPE link signaling rate. Selects the data rate. 2'b00: PCIe Gen1 2'b01: PCIe Gen2 2'b10: PCIe Gen3 2'b11: PCIe Gen4

Signal Name	Direction	Width	Clock Domain	Descriptions
CORRECTABLE_ERROR_IN	Input	1	AXI_CLK	The client can activate this input for one clock cycle to indicate that the client detected a correctable error that needs to be reported as an internal error through using PCIe advanced error reporting. In response, the PCIe Controller sets the Corrected Internal Error Status bit in the enabled function(s)' AER Correctable Error Status Register. In endpoint mode it also sends an error message if enabled to do so. This error is not function specific.
UNCORRECTABLE_ERROR_IN	Input	1	AXI_CLK	The client can activate this input for one clock cycle to indicate that the client detected an uncorrectable error that needs to be reported as an internal error through using PCIe advanced error reporting. In response, the PCIe Controller sets the Uncorrected Internal Error Status bit in the enabled function(s)' AER Correctable Error Status Register. In endpoint mode it also sends an error message if enabled to do so. This error is not function specific.
FATAL_ERROR_OUT	Output	1	AXI_CLK	This output is a single clock cycle for endpoints. Endpoints: The PCIe Controller activates this output for one clock cycle when it detects a fatal error and its reporting is not masked. When using multiple functions, it is the logical OR of the fatal error status bits in the function(s)' Device Status Registers.
NON_FATAL_ERROR_OUT	Output	1	AXI_CLK	This output is a single clock cycle for endpoints. Endpoints: The PCIe Controller activates this output for one clock cycle when it detects a non-fatal error and its reporting is not masked. When using multiple functions, it is the logical OR of the non-fatal error status bits in the function(s)' Device Status Registers.
CORRECTABLE_ERROR_OUT	Output	1	AXI_CLK	This output is a single clock cycle for endpoints. Endpoints: The PCIe Controller activates this output for one cycle when it detects a correctable error and its reporting is not masked. When using multiple functions, it is the logical OR of the correctable error status bits in the function(s)' Device Status Registers.
PHY_INTERRUPT_OUT	Output	1	AXI_CLK	Active-high, level-interrupt output. The PCIe Controller asserts this output in the root port mode to signal a link training-related event: Local change: The link bandwidth changed because the link width or operating speed changed, and the change was initiated locally (not by the link partner) without the link going down. This interrupt is enabled by the Link Bandwidth Management Interrupt Enable bit in the Link Control Register. You can read the this interrupt's status in the Link Bandwidth Management Status bit of the Link Status Register. Automomous change: The link bandwidth changed autonomously because the link width or operating speed changed and the change was initiated by the remote node. This interrupt is enabled by the Link Autonomous Bandwidth Interrupt Enable bit in the Link Control Register. You can read this interrupt's status from the Link Autonomous Bandwidth Status bit of the Link Status Register. This signal is not active when the PCIe Controller is configured as an endpoint.

Function-Level Reset Signals

These signals are only used in endpoint mode. Refer to **Function-Level Reset (FLR)** on page 64 for a more comprehensive overview of handshake.

Table 69: Function Level Reset

Signal Name	Direction	Width	Clock Domain	Descriptions
FLR_IN_PROGRESS	Output	4	AXI_CLK	There are four PFs in the PCIe Controller, where FLR_IN_PROGRESS[0, 1, 2, 3] correlates to PF[0, 1, 2, 3], respectively. The PCIe Controller asserts FLR_IN_PROGRESS[*n*] to indicate the PF that received FLR. Example: When the PCIe Controller asserts FLR_IN_PROGRESS[3], it signals that PF[3] is currently performing an FLR. FLR_IN_PROGRESS[3] remains asserted until the PCIe Controller receives the FLR_DONE[3] assertion from the user. The subsequent de-assertion of FLR_IN_PROGRESS[3] indicates that the FLR process for PF3 has finished. **Note:** When a PF undergoes an FLR, all the associated VFs undergo the FLR too.
VF_FLR_IN_ PROGRESS	Output	64	AXI_CLK	There are 64 VFs in the PCIe Controller, where VF_FLR_IN_PROGRESS[0, 1, ..., 63] correlates to VF[0, 1, …, 63], respectively. The PCIe Controller asserts VF_FLR_IN_PROGRESS[*n*] to indicate the *n*th VF received FLR. VF_FLR_IN_PROGRESS[*n*] remains asserted until the PCIe Controller receives the VF_FLR_DONE[*n*] assertion from the user. The de-assertion of VF_FLR_IN_PROGRESS[*n*] marks the completion of FLR in the *n*th VF.
FLR_DONE	Input	4	AXI_CLK	When FLR_IN_PROGRESS[*n*] asserts, you need to clear any pending transactions associated with the PF/VF being reset. Then, assert FLR_DONE[*n*] and hold FLR_DONE[*n*] high until FLR_IN_PROGRESS[*n*] finishes de-asserting.
VF_FLR_DONE	Input	64	AXI_CLK	When VF_FLR_IN_PROGRESS[*n*] asserts, you need to clear any pending transactions associated with the VF being reset. Then, assert VF_FLR_DONE[*n*] and hold VF_FLR_DONE[*n*] high until FLR_IN_PROGRESS[*n*] finishes de-asserting.

Configuration Snoop Interface Signals

Table 70: Configuration Snoop Interface

Signal	Direction	Width	Clock Domain	Description
CONFIG_READ_DATA	Input	32	AXI_CLK	The client can provide data from an externally implemented configuration register to the PCIe Controller using this bus. If the client has set CONFIG_READ_DATA_VALID, the PCIe Controller samples this data on the next positive clock edge after it sets CONFIG_READ_RECEIVED high.
CONFIG_READ_DATA_PAR	Input	4	AXI_CLK	Contains the end-to-end parity for CONFIG_READ_DATA.

Signal	Direction	Width	Clock Domain	Description
CONFIG_READ_DATA_VALID	Input	1	AXI_CLK	The client asserts this input to the PCIe Controller to supply data from an externally implemented configuration register. The PCIe Controller samples this input data on the next positive clock edge after it sets CONFIG_READ_RECEIVED high. If the PCIe Controller detects this input is asserted, it uses the data supplied on the CONFIG_READ_DATA bus as the completion payload for the received configuration read request. You can extend the timing of this signal by programming the Debug Mux Control 2 Register. See **Configuration Snoop Interface** on page 80 for timing diagrams.
CONFIG_READ_RECEIVED	Output	1	AXI_CLK	The PCIe Controller generates a one clock cycle pulse on this output when receives a configuration read request from the link.
CONFIG_REG_NUM	Output	10	AXI_CLK	The 10-bit address of the configuration register being read or written. The data is valid when CONFIG_READ_RECEIVED or CONFIG_WRITE_RECEIVED is high.
CONFIG_WRITE_BYTE_ENABLE	Output	4	AXI_CLK	Byte enables for a configuration write transaction.
CONFIG_WRITE_BYTE_ENABLE_PAR	Output	1	AXI_CLK	Contains the end-to-end parity for CONFIG_WRITE_BYTE_ENABLE.
CONFIG_WRITE_DATA	Output	32	AXI_CLK	Data being written to a configuration register. This output is valid when CONFIG_WRITE_RECEIVED is high.
CONFIG_WRITE_DATA_PAR	Output	4	AXI_CLK	Contains the end-to-end parity for CONFIG_WRITE_DATA.
CONFIG_WRITE_RECEIVED	Output	1	AXI_CLK	The PCIe Controller generates a one clock cycle pulse on this output when it has received configuration write request from the link.
CONFIG_FUNCTION_NUM	Output	8	AXI_CLK	Function number.

Vendor Specific (VSEC) Interface Signals

Table 71: Vendor Specific (VSEC) Interface

Signal	Direction	Width	Clock Domain	Description
F0_VSEC_CONTROL_IN	Input	8	AXI_CLK	Read the input state from Vendor-Specific Control Register bits [7:0] in the PF0 Vendor-Specific Capability Structure. The setting does not affect the operation of the PCIe Controller.
F1_VSEC_CONTROL_IN	Input	8	AXI_CLK	Read the input state from Vendor-Specific Control Register bits [7:0] in the PF1 Vendor-Specific Capability Structure. The setting does not affect the operation of the PCIe Controller.
F2_VSEC_CONTROL_IN	Input	8	AXI_CLK	Read the input state from Vendor-Specific Control Register bits [7:0] in the PF2 Vendor-Specific Capability Structure. The setting does not affect the operation of the PCIe Controller.
F3_VSEC_CONTROL_IN	Input	8	AXI_CLK	Read the input state from Vendor-Specific Control Register bits [7:0] in the PF3 Vendor-Specific Capability Structure. The setting does not affect the operation of the PCIe Controller.

Signal	Direction	Width	Clock Domain	Description
F0_VSEC_INTERRUPT_OUT	Output	1	AXI_CLK	Driven by Vendor-Specific Control Register bit [8] in the PF0 Vendor-Specific Capability Structure. The host can use it to signal a software-driven interrupt to the application logic outside the PCIe Controller.
F1_VSEC_INTERRUPT_OUT	Output	1	AXI_CLK	Driven by Vendor-Specific Control Register bit [8] in the PF1 Vendor-Specific Capability Structure. The host can use it to signal a software-driven interrupt to the application logic outside the PCIe Controller.
F2_VSEC_INTERRUPT_OUT	Output	1	AXI_CLK	Driven by Vendor-Specific Control Register bit [8] in the PF2 Vendor-Specific Capability Structure. The host can use it to signal a software-driven interrupt to the application logic outside the PCIe Controller.
F3_VSEC_INTERRUPT_OUT	Output	1	AXI_CLK	Driven by Vendor-Specific Control Register bit [8] in the PF3 Vendor-Specific Capability Structure. The host can use it to signal a software-driven interrupt to the application logic outside the PCIe Controller.

Power Management Interface Signals

Refer to **Power Management** on page 66 for a description of the PCIe Controller's power management capabilities.

Table 72: Power Management interface

Signal	Direction	Width	Clock Domain	Description
CLIENT_REQ_EXIT_L1	Input	1	Asynchronous	This signal triggers an exit to L0 from L1 or from L1-substates. This signal can also block L1 entry in endpoint mode. The client can trigger an explicit L1 exit by asserting this signal. You can drive this signal from the PM_CLK, core clock, or AXI_CLK domains, depending on hyour configuration. It is synchronized inside the PCIe Controller before use.
CLIENT_REQ_EXIT_L2	Input	1	AXI_CLK	The client can only assert this input in the short interval of time after the link enters L2 and before the system is powered off. While the power and clocks are on, the client can assert this input to initiate an exit from L2.Idle detect.
REQ_PM_TRANSITION_L23_READY	Input	1	AXI_CLK	In the PCIe Controller is in endpoint mode, the client can assert this input to transition the PCIe Controller's power management state to L23_READY (see PCIe specifications chapter 5 for a detailed description of power management). This transition happens after the PCIe Controller's functions are in the D3 state and after the client has acknowledged the PME_Turn_Off message from the root rort. Asserting this input causes the link to transition to the L2 state and requires a power-on reset to resume operation. You can hardwire this signal to 0 if the link does not need to transition to L2. This input is not used in the root port mode.

Signal	Direction	Width	Clock Domain	Description
POWER_STATE_CHANGE_ACK	Input	1	AXI_CLK	When it is ready to transition to the low-power state requested by the configuration write request, the client must assert this input for one clock cycle in response to the assertion of POWER_STATE_CHANGE_INTERRUPT. The client can keep this input high if it does not need to delay the return of the completions for the configuration write transactions causing power-state changes.
FUNCTION_POWER_STATE	Output	12	AXI_CLK	These outputs provide the current power state of the PFs. Bits [2:0] capture the power state of function 0, bits [5:3] capture that of function 1, and so on. The possible power states are: 000: D0_uninitialized 001: D0_active 010: D1 100: D3hot
PCIE_LINK_POWER_STATE	Output	4	AXI_CLK	Current power state of the PCIe link. 0001: L0 0010: L0s 0100: L1 1000: L2
POWER_STATE_CHANGE_ INTERRUPT	Output	1	AXI_CLK	The PCIe Controller asserts this output when the power state of a PF or VF is changing to the D1 or D3 states by a write into its Power Management Control Register. The PCIe Controller keeps this output high until the client asserts the POWER_STATE_CHANGE_ACK input. While this signal is high, the the PCIe Controller will not return completions for any pending configuration read or write transactions it receives. The intent is to delay the completion for the configuration write transaction that caused the state change until the client is ready to transition to the low power state. When this signal is high, the function number associated with the configuration write transaction is provided on the POWER_STATE_CHANGE_FUNCTION_ NUM[7:0] output. When the client asserts POWER_STATE_CHANGE_ACK, the new state of the function that underwent the state change is reflected on the PCIe Controller's FUNCTION_POWER_STATE (for PFs) or the VF_POWER_STATE (for VFs) outputs.
POWER_STATE_CHANGE_ FUNCTION_NUM	Output	8	AXI_CLK	Number of the function for which a power state change occurred.
DPA_INTERRUPT	Output	4	AXI_CLK	The PCIe Controller generates a one clock cycle pulse on one of these outputs when a configuration write transaction writes into the Dynamic Power Allocation Control Register to modify the device's DPA power state. [0] A pulse indicates a DPA event for PF0. [1] A pulse indicates a DPA event for PF1 and so on. The endpoitnt's software running must read the corresponding function's DPA control register to determine the DPA substate requested by the host and set the device's power state of the device. These outputs are not active in root port mode.

L1 Interface Signals

Table 73: L1 Interface

Signal	Direction	Width	Clock Domain	Description
CLIENT_REQ_EXIT_L1	Input	1	Async	Client logic can trigger an explicit L1 exit by asserting this signal. This signal triggers an exit to L0 from L1 or from L1 substates. This signal can also be used to block L1 entry in endpoint mode. You can drive this signal from the PM_CLK, core clock, or AXI_CLK domain depending on your configuration. It is synchronized inside the PCIe Controller before use.

L1 Substate Interface Signals

Refer to **L1 Power Substates** on page 69 for a description of the L1 substate interface

Table 74: L1 Substrate Interface

Signal	Direction	Width	Clock Domain	Description
CLKREQ_IN_N	Input	1	Async	This asynchronous input must be connected to the shared CLKREQ# bus, so that its state reflects the combined effect of the upstream and downstream interfaces' CLKREQ# outputs. The PCIe Controller samples this input on the positive edge of PM_CLK.
CLIENT_REQ_EXIT_L1_SUBSTATE	Input	1	PM_CLK	Client logic can trigger an explicit L1 substate exit by asserting this signal. This signal triggers an exit from L1 substates to L0 if the PCIe Controller is already in an L1 substate. The PCIe Controller waits in L1 state for this signal to be de-asserted before entering an L1 substate. The PCIe Controller responds to normal L1 exit triggers while it waits for this signal to de-assert.
L1_PM_SUBSTATE_OUT	Output	3	PM_CLK	This output provides the current state of the L1 PM substates state machine. This output is in the PM_CLK clock domain. Its encodings are: 000: L1-substate machine not active 001: L1.0 substate. Shown after the delay programmed in the L1 substate entry delay field in the Low Power Debug Control Register 0 010: L1.1 substate 011: Reserved 100: L1.2.Entry substate 101: L1.2.Idle substate 110: L1.2.Exit substate 111: Reserved
CLKREQ_OUT_N	Output	1	PM_CLK	The PCIe Controller asserts this output in the L1 substates when the core clock can be turned off. You drive this output from the PM_CLK clock domain. You can use it to enable the tri-state driver for the device's CLKREQ# output.

APB Interface Signals

Table 75: APB interface

Signal	Direction	Width	Clock Domain	Description
USER_APB_PADDR	Input	24	USER_APB_CLK	APB address bus. The address is the byte address of the PCIe configuration space or local management space registers.
USER_APB_PSEL	Input	1	USER_APB_CLK	Select. It indicates that the slave device is selected and that a data transfer is required. Each slave has a PSEL*x* signal.
USER_APB_PENABLE	Input	1	USER_APB_CLK	Enable. This signal indicates the second and subsequent cycles of an APB transfer.
USER_APB_PWRITE	Input	1	USER_APB_CLK	Read or writee access. High: APB write access. Low: APB read access.
USER_APB_PWDATA	Input	32	USER_APB_CLK	Write data. Only used when USER_APB_PWRITE is high.
USER_APB_PWDATA_PAR	Input	4	USER_APB_CLK	Contains the end-to-end parity for USER_APB_PWDATA.
USER_APB_PSTRB	Input	4	USER_APB_CLK	Write the strobe signal to enable sparse data transfer on the write data bus.
USER_APB_PSTRB_PAR	Input	1	USER_APB_CLK	Contains the end-to-end parity for USER_APB_PSTRB.
USER_APB_PRDATA	Output	32	USER_APB_CLK	Read data. Only applies when USER_APB_PWRITE is low.
USER_APB_PRDATA_PAR	Output	4	USER_APB_CLK	Contains the end-to-end parity for USER_APB_PRDATA. Odd parity is computed for every byte of the data and propagated through the PCIe Controller for end-to-end parity protection.
USER_APB_PREADY	Output	1	USER_APB_CLK	Ready. The slave uses this signal to extend an APB transfer.

Appendix A: Acronyms and Abbreviations

Table 76: Acronyms and Abbreviations

Term	Definition
AER	Advanced Error Reporting
ARI	Alternative Routing-ID Interpretation
ASPM	Active-state power management
AXI	Advanced eXtensible Interface
EP	Endpoint
PCIe	Peripheral Component Interface Express
PNP	Posted/Non-Posted
SOP	Start Of Packet
EOP	End Of Packet
CRC	Cyclic Redundancy Check
LCRC	Link Cyclic Redundancy Check
DLLP	Data Link Layer Packet
LTSSM	Link Training and Status State Machine
TLP	Transaction Layer Packet
AER	Advanced Error Reporting
FLR	Function-Level Reset
PBA	Pending Bit Array
MSI	Message-Signaled Interrupt
OS	Ordered Set
ASPM	Active State Power Management
UR	Unsupported request
LTR	Latency Tolerance Reporting
PTM	Precision Time Measurement
RP	Root Port
OBFF	Optimized Buffer Flush/Fill
IDO	ID-based Ordering
VC	Virtual Channel
Register notation	
R	Read Only for Software Root Complex.
RW	The software can read or write. Write access to configuration registers through the local management interface.
WOCLR	Software has to write a 1 to clear. The PCIe Controller sets the bit and software clears the bit.
R/WOCLR	The software can read or write. The software has to write a 1 to clear. The PCIe Controller sets the bit and software clears the bit.
W	Write Only Register Field. A read will return 00s.

Appendix B: Error Handling

The PCIe Controller has the following registers to report error status:

Table 77: Error Status Registers

Register	Location	Per Function?
Status Register	PCI-Compatible Configuration Space	Yes
Device Status Register	PCI Express Capability Structure	Yes
AER Uncorrectable Error Status Register	PCIe Configuration Space	Yes
AER Correctable Error Status Register	PCIe Configuration Space	Yes
Local Error Status Register	Local Management Space	N.A. Reports errors that are not covered in PCIe configuration space.

The PCIe Controller implements the following to report error conditions:

- NON_FATAL_ERROR_OUT—This pulse output is asserted if the PCIe Controller detects an unmasked AER uncorrectable error with non-fatal severity.
- FATAL_ERROR_OUT—This pulse output is asserted if the PCIe Controller detects an unmasked AER uncorrectable error with fatal severity.
- CORRECTABLE_ERROR_OUT—This pulse output is asserted if the PCIe Controller detects an unmasked AER correctable error.
- LOCAL_INTERRUPT—This is level-output is asserted if the PCIe Controller detects an unmasked error in the Local Error Status Register.

In endpoint mode, the PCIe Controller transmits the appropriate `ERR_COR`, `ERR_NON_FATAL`, or `ERR_FATAL` error messages when it detects an unmasked AER error.

Non-Fatal Errors

In some cases the agent that detects a non-fatal error is not the most appropriate one to determine whether the error is recoverable or not, or if it even needs a recovery action. The PCIe Controller handles the following errors as advisory non-fatal as recommended by the PCIe specification:

- Unsupported Non-Posted Request Received
- Unexpected Completion Received
- Poisoned Completion TLP Received
- Poisoned Vendor Defined Msg with Data TLP Received
- Poisoned MWr, IOWr, or MsgD Request TLP received. Per PCIe specification section 2.7.2.2, if these requests target a control register or control structure, they must be handled as uncorrectable and not as advisory non-fatal. The PCIe Controller cannot determine if these requests target a control or a data structure in the system., Therefore, it uses the Poisoned TLP Received Advisory Non-Fatal bit in the Debug Mux Control 2 Register as follows:
  - When 0 (default), the PCIe Controller reports poisoned MWr, IOWr, MsgD as uncorrectable.
  - When 1, the PCIe Controller reports poisoned MWr, IOWr, MsgD as advisory non-fatal.
- Completion Timeout (with Completion Timeout Advisory Non-Fatal bit set to 1 in the Debug Mux Control 2 Register). A completion timeout should always be reported as advisory non-fatal as recommended by the PCIe specification. The Completion

Timeout Advisory Non-Fatal bit in the Debug Mux Control 2 Register is provided only for debugging purposes. You should not change this bit from its default value.

Table 78: Error Handling: Advisory Non-Fatal Errors

Except as noted, these errors result in an ERROR_OUT of FATAL/ CORRECTABLE_ERROR_OUT, based on the severity and mask.

Error Case	AXI Interface Response	Error Status Registers and Bits	Client Action	Local Interrupt
Unsupported Non-Posted Request Received	Not delivered to the PCIe Controller's AXI interface. The PCIe Controller detects this error internally and responds with UR status.	Unsupported Request Error Status bit in AER Uncorrectable Error Status Register.	None required.	No
Poisoned Completion TLP Received	Reports SLVERR to the AXI slave MASTER_AXI_RRESP.	Poisoned TLP Status bit in AER Uncorrectable Status Register. Detected parity Error bit in Command Status Register.	Client should ignore the completion data because it is poisoned. Client can retry the read request after the read data is completely received.	No
Poisoned MWr, MsgD Request TLP Received (Poisoned TLP Received Advisory Non-Fatal bit set to 1 in Debug Mux Control 2 Register)	Not delivered to the PCIe Controller's AXI interface. The request is discarded internally.	Poisoned TLP Status bit in AER Uncorrectable Status Register. Detected parity Error bit in Command Status Register.	None required.	No
Poisoned IOWr Request TLP Received (Poisoned TLP Received Advisory Non-Fatal bit set to 1 in Debug Mux Control 2 Register)	Not delivered to the PCIe Controller's AXI interface. The PCIe Controller responds internally with UR status.	Poisoned TLP Status bit in AER Uncorrectable Status Register. Detected parity Error bit in Command Status Register.	None required.	No
Poisoned Vendor Defined Msg with Data TLP Received	Not delivered to the PCIe Controller's AXI interface. The request is discarded internally.	Poisoned TLP Status bit in AER Uncorrectable Status Register. Detected parity Error bit in Command Status Register.	None required.	No
Completion Timeout (Completion Timeout Advisory Non-Fatal bit set to 1 in Debug Mux Control 2 Register)	Reports SLVERR to the AXI slave MASTER_AXI_RRESP.	Completion Timeout Status bit in AER Uncorrectable Error Status Register and Local Error Status Register.	Client can retry the request.	Yes
Unexpected Completion Received	Not delivered to the PCIe Controller's AXI interface.	Unexpected Completion Received Status bit in AER Uncorrectable Error Status Register and Local Error Status Register.	None required.	Yes
Requester Received Completion with CA status Error out: NIL	Reports SLVERR to the AXI slave MASTER_AXI_RRESP.	Received Target Abort Status bit in Command and Status Register.	Client should ignore the read data.	No

Table 79: Error Handling: Uncorrectable Errors

These errors result in an ERROR_OUT of FATAL/NON_FATAL_ERROR_OUT, based on the severity and mask.

Error Case	AXI Interface Response	Error Status Registers and Bits	Client Action	Local Interrupt?
Unsupported Posted Request Received	Not delivered to the PCIe Controller's AXI interface. The PCIe Controller detects this error internally and responds with UR status.	Unsupported Request Error Status bit in AER Uncorrectable Error Status Register.	None required.	No

Error Case	AXI Interface Response	Error Status Registers and Bits	Client Action	Local Interrupt?
Poisoned CfgWr Request TLP Received	Not delivered to the PCIe Controller's AXI interface. The PCIe Controller detects this error internally and responds with UR status.	Poisoned TLP Status bit in AER Uncorrectable Status Register. Detected Parity Error bit in Command Status Register.	None required.	No
Poisoned IOWr Request TLP Received (Poisoned TLP Received Advisory Non-Fatal bit set to 0 in Debug Mux Control 2 Register)	Not delivered to the PCIe Controller's AXI interface. The PCIe Controller detects this error internally and responds with UR status.	Poisoned TLP Status bit in AER Uncorrectable Status Register. Detected Parity Error bit in Command Status Register.	None required.	No
Poisoned MWr, MsgD Request TLP Received (Poisoned TLP Received Advisory Non-Fatal bit set to 0 in Debug Mux Control 2 Register)	Not delivered to the PCIe Controller's AXI interface. Request is discarded internally.	Poisoned TLP Status bit in AER Uncorrectable Status Register. Detected Parity Error bit in Command Status Register.	None required.	No
Uncorrectable RAM ECC Errors	RAM ECC Errors in the RX path result in SLVERR on the AXI interface. On TX path, the AXI response can be SLVERR or OK.	Uncorrectable Internal Error bit in AER Uncorrectable Error Status Register and bits in Local Error Status Register.	Reset upon interrupt.	Yes
Request with Unmapped TC Received	Not delivered to the PCIe Controller's AXI interface. Internally handled as a malformed TLP request.	Malformed TLP Received Status bit in AER Uncorrectable Error Status Register.	None required.	Yes
PNP RX FIFO Overflow	The AXI interface does not receive the packet that caused the overflow.	PNP RX FIFO Overflow bits in Local Error Status Register. Receiver Overflow Status bit in AER Uncorrectable Error Status register.	Reset	Yes
Completion RX FIFO Overflow	The AXI interface drops the completion that caused the overflow.	Receiver Overflow Status bit in AER Uncorrectable Error Status Register. Receiver Overflow Status bit in AER Uncorrectable Error Status register.	Reset	Yes
End to End Parity Error on Transmit Path	If an outbound TLP has an end-to-end parity error, the TLP is dropped internally or nullified. The AXI interface does not respond with SLVERR for the same TLP at the AXI slave.	Uncorrectable Internal Error bit in AER Uncorrectable Error Status Register. End to End Parity Error in Local Error Status Register.	Reset upon interrupt.	Yes
End to End Parity Error on Receive Path	If an inbound TLP has a end-to-end parity error, the PCIe Controller forwards the TLP to the AXI master with errored parity.	Uncorrectable Internal Error bit in AER Uncorrectable Error Status Register. End to End Parity Error in Local Error Status Register.	Reset upon interrupt.	Yes
ECRC Error	Not delivered to the PCIe Controller's AXI interface. Request is discarded internally.	ECRC Error Status bit in AER Uncorrectable Error Status Register.	None required.	No
Flow Control Error	No effect.	Flow Control Error bit in AER Uncorrectable Error Status Register and in Local Error Status Register.	None required.	Yes
Malformed TLP Received	Not delivered to the PCIe Controller's AXI interface. Request is discarded internally.	Malformed TLP Received Status bit in AER Uncorrectable Error Status Register.	None required.	Yes
Data Link Protocol Error Status	No effect.	Data Link Protocol Error Status bit in AER Uncorrectable Error Status Register.	None required.	No

Table 80: Error Handling: Correctable Errors

These errors result in an ERROR_OUT of CORRECTABLE_ERROR_OUT, based on the severity and mask.

Error Case	AXI Interface Response	Error Status Registers and Bits	Client Action	Local Interrupt?
Header Log Overflow	No effect.	Header Log Overflow Status bit in AER Correctable Error Status Register.	Client needs to read the header log and clear the AER error status registers.	No
Correctable ECC error	No effect.	Corrected Internal Error Status in AER Correctable Error Status Register.	None required.	No
Replay Timeout	No effect.	Replay Timeout bit in AER Correctable Error Status Register and Local Error Status Register.	None required.	Yes
Replay Num Rollover	No effect.	Replay Num Rollover bit in AER Correctable Error Status Register and Local Error Status Register.	None required.	Yes
PHY Layer Errors	No effect.	PHY Error Detected bit in AER Correctable Error Status Register and Local Error Status Register.	None required.	Yes
TLP LCRC Errors	No effect.	Bad TLP Status bit in AER Correctable Error Status Register.	None required.	No
DLLP LCRC Error	No effect.	Bad DLLP Status bit in AER Correctable Error Status Register.	None required.	No

Table 81: Error Handling: Other Errors

These errors result in an ERROR_OUT of NIL.

Error Case	AXI Interface Response	Error Status Registers and Bits	Client Action	Local Interrupt?
Client sends SLVERR with TARGET_ AXI_RRESP	SLVERR sent from TARGET_AXI_RRESP	Signaled Target Abort bit in Command and Status Register.	Client sends abort by sending SLVERR on TARGET_ AXI_RRESP.	No
Outbound MemWr TLP Poisoned	The AXI interface returns OK (2'd0) response. TLP is be transmitted on the link with the endpoint bit set.	Master Data Parity Error bit in Command and Status Register.	None required.	No
Function-Level Reset from Host	The PCIe Controller asserts FLR_IN_PROGRESS for the affected function's VF_FLR_IN_PROGRESS for the VFs.	Function-Level Reset bit in PCI Express Device Control and Status Register.	Client must assert FLR_DONE.	No
Link Down Reset	Asserts the LINK_DOWN_RESET_ OUT signal for eight clock cycles.	Link Down Indication bit in AXI Configuration Registers.	Client must reset the link down indication bit after clearing all outstanding requests.	No

Multiple Errors

When multiple errors are detected in a single TLP, the PCIe specification recommends that a single error be reported. The precedence of errors the PCIe Controller reports is (from highest to lowest):

1. Uncorrectable Internal Error
2. Receiver Overflow
3. Flow Control Protocol Error
4. Malformed TLP
5. ECRC Check Failed
6. Unsupported Request (UR), Completer Abort (CA), or Unexpected Completion
7. Poisoned TLP Received

Multiple-Error Scenarios

A multiple-error scenarios has a mix of uncorrectable and advisory non-fatal errors in a single TLP. The error precedence determines which error is reported when the PCIe Controller detects multiple errors. However, if the higher precedence error is advisory and the lower precedence error is not, the reported error should not actually be advisory.

Per PCIe specification section 2.7.2.2, the PCIe Controller these error combinations:

- *AtomicOp request that is poisoned as well as unsupported*—The PCIe Controller reports it as an unsupported request received error because of the higher precedence. However, because it is also poisoned, the PCIe Controller reports it as an uncorrectable error and not as advisory non-fatal.CfgWr
- *For a request that is poisoned as well as unsupported*—The PCIe Controller reports it as an unsupported request received error because of the higher precedence. However, because it is also poisoned, it is reported as an uncorrectable error and not as advisory non-fatal.
- *For a MWr, IOWr, or MsgD request that is poisoned as well as unsupported*—The PCIe Controller reports it as an unsupported request received error because of the higher precedence. If the Poisoned TLP Received Advisory Non Fatal bit is set to 0 in the Debug Mux Control 2 Register, it is reported as an uncorrectable error and not as advisory non-fatal.

Appendix C: LTSSM State Encoding

The following table provides the LTSSM state encoding for the PCIe Controller's `LTSSM_STATE` output signal as well the state read from the Physical Layer Configuration Register 0.

Table 82: LTSSM State Encoding

LTSSM State Name	Register Value (Hex)
Detect.Quiet	00
Detect.Active	01
Polling.Active	02
Polling.Compliance	03
Polling.Configuration	04
Configuration.Linkwidth.Start	05
Configuration.Linkwidth.Accept	06
Configuration.Lanenum.Accept	07
Configuration.Lanenum.Wait	08
Configuration.Complete	09
Configuration.Idle	0A
Recovery.RcvrLock	0B
Recovery.Speed	0C
Recovery.RcvrCfg	0D
Recovery.Idle	0E
L0	10
Rx_L0s.Entry	11
Rx_L0s.Idle	12
Rx_L0s.FTS	13
Tx_L0s.Entry	14
Tx_L0s.Idle	15
Tx_L0s.FTS	16
L1.Entry	17
L1.Idle	18
L2.Idle	19
L2.TransmitWake	1A
Disabled	20
Loopback.Entry (Master)	21
Loopback.Active (Master)	22
Loopback.Exit (Master)	23
Loopback.Entry (Slave)	24
Loopback.Active (Slave)	25
Loopback.Exit (Slave)	26
Hot Reset	27
Recovery.Equalization, Phase 0	28
Recovery.Equalization, Phase 1	29
Recovery.Equalization, Phase 2	2A

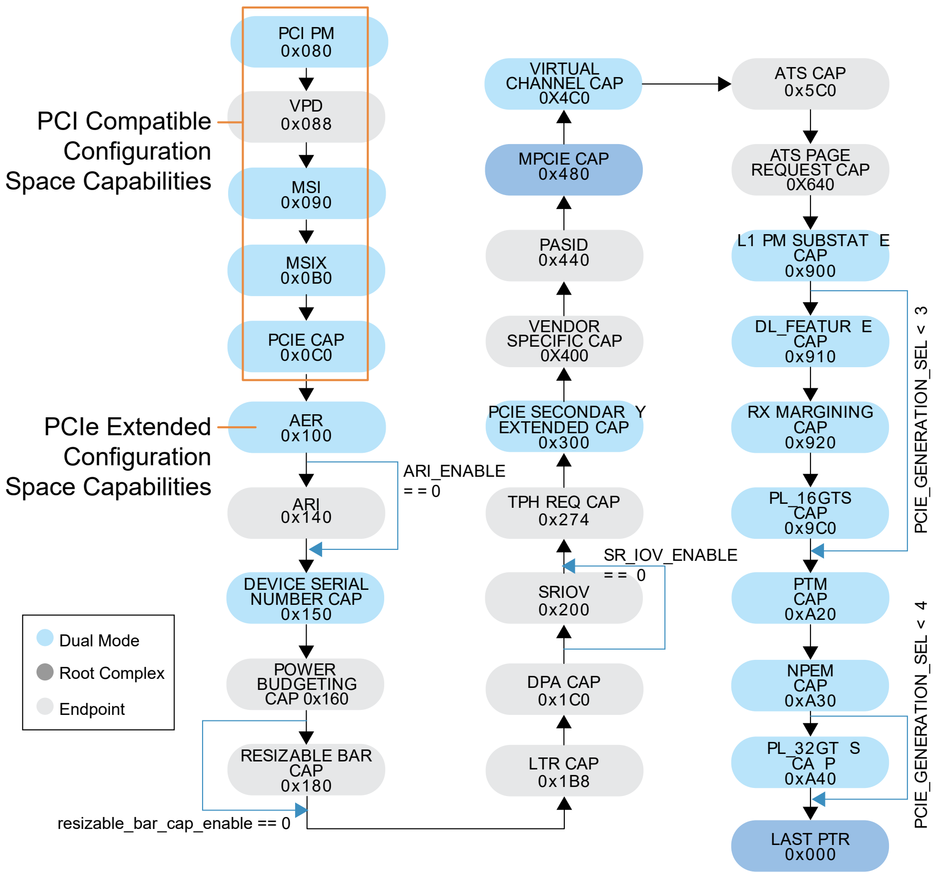
LTSSM State Name	Register Value (Hex)
Recovery.Equalization, Phase 3	2B

Appendix D: PCIe Configuration Capabilities Linked List

The following figure shows the PCIe Controller Capabilities Linked List implementation. Note that:

- Each bubble shows the address offset for each capability structure.
- The address offsets are all 12 bits.

Figure 45: Configuration Capabilities



Some capability structures may not be selected in this configuration. In that case, the next capability pointer of the previous selected capability structure points to the next selected capability structure. For example, if DPA, SR-IOV, or TPH_REQ capabilities are not selected, the `LTR_NEXT_CAPABILITY_POINTER` points to the PCIe Secondary Extended Capability Structure.

Additionally, some capability structures are only visible to the host configuration software if the corresponding strap input is enabled.

- ARI capability is visible only if the `ARI_ENABLE` strap input is 1.
- SR-IOV Capability is visible only if the `SR_IOV_ENABLE` strap input is 1.

If these strap inputs are not enabled, the capability linked list is automatically modified to link the previous capability with the next capability. These strap inputs must be stable before deasserting `RESET_N`, `MGMT_RESET_N`, or `MGMT_STICKY_RESET_N`.

Configuration-Specific Capabilities

The following tables show the PF and VF PCIe capabilities.

Table 83: Endpoint PF0 PCIe Capabilities List

PCIe Capability	Offset (Hex)	Notes
PCI PM Capability	0x80	–
MSI Capability	0x90	–
MSI-X Capability	0x0B0	–
PCI Express Capability	0x0C0	–
AER Capability	0x100	–
ARI Capability	0x140	–
Device Serial Number Capability	0x150	–
Power Budgeting Capability	0x160	–
Resizable BAR Capability	0x180	LM Register, Resizable BAR Capability Enable bit: 1: Capability Present 0: Capability Bypassed
LTR Capability	0x1B8	–
DPA Capability	0x1C0	–
SR-IOV Capability	0x200	–
TPH Requester Capability	0x274	–
Secondary PCI Express Extended Capability	0x300	–
Vendor-Specific Capability	0x400	–
PASID Capability	0x440	–
ATS Capability	0x5C0	–
ATS PR Extended Capability	0x640	–
L1 PM Substates Extended Capability	0x900	–
Data Link Feature Extended Capability	0x910	–
Lane Margining Extended Capability	0x920	–
Physical Layer 16.0G Extended Capability	0x9C0	–

Table 84: Endpoint PCIe Capabilities List for PFn (n>0)

PCIe Capability	Offset (Hex)	Notes
PCI PM Capability	0x80	–
MSI Capability	0x090	–
MSIX Capability	0x0B0	–
PCI Express Capability	0x0C0	–
AER Capability	0x100	–
ARI Capability	0x140	–
Device Serial Number Capability	0x150	–
Power Budgeting Capability	0x160	–
Resizable BAR Capability	0x180	LM Register, Resizable BAR Capability Enable bit: 1: Capability Present 0: Capability Bypassed
DPA Capability	0x1C0	–
SR-IOV Capability	0x200	–
TPH Requester Capability	0x274	–
Vendor-Specific Capability	0x400	–
PASID Capability	0x440	–
ATS Capability	0x5C0	–

PCIe Capability	Offset (Hex)	Notes
ATS PR Extended Capability	0x640	–
Data Link Feature Extended Capability	0x910	–

Table 85: VF0 PCIe Capabilities List

PCIe Capability	Offset(hex)
PCI PM Capability	0x80
MSI Capability	0x090
MSIX Capability	0x0B0
PCI Express Capability	0x0C0
AER Capability	0x100
ARI Capability	0x140
TPH Requester Capability	0x274
ATS Capability	0x5C0

Table 86: PCIe Capabilities List for VFn (n>0)

PCIe Capability	Offset(hex)
PCI PM Capability	0x80
MSI Capability	0x090
MSIX Capability	0x0B0
PCI Express Capability	0x0C0
AER Capability	0x100
ARI Capability	0x140
TPH Requester Capability	0x274
ATS Capability	0x5C0

Appendix E: Supported Chipsets

The functionality of the example design has been tested and verified on the following chipsets. The negotiated speed and width of the PCIe link is displayed in the table below.

Table 87: Supported Chipsets

Chipset	CPU	PCIe Slot	Negotiated Speed & Width
Intel B360	Intel Core i5-8400	PCI Express 3.0/2.0 x16 slot	Gen3x4
AMD B550M	AMD Ryzen 5 5600X	PCIE1 slot	Gen4x4
AMD B650M	AMD Ryzen 7 7700	PCIEX16 slot	Gen4x4
Intel H610I	Intel Core i3-12100	PCIEX16 slot	Gen4x4
AMD X570	AMD Ryzen 9 5900X	PCIE3 slot	Gen4x4
Intel B760M	Intel Core i3-12100	PCIE1 slot	Gen4x4

Revision History

Table 88: Document Revision History

Date	Version	Description
June 2025	1.6	Updated AXI read: **TLP (2n and 2n>1, up to 32 Bytes, Unaligned Address)** on page 26. (DOC-2528) Updated AXI write: **AXI Master Write Operation** on page 28. (DOC-2528) Added **End-to-End Data Protection** on page 31. (DOC-2541). Updated **Clock Sources** on page 61. (DOC-2572) Updated **Function-Level Reset Signals** on page 105. (DOC-2541) Updated note in **Figure 25: FLR Handshake** on page 65. (DOC-2541)
April 2025	1.5	Fixed typos in **AXI Master Read Operation** on page 25. (DOC-2515)
April 2025	1.4	Added **Appendix E: Supported Chipsets** on page 122. (DOC-2207) Update **Table 29: AXI Slave Sideband Signal Description (MASTER_AXI_AWUSER and MASTER_AXI_ARUSER)** on page 50. (DOC-2493)
February 2025	1.3	Added note about incompatible clock tolerances for SRIS. (DOC-2265) Added information on clock sources from PLLs. (DOC-2265) Clarified supported SPI active configuration details. (DOC-2265)
January 2025	1.2	Updated **Interrupt Sideband Signals** on page 59.
September 2024	1.1	Corrected link up diagram. (DOC-2043)
July 2024	1.0	Initial release.