OpenPOWER I/O Design Architecture

Version 2

Workgroup Specification

Revision 1.0.0 (February 17, 2016)

www.openpowerfoundation.org

OpenPOWER I/O Design Architecture: Version 2

Revision 1.0.0 (2016-02-17)

This document is the workproduct of the OpenPOWER Foundation Hardware Architecture Workgroup.
Acknowledgement to members of the workgroup for their contributions

Table of Contents

List of Figures

List of Tables

Preface

1. Conventions

The OpenPOWER Foundation documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Command prompts

$ prompt Any user, including the `root` user, can run commands that are prefixed with the $ prompt.

 # prompt The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

2. Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
January 13, 2016	• Revision 1.0.0 - Workgroup Specification • Clean-up markings and rev #. Clean-up legal wording and added foundation info appendix
October 1, 2015	• Revision 09. - Public Review Draft • Clean-up of typo/conversion errors.
August 21, 2015	• Revision 09. - Public Review Draft
April 23, 2015	• Creation based on IBM IODA2 Specification - revision 1.0 • Updates from Hardware Architecture WG review of original submission

1. About This Document

1.1. Purpose

The purpose of the I/O Design Architecture, version 2 (IODA2) specification is to describe the chip architecture for key aspects of PCIe® based host bridge (PHB) designs for IBM® POWER8™ systems.

1.2. Numbering Conventions

Big-endian numbering of bytes and bits is used in this document unless otherwise indicated. In big-endian systems, numbering of bits starts at 0 for the most significant bit and continues to the least significant bit. Little-endian numbering might be implied by the bit-ordering sequence in figures or text where the low-numbered bits are on the right. For example, [31:0] is little-endian ordering and [0:31] is big-endian ordering.

1.3. Reference Documentation

For additional information, see *IBM Power Architecture® Platform Requirements (PAPR)*.

1.4. OpenPOWER Foundation Standards Track Work Product

This document is an OpenPOWER Foundation Standards Track Work Product, and is intended to progress through these steps:

1. Work Group Specification Draft
2. Work Group Specification Public Review Draft
3. **Work Group Specification**
4. Candidate OpenPOWER Standard
5. OpenPOWER Standard
6. Approved Errata

2. Introduction

The purpose of the I/O Design Architecture, version 2 (IODA2) specification is to describe the chip architecture for key aspects of PCIe®-based host bridge (PHB) designs for IBM® POWER8™ systems.

The following terminology is used in this document:

- The term “real” used in relationship to addresses means “processor real address.”
- The term “PCI” is used to describe the most recent versions of all forms of PCI® standards. Where there are significant differences between individual PCI standards, the following terminology is used to differentiate between the PCI® standards: conventional PCI®, PCI-X®, and PCI-Express®. For example, POWER8 implements PCIe Gen 3.
- The term “MSI” is used to refer to “MSI” and “MSI-X”, generically. Where there are differences, the distinction is made in context.
- The term “implementation dependent” is used to refer to specifics beyond the scope of this architecture, which should be provided in the implementation’s specification.

For the definitions of more terms and acronyms used in this document, see the Appendix C, Glossary [60].

2.1. Conformance to this Specification

Any implementation of this specification must adhere to the following set of numbered conformance clauses to claim conformance to this specification (or any optional portion of it):

1. Hardware Requirement: The PCI host bridge (PHB) hardware must implement all the requirements specified as “Hardware Requirements” in this architecture, unless otherwise required by the specific requirement.
2. Firmware Requirement: The platform firmware must implement all the requirements specified as “Firmware Requirements” in this architecture, unless otherwise required by the specific requirement.

2.2. General Information

This section provides some general background on translation control entries (TCEs), message signalled interrupts (MSIs), enhanced I/O error handling (EEH), and direct memory access (DMA) ordering rules.

2.2.1. I/O Load/Store Address Space

Load and *Store* instructions that are issued to addresses that target I/O adapter (IOA) memory or I/O address ranges [1] are called memory mapped I/O (MMIO).

[1] There are three PCI address ranges: configuration, I/O, and memory. The I/O space is primarily for legacy material; its use is discouraged by PCI-X and later versions of the architecture in favor of the memory address space. This document primarily addresses the memory address space, as used by both MMIO and DMA. Some reference are made to the I/O space as it relates to MMIO.

2.2.2. TCEs

Translation control entries (TCEs) are to I/O what page table entries (PTEs) are to the processor. That is, they translate from one address space to another. In particular, TCEs translate from an I/O bus memory address to a physical system memory address. The TCEs perform the following functions:

- Expand the I/O address space addressing for IOAs that cannot access all of the system memory address space. For example, a 32-bit IOA must have its address expanded for systems with more than 4 GB of system memory. Otherwise, such an IOA has to DMA its data to and from a buffer in the lower address range. The processor has to move the data from and to the real target page in memory.
- Provide indirection in addressing:
  - —For logical partitioning (LPAR), it is necessary to hide the real address of the memory from the partitions.
  - —For dynamic logical partitioning (DLPAR) and memory migration, and for virtual partition memory, it is necessary to be able to move the physical memory transparently under the IOA from one location to another.
  - —For virtual I/O, the address of the memory in the client partition must be hidden from the server partition.
  - —Assist some IOAs by providing hardware scatter-gather. This provides the IOA with a contiguous address space instead of one that is broken at every 4K page boundary. This can actually improve the performance of some IOAs if the platform can perform the TCE manipulation faster than the IOA can process scatter/gather lists.
- Provide extra protection from IOA hardware, microcode, and device driver bugs by providing read-only and write-only (as well as read-write) protection through two control bits in the TCE.

2.2.3. MSIs

The PCI architecture allows signalling of interrupts in either of two ways:

- Through a signal pin. [2] This is called a level-signalled interrupt, or LSI.
- Through a message. This is called a message-signalled interrupt, or MSI. [3]

MSIs have the advantage of pushing an IOA's DMA data that it has previously written ahead of it. Therefore, when the interrupt is presented, the device driver (DD) knows that the data is in the processor's coherency domain. That is, it is immediately available.

There is no such guarantee with an LSI. Therefore, when the DD sees an LSI, it must perform a *Load* instruction targeted to its IOA. Then, it must wait for the *Load* data to return before being assured that the previously DMAed data is available to be used. This *Load* is a performance penalty. In addition, PCIe allows only four LSIs per PHB, which severely restricts usability.

[2] For PCIe, the LSI interrupts are not signalled by a physical pin (sometimes called out-of-band signalling), but rather through a logical pin that is shipped across the PCIe fabric as a packet (sometimes called in-band signalling).
[3] The "message" for MSI is really a DMA write operation to a special address with special data, as far as the IOA and the I/O fabric are concerned.

LSIs are defined by this architecture, but MSI and MSI-X are the focus. PCI defines:

- For base MSI: Up to 32 interrupts per function of the IOA. The IOA can have up to eight functions, giving up to 256 interrupts per IOA.
- For MSI-X: Up to 2K interrupts per function of the IOA. The IOA can have up to eight functions, giving up to 16K interrupts per IOA.

However, this flexibility comes at a cost in the scalability of the interrupt controller structure. This architecture addresses the scalability issue by placing the interrupt vector structures and interrupt state in system memory. It has firmware assist with the interrupt state machine during end of interrupt (EOI) processing and with the processing of missed interrupts while an MSI is disabled.

2.2.4. EEH

Enhanced error handling (EEH) is a powerful technology developed by IBM to prevent I/O errors from propagating to the system and causing unrecoverable errors, which generally bring down the operating system. EEH is a required technology for logically partitioned systems, so that an error in the I/O subsystem of one partition does not affect the other LPAR partitions.

EEH stops operations to and from an IOA when an error is detected with that IOA. This stopped condition is called the Stopped state [4]. The Stopped state has the following key requirements:

- The IOA function must be prevented from completing the I/O operation in error so that the requester of the I/O operation does not use bad data.
- The Stopped state must appear to a DD to be isolated to just that DD. This implies extra hardware or firmware to support the continuation of the I/O operation of other IOA functions when an error is generated from another IOA function.

Exceptions:

In the following cases, the DDs for these functions must coordinate any Stopped state recovery:

- For a plug-in adapter where the EEH functionality is implemented above the physical plug-in connector and where the plug-in adapter has multiple IOA functions on it under a PCI-to-PCI bridge
- For an IOA that has multiple functions on it, and for which there exist multiple DDs (potentially one per function)

- Software (DD or above) must not be able to introduce an error that can cause a Stopped state of other IOA functions. That is, it must not introduce a stopped-state error to IOA functions other than the ones controlled by the DD.

  —Software might, for example, improperly set up the TCEs for an I/O operation or pass the wrong address to its IOA. This can cause an access to a TCE that is invalid. (The TCE is not set up, or the TCE is set to read-only for a write or to a PCI atomic operation or write-only for a read or PCI atomic operation.) This causes a Stopped state.

  —It is acceptable for a platform hardware error, but not a DD or IOA function hardware error, to affect multiple IOA functions. However, the recovery from such an error must be transparent to the DD. That is, the platform makes it appear to all IOA functions that they have encountered the error condition themselves.

[4] Sometimes this state is also referred to as the “freeze” state or condition. In addition, the IOA Stopped state can be broken down into the MMIO Stopped state and the DMA Stopped state. In this document, if “MMIO” or “DMA” is not specified along with “Stopped state”, the reference is either to the general concept or to both the MMIO and DMA Stopped states.

- The DD must be able to detect the Stopped state condition.
- The DD (and, therefore, the platform) must be able to remove its IOA function from the MMIO Stopped state for MMIO operations, independent of other IOA functions.

  —The capturing of fault information for problem determination must be allowed after the Stopped state condition occurs.
- The DD (and, therefore, the platform) must be able to remove the IOA function from the DMA Stopped state for DMA operations independent of other IOA functions. The DD is responsible for bringing its IOA function to a known good state before removing it from the DMA Stopped state, to avoid the possibility of improper operations from its IOA function. In many cases, the DD needs to bring its IOA function back to the reset state or as close to the reset state as possible, and then restart any incomplete operations.
- The platform must not pass along MSI interrupts from the IOA function while the IOA function is in the DMA Stopped state.

3. Design Specifics

This chapter describes applicable design specifics as they apply to IBM POWER8™ systems. The designs in this chapter are not the only designs that meet the Power Architecture Platform Requirements (PAPR). However, to enable design sharing and to prevent firmware impacts from one implementation to the next, the designs in this chapter are more or less “fixed” unless negotiation between the hardware and firmware designers provide changes to this direction. (That is, the changes become a chip I/O architecture.) Anyone making changes to these design points must ensure that the changes allow the designs to continue to meet the architectures specified in the PAPR.

This chapter does not provide the detailed definitions (bits, bytes, and addresses) of the registers needed to implement these designs. However, the stability of those is no less important. Designers of chips that generate the same buses are expected to use the same register definitions whenever possible to reduce the impact to firmware implementations.

Each implementation is expected to devise a consistent way to self-identify its capabilities. For example, a register or set of registers, or some sort of informational header in the chip’s register space, can be used. This document does not propose a way to do this.

3.1. High-Level Specifics

Endpoint partitioning is the concept of being able to identify operations to or from an individual partitionable endpoint (PE) across an I/O fabric. For more information, see Appendix A, Endpoint Partitioning [49].

R1-3.1-1 Hardware Requirement:

The PCI host bridge (PHB) hardware must implement all the requirements specified as “Hardware Requirements” in this architecture, unless otherwise required by the specific requirement.

R1-3.1-2 Firmware Requirement:

The platform firmware must implement all the requirements specified as “Firmware Requirements” in this architecture, unless otherwise required by the specific requirement.

3.2. Lower-Level Details

3.2.1. PE# Determination, PE State, EEH, and Error Injection

The PHB hardware determines, for any given operation, the PE numbers (PE#s) to which the operation belongs. It tracks the state of that PE# so that it can stop the PE on an error and prevent further operations after the error. It does this on a per PE# basis so that nonaffected PEs can continue to operate while the affected PE is recovered. For more information, see Section 3.2.1.3, “PE State and EEH” [12].

The PE# determination is made during the following operations:

- MMIO operations: The address is decoded and a range or multiple ranges of addresses are assigned to each PE#. This is done in an implementation-dependent way. For requirements, see Section 3.2.1.1, “MMIO PE# Determination” [7].
- DMA or MSI operations or error message from the PCIe link: The requester ID (RID) associated with the operation is used as an index into an RID translation table (RTT). For requirements, see Section 3.2.1.2, “DMA and Error Message PE# Determination, RTT, RTC Invalidate, and PELT-V” [7].
  - —If the operation is a DMA or MSI operation, the PE# field of the RID translation entry (RTE) indicates the PE# associated with the RID.
  - —If the operation is an error message, the PE# field is the index of the PE lookup table (vector) (PELT-V). When the PELT-V is accessed, the entry indicates, by a vector of bits, which PE#s are affected by the RID. PELT-V entries are generated by firmware for all PE#s. That is, the depth of the PELT-V is equal to the number of PEs implemented. Hierarchical RIDs, such as switch RIDs and IOV PFs, have more than one bit set in the PE look-up entry (vector) (PELE-V). Single RIDs, such as those for VFs, have only one bit set.

3.2.1.1. MMIO PE# Determination

PHBs are required to support MMIO address-space decoding and the assignment of PE#s to those decodes, as specified by requirements in this section. How this is implemented is implementation dependent.

R1-3.2.1.1-1 Hardware Requirement:

The PHB hardware must support the decoding of MMIO addresses, and both of the following conditions must be met:

a. Enough address-space decodes must be provided to support the necessary, probably noncontiguous, BAR spaces of the devices to be located below the PHB, including legacy devices that require an address programmed in their BARs that are below 4 GB.

b. The address decodes must be assigned an appropriate PE#. When multiple decodes are provided for any given function, the PE# assigned must be the same.

Hardware Implementation Note: Relative to this requirement, how the hardware chooses to implement this is outside of the scope of this architecture. However, the hardware implementation must take special care relative to the configurations to be supported under the PHB, especially in terms of the implications of switches, IOV endpoints, and hot plug. Consideration must also be given to the fact that devices can implement multiple sets of BARs, and implementations generally need to allow for three, potentially noncontiguous, BARs per function.

R1-3.2.1.1-2 Firmware Requirement

The platform firmware must set up any chip implementation-specific address ranges appropriately.

3.2.1.2. DMA and Error Message PE# Determination, RTT, RTC Invalidate, and PELT-V

The RTT and PELT-V tables are implemented in system memory. The most recently used RTEs are cached in the PHB hardware for DMA performance and, optionally, for MSI performance. RTEs can be cached for processing of PCI error messages.

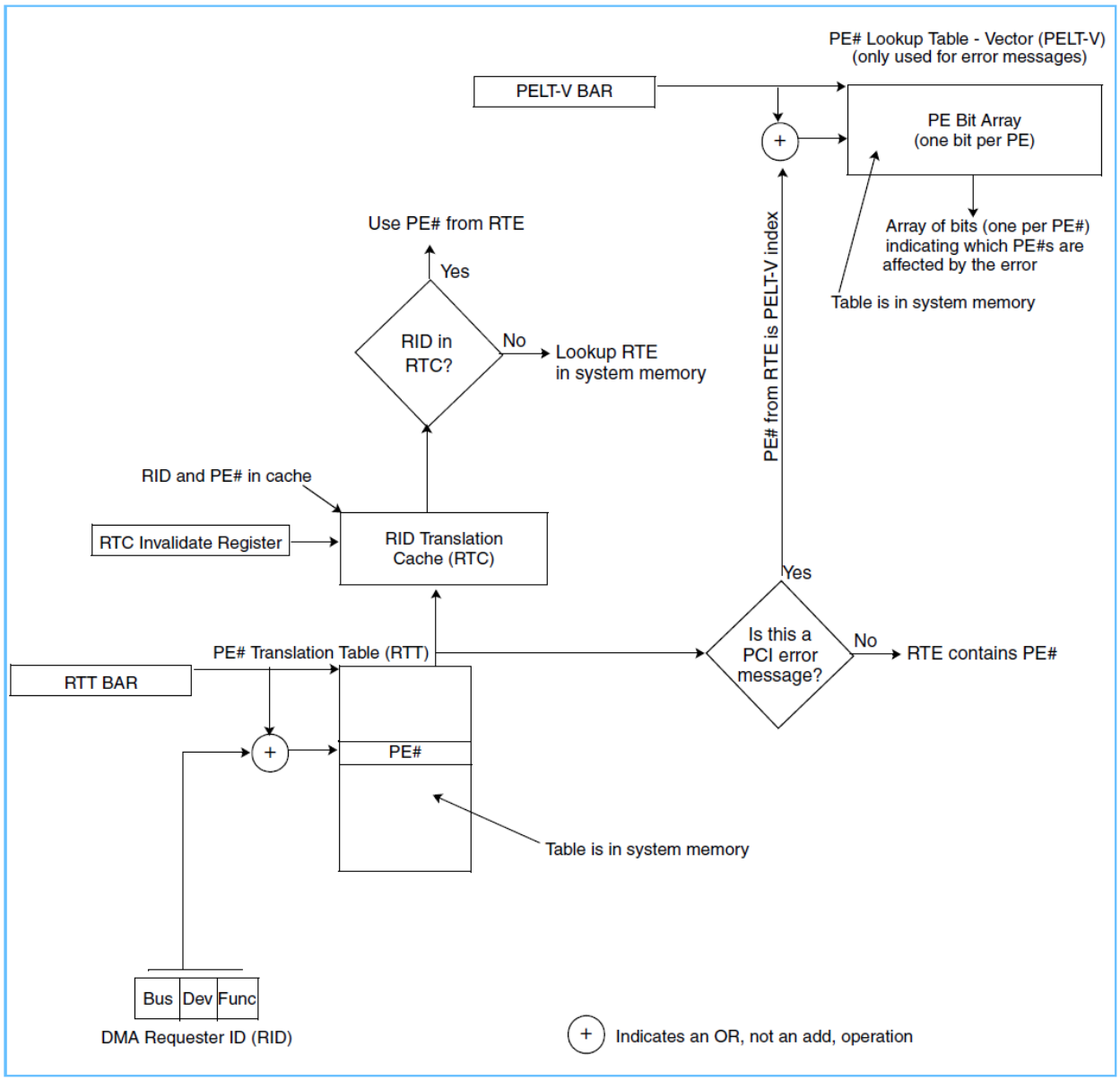
DMA operations or error messages that come from the PCIe link contain a requester ID (RID) associated with the operation. The RID is used as an index into an RID translation table (RTT). The RTT entry (RTE) contains a PE# field.

If the operation is a DMA or MSI operation, the RTE indicates the PE# associated with the RID. If the operation is an error message, the PE# field is the index of the PELT-V. When the PELT-V is accessed, the entry indicates, by a vector of bits, which PE#s are affected by the RID. PELT-V entries are generated by firmware for all PE#s. That is, the depth of the PELT-V is equal to the number of PEs implemented. Hierarchical RIDs, such as switch RIDs and IOV PFs, have more than one bit in the PELE-V set. Single RIDs, such as VFs, have only one bit set. For DMA operations and, optionally, for MSI and error message operations, the RID and PE# are stored in a cache on the PHB chip. The cache is referenced first in the PE translation process. If the entry is not in the cache, a reference is made to the RTT in system memory.

Figure 3.1, “PE# Determination for DMA and Error Messages” [9]shows how PE#s are determined for DMA and error messages. For specific hardware and firmware requirements related to this, see R1-3.2.1.2 [7]-1 and R1-3.2.1.2 [7]-2.

The RTT is 64K-entries deep because the RID, which is the index into the table, is 16 bits in length. The width and depth of the PELT-V table is determined by the number of PEs implemented, with one entry per PE and one bit width for each PE, and with the width in bytes being a power of 2.

Figure 3.1. PE# Determination for DMA and Error Messages



The definitions of the RTT, RTC Invalidate Register, and PELT-V tables are shown in Table 3.1, “RTE Definition” [10], Table 3.2, “RTC Invalidate Register Definition” [10], and Table 3.3, “PELE-V Definition” [10].

In the tables, [ ] designates optional bits or bytes. Optional bits and bytes that are not implemented by the hardware must be ignored by the hardware. Implementations that do not implement the full size of the field must treat unused bits and bytes the same as optional bits and bytes. Reserved bits and bytes must be set as zeros by firmware and must be returned as written on a Load (these tables are in system memory).

Table 3.1. RTE Definition

Bytes	Bits	Field	Definition
0:1	All	PE#	The PE# or index into the PELT-V. A PE# of all ones is invalid. Therefore, firmware must set the PE# to all ones for RIDs that are not configured.

Table 3.2. RTC Invalidate Register Definition

Bits	Field	Definition
0	Invalidate All	0: Invalidate the entry in the RTC specified by the Requester ID field. 1: Invalidate all entries in the RTC regardless of the value in the Requester ID field.
1:15	Reserved	Reserved
16:31	Requester ID (0:15)	The 16-bit Requester ID field of the RTC entry to invalidate when the Invalidate All bit is set to a zero.
32:63	Reserved	Reserved

Table 3.3. PELE-V Definition

Bytes	Bits	Field	Definition
0:[n]	[All]	PE Array	An array of bits, one bit per PE, that indicates which PEs are affected. An implementation only needs to support the number of bits necessary to support the number of PEs that it supports. Unused bits are at the highest bit numbers (bit 0 of byte 0 corresponds to the first PE#, bit 1 to the second, and so on). The number of bytes implemented is a power of 2.

R1-3.2.1.2-1 Hardware Requirement

The PHB hardware must take all of the following actions:

a. Implement the RTT in system memory, with the entries defined by Table 3.1, “RTE Definition” [10], and provide a register that firmware can set to point to the starting address of that table.

b. Implement a BAR to point to the start of the RTT (RTT BAR), loadable by the firmware.

c. Implement the PELT-V in system memory, with entries defined by Table 3.3, “PELE-V Definition” [10], and provide a register that firmware can set to point to the starting address that table.

d. Implement a BAR to point to the start of the PELT-V (PELT-V BAR), loadable by the firmware.

e. Provide an RID translation cache (RTC) for caching RTEs used for DMA operations, optionally for MSI operations and for error message operations.

Hardware Implementation Notes:

1. Appropriate sizing of the RTC is necessary to have a high probability of a cache hit during DMA. Currently, no performance analysis has been done, and no rule-of-thumb can be provided. However, the target size for the first implementation of this architecture is a number of entries in the RTC equal to one-fourth of the number of PEs.

2. In part e of this requirement, it is best for the optional caching for MSI operations to be controllable on a per-PHB basis by a configuration bit setting.

f. Provide an RTC Invalidate Register, as defined by Table 3.2, “RTC Invalidate Register Definition” [10], for invalidating individual cached RTEs or all RTEs. The hardware must stop using

the entry when firmware indicates the invalidate, but can wait until the RTC entry is used once by a DMA operation. A *Store* to this register must perform the specified invalidation operation. A *Load* from this register must return the last value *Stored* to this register.

Hardware Implementation Notes:

Relative to requirement part f, the hardware is required to provide firmware a way to clean up cache entries when they are no longer needed or when the RID to PE# might have changed. This is done during firmware clean-up operations, for example on hot plug or partition shutdown. The firmware might also need to be able to invalidate all entries in the RTC, for example if the RTC BAR is to be changed. The RTC Invalidate register format is not defined by this architecture.

g. During RID translation, if the RID is in the RTC, use the cached value.

h. During RID translation, if the RID is not in the RTC, use the RID as an index into the RTT for the PHB, and read the RTE. Cache the entry if this is a DMA operation, optionally cache it for MSI operations, and optionally cache it for error messages. For error messages, if the PE# is not all ones, use the PE# field in the RTE as a PELT-V index. Access the PELT-V entry and use the bit array obtained as the array of PE#s that are affected by the error.

Architecture Notes

DMAs from entities like PCIe IOV PFs, and MSIs from PFs, and switch RIDs have only one PE# associated with them. Error messages from PFs and switch RIDs are likely to point to multiple PE#s in the PELT-V.

i. When accessing the RTT, if the RTE is all ones, an invalid RID has been received. The hardware must set the appropriate error bit in the PHB, store the RID information, and interrupt the firmware for processing of the error.

R1-3.2.1.2-2 Firmware Requirement:

The platform firmware must take all of the following actions:

a. Set up the RTT BAR and PELT-V BAR to point to the start of those structures in contiguous real system memory, with a size that is a power of 2 and with an address alignment on an integer multiple of the size of the table.

b. To change the RTT BAR or PELT-V BAR while DMA operations might be in progress, the firmware must first create the new table and change the BAR. Then, firmware must make sure that all DMA operations that are queued in the PHB (DMA write/read and MSIs) are completed before reusing the system memory locations that were previously used by the table.

c. For RIDs that do DMA or which issue MSIs, set up the PE# in the RTT to the PE# that is associated with the RID. There might be more than one RID associated with the same PE#.

d. For each PE#, create an entry in the PELT-V with an offset equal to the PE#. That entry must contain the appropriate bits set for each PE that might be affected by an error against the RID associated with the PE#.

e. For invalid RIDs (that is, ones that are not configured in the PCIe hierarchy), set the RTE to all ones.

f. Manage the RTT and PELT-V entries in system memory and the RTC on the PHB chip, appropriately, at all times, including during hot plug and DLPAR operations.

3.2.1.3. PE State and EEH

The PE state includes, but is not limited to, the following items:

- The EEH Enablement state: Indicates whether EEH is enabled for the PE or not.
- The MMIO Stopped state: Indicates whether MMIO operations are frozen for the PE or not. If MMIO is stopped for the PE, the PE is said to have its MMIO Stopped state set or to be in the MMIO Stopped state.
- The DMA Stopped state: Indicates whether DMA (and MSI) operations are frozen for the PE or not. If DMA is stopped for the PE, the PE is said to have its DMA Stopped state set or to be in the DMA Stopped state.

Note: For EEH-enabled DDs, on the detection that their PE is in the Stopped state (all ones on a *Load* when not expected followed by a query call to firmware), the normal progression is as follows:

1. Remove their PE from the MMIO Stopped state (that is, reset that state).
2. Issue a series of *Load*/*Stores* to determine the problem.
3. Clear it either by a hardware reset to the PE or by separately removing the IOA function from the DMA Stopped state. The latter approach might not be possible for some IOA functions or under certain circumstances.

R1-3.2.1.3-1 Hardware Requirement:

Each PE's MMIO Stopped state and DMA Stopped state must be independent of each other. The hardware must give the firmware a way to set and clear the DMA Stopped state and the MMIO Stopped state:

- Independently from each other
- Independent for those Stopped states for other PEs
- Atomically with any other errors that might be occurring at the time

R1-3.2.1.3-2 Hardware Requirement:

The PHB hardware must take all of the following actions:

a. For any detected failure to/from a PE, set both the MMIO Stopped and DMA Stopped states for the PE.

Exception: Not required if the error that caused the failure can be reported to the IOA function in a way that enables it to report the error to its device driver while avoiding any data corruption.

b. If an I/O fabric consists of a hierarchy of components, when a failure is detected in the fabric and that failure cannot be isolated to a single PE, put all PEs that are downstream of the failure into the MMIO Stopped and DMA Stopped states if they might be affected by the failure.

c. From the time that the MMIO Stopped state is entered for a PE, prevent the PE from responding to *Load* and *Store* operations including the operation that caused the PE to enter the MMIO Stopped state. A *Load* operation must return all ones with no error indication and a *Store* opera-

tion must be discarded until the firmware directs the hardware otherwise or until the PHB chip is reset. That is, *Load* and *Store* operations are treated as if they received a conventional PCI master abort error,

d. From the time that the DMA Stopped state is entered for a PE, prevent the PE from initiating a new DMA request or completing a DMA request that caused the PE to enter the DMA Stopped state, including MSI or MSI-X DMA operations, until the firmware directs the hardware otherwise or until the PHB chip is reset. DMA requests that were started before the DMA Stopped state is entered can be completed. DMA requests requiring a response that are discarded due to the PE being in the DMA Stopped state (for example, a read request or an atomic request), return a UR to the requester. DMA read response data that is returned to the PHB after the setting of the DMA Stopped state must be returned to the requester with a Completer Abort status, if possible. Otherwise, discard the response; for example, a link down condition is a case where return of a response is not possible.

e. Provide the capability to the firmware to determine, on a per-PE basis, that a failure occurred which caused the PE to be put into the MMIO Stopped and DMA Stopped states and to read the actual state information (MMIO Stopped state and DMA Stopped state).

f. Provide the capability of separately enabling and resetting the DMA Stopped and MMIO Stopped states for a PE without disturbing other PEs on the platform. The hardware must provide this capability without requiring a PE reset and must do so through normal processor *Store*instructions. Firmware enabling of MMIO or DMA Stopped states must have the same, and immediate, effect as if a PHB-detected error set those states.

g. Provide the capability to the firmware to deactivate all provided resets (hot reset, fundamental reset), independent of other resets. The hardware must provide the proper controls on the reset transitions to prevent failures from being introduced into the platform by the changing of the reset.

h. Provide the capability to the firmware to activate all provided resets (hot reset, fundamental reset), independent of other resets. The hardware must provide the proper controls on the reset transitions to prevent failures from being introduced into the platform by the changing of the reset.

i. When a PE is put into the MMIO Stopped and DMA Stopped states, do so in a way that does not introduce failures that might corrupt other parts of the platform.

j. Allow firmware access to internal PHB and I/O fabric PCI configuration registers when any or all of the PEs are in the MMIO Stopped state.

Harware Implementation Notes:

1. The type of error information trapped by the hardware when a PE is placed into the MMIO Stopped and DMA Stopped states is implementation dependent and is beyond the scope of this architecture.

2. A DMA operation (Read or Write) that was initiated before a *Load*, *Store*, or DMA error, does not necessarily need to be blocked because it was not a result of the *Load*, *Store*, or DMA that failed. The normal PCI Express ordering rules require that an ERR_FATAL or ERR_NONFATAL from a failed *Store* or DMA error, or a *Load* Completion with error status must reach the PHB before any DMA that might have been kicked off in error as a result of a failed *Load* or *Store* or a *Load* or *Store* that follows a failed *Load* or *Store*. This means that as long as the PHB processes an ERR_FATAL, ERR_NONFATAL, or *Load* Completion that indicates a failure, before processing any more DMA operations or *Load* Completions, and puts the PE into the MMIO and Stopped DMA Stopped states, implementations can block DMA operations that were kicked off after a failing DMA operation and allow DMA operations that were kicked off before a failing DMA operation without violating the normal PCI Express ordering rules.

3.2.1.4. Error-Injection Hardware Requirements

The error-injection hardware is defined primarily to test enhanced error-recovery software. As implemented in the I/O bridge, this option is used to test the software that implements the recovery that is enabled by the EEH option in that bridge. Specifically, the PAPR *ioa-bus-error* and *ioa-bus-error-64* functions of the *ibm,errinjct* RTAS call are used to inject errors onto each PE primary bus. This, in turn, causes certain actions on the bus and certain actions by the PE, the EEH logic, and by the error recovery software.

The type of errors and the injection qualifiers place the following additional requirements on the hardware for this option.

R1-3.2.1.4-1 Hardware Requirement:

The PHB hardware must take all of the following actions:

a. Provide a way to inject the required errors for each PE primary bus. The errors must be injectable independently without affecting the operations on the other buses in the platform.

b. Provide a way to set up for the injection of the required errors without disturbing operations to other buses outside the PE.

c. Provide firmware with a way to set up the following information for the error injection operation by normal processor *Load* and *Store* instructions:

   - Address at which to inject the error
   - Address mask to mask off any combination of the least-significant 24 (64 for the *ioa-bus-error-64* function) bits of the address
   - PE primary bus number that is to receive the error
   - Type of error to be injected

d. Provide the capability of selecting the errors specified in Table 3.4, “Supported Errors for PCI Express Error Injectors” [14]and an indication of when that error is appropriate for the platform configuration.

e. Provide a way to inject the errors in Table 3.4, “Supported Errors for PCI Express Error Injectors” [14]in a non-persistent manner (that is, at most one injection for each invocation of the *ibm,errinjct* RTAS call).

R1-3.2.1.4-2 Firmware Requirement:

The firmware must limit the injection of errors that are inappropriate for the given platform configuration.

Table 3.4. Supported Errors for PCI Express Error Injectors

Operation	PCI Address Spaces	Errors	Other Requirements
Load	Memory, I/O, Config	TLP ECRC Error	The TLP ECRC covers the address and data bits of a TLP. Therefore, you cannot determine if the integrity error resides in the address or data portion of a TLP.
Store	Memory, I/O, Config	TLP ECRC Error	

Operation	PCI Address Spaces	Errors	Other Requirements
DMA Read	Memory	TLP ECRC Error	The TLP ECRC covers the address and data bits of a TLP. Therefore, you cannot determine if the integrity error resides in the address or data portion of a TLP.
		Completer Abort or Unsupported Request	Inject the error that is injected on a TCE page fault.
DMA Write	Memory	TLP ECRC Error	The TLP ECRC covers the address and data bits of a TLP. Therefore, you cannot determine if the integrity error resides in the address or data portion of a TLP.

3.2.2. DMA Design,TVEs, and TCEs

This section describes the constructs for DMA operations, except when those DMA operation are for MSI. For 64-bit MSI operations, the bits 61:60 in the address are set to 0b01. For 32-bit MSI operations, the bits 31:16 are set to 0xFF. For information about MSI operations, see Section 3.2.4, “MSI Design” [35].

The DMA address for non-MSI operations (that is, for normal DMA operations), is broken up into fields as shown in Figure 3.2, “PCIe Non-MSI DMA Operation Address Fields” [15]. The size of the fields in the address, the number of levels of a TCE table (in the case of the multilevel TCE table), the I/O page size, and the TCE table size are defined by the fields of the TVE (see Table 3.5, “TVE Definition” [23]).

Figure 3.2. PCIe Non-MSI DMA Operation Address Fields

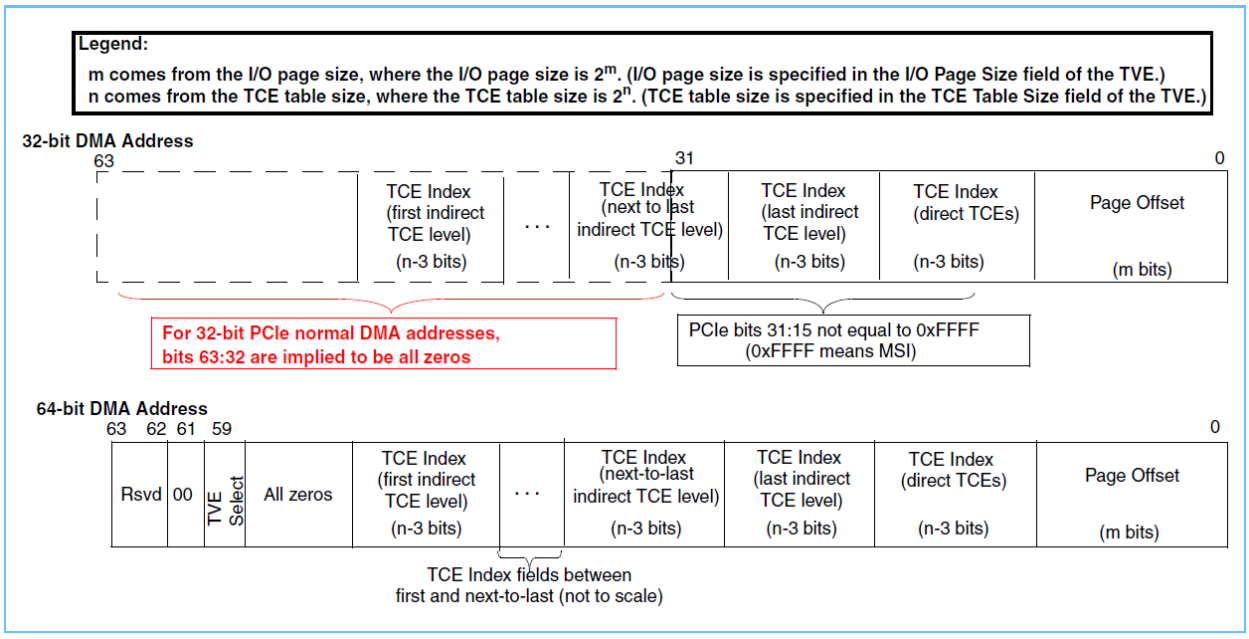


Figure 3.2, “PCIe Non-MSI DMA Operation Address Fields” [15]shows that multiple levels of TCE tables are possible (that is, multiple TCE index levels are shown). Details of single-level tables, for which there exists only the final (direct TCE level) table, are shown in the following sections:

- Section 3.2.2.1, “DMA Design Details: No Page Migration” [16].

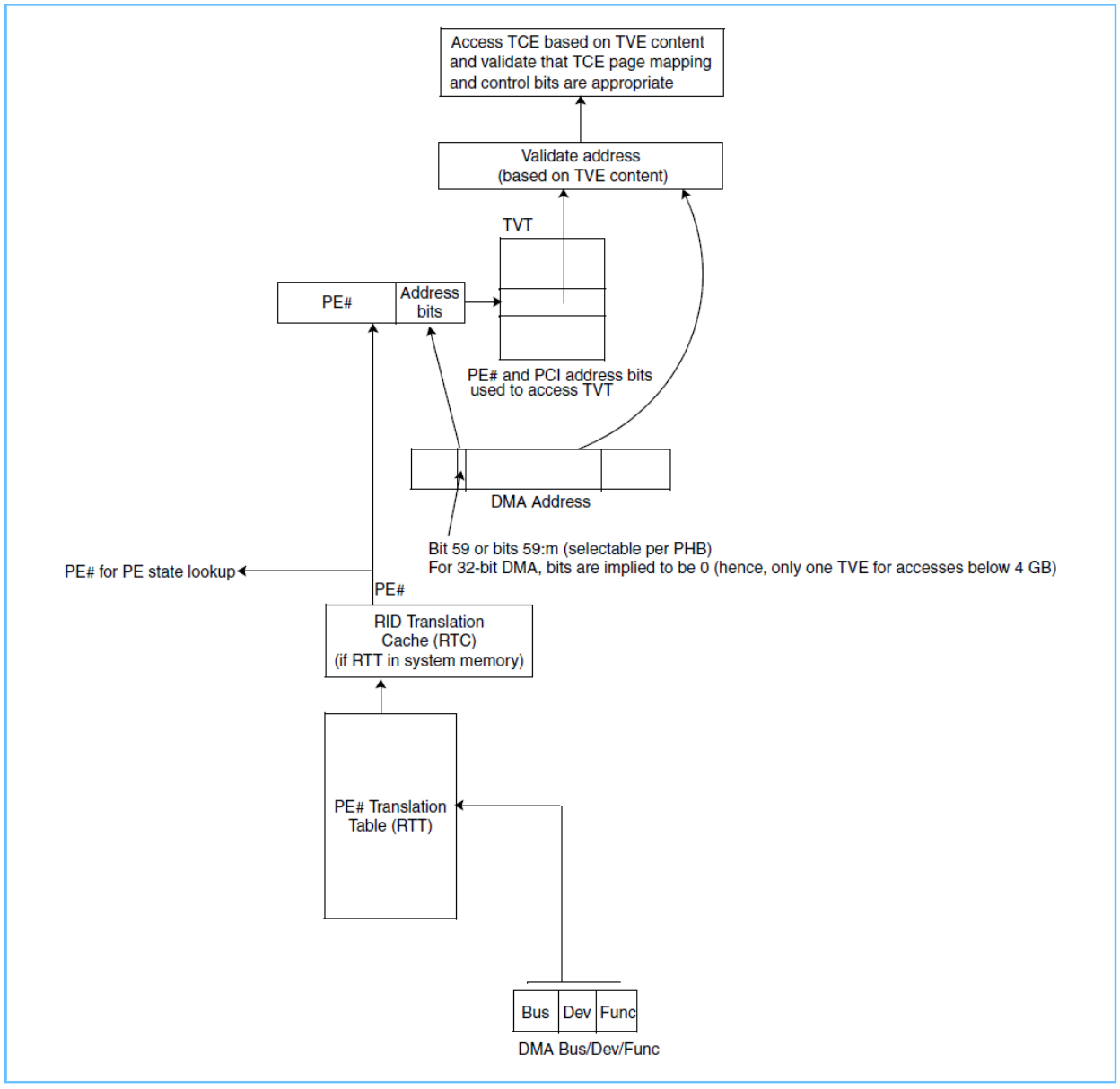
- Section 3.2.2.2, “Additional DMA Design Details: Page Migration” [25].

The differences for multilevel tables are described in Section 3.2.2.3, “Additional DMA Design Details: Multilevel TCE Tables” [30].

3.2.2.1. DMA Design Details: No Page Migration

Figure 3.3, “DMA Operation High-Level Diagram - No Page Migration” [16]shows the general flow of an I/O DMA operation through the system: the address validation, address translation (via TCEs), and caching of the TCEs and data. The description of this figure follows the figure.

Figure 3.3. DMA Operation High-Level Diagram - No Page Migration



Referring to Figure 3.3, “DMA Operation High-Level Diagram - No Page Migration” [16]:

1. The IOA function places the DMA address and RID on the I/O bus.
   - The low-order bits indicate the offset into the page (the page offset). For example, for 4K pages, this is the low-order 12 bits.
   - The bits immediately above the Page Offset are the index into the TCE table (TCE Index). The number of TCE Index bits is determined by the size of the TCE table (that is, the number of TCEs) that is accessible by the RID (not the total size of the TCE table).
   - The RID ties the requester to a particular PE#.
2. The PE# is determined from lookup in the RTC. If it is not in the RTC, it is accessed from the RTT and placed in the RTC.
3. If the PE# is in the DMA Stopped state, abort the operation (see Section 3.2.1.3, “PE State and EEH” [12]).
   - If the operation is a DMA Read Request, return a UR error to the requester.
   - If the operation is a DMA Write, discard the data.
4. The PE# and one or more high-order address bits are used to access the correct TVE for the PE.
   - Bit 59 is used at a minimum. Implementations can allow more bits to be used as an option (for example, bit 59 or bits 59:55). PHBs operate bi-modally relative to the whole PHB, using either one bit or more than one bit, but not both (operation with bit 59 is required).
   - 32-bit DMAs are essentially 64-bit DMAs with the high-order 32-bits set to all zeros. Therefore, for the purpose of TVE selection, all zeros are used for the address selection bits. Thus, only one TVE is available for addresses below 4G.
5. The DMA address is validated to make sure that the RID is allowed to access that I/O bus address. Otherwise, the operation is denied to the IOA function. This is done by the parameters in the TVE in one of two ways:
   - For translate mode, the I/O Page Size and the TCE Table Size are used.
   - For no-translate mode, the address range for no-translate, a base/bounds, is used. See also Appendix B, No-Translate Operation [57].
6. A determination is made about the address of the TCE translation table (TTA from the TVE) to be used to translate the address and the route to the TCE table.

   Note

   This step might follow the next step (determination of whether TCE is already cached) in some implementations.
7. A determination is made about whether the needed TCE is already cached. The caching algorithm must take into consideration the I/O Page Size.

8. If the TCE is not already cached, it is fetched. If it is valid for the operation, it is cached.

   - In the following cases, abort the operation and set the MMIO and DMA Stopped states for the PE:
     - The TCE is invalid (the read-valid and write-valid Page Mapping and Control bits are both 0).
     - The operation is a write and the write-valid bit is off.
     - The operation is a read and the read-valid bit is 0.

9. The PCIe bus address is translated using the information from the TCE and the DMA bus address.

   - The address is translated by using the page offset from the DMA bus address as the low-order bits of the translated address in the TCE.

Note

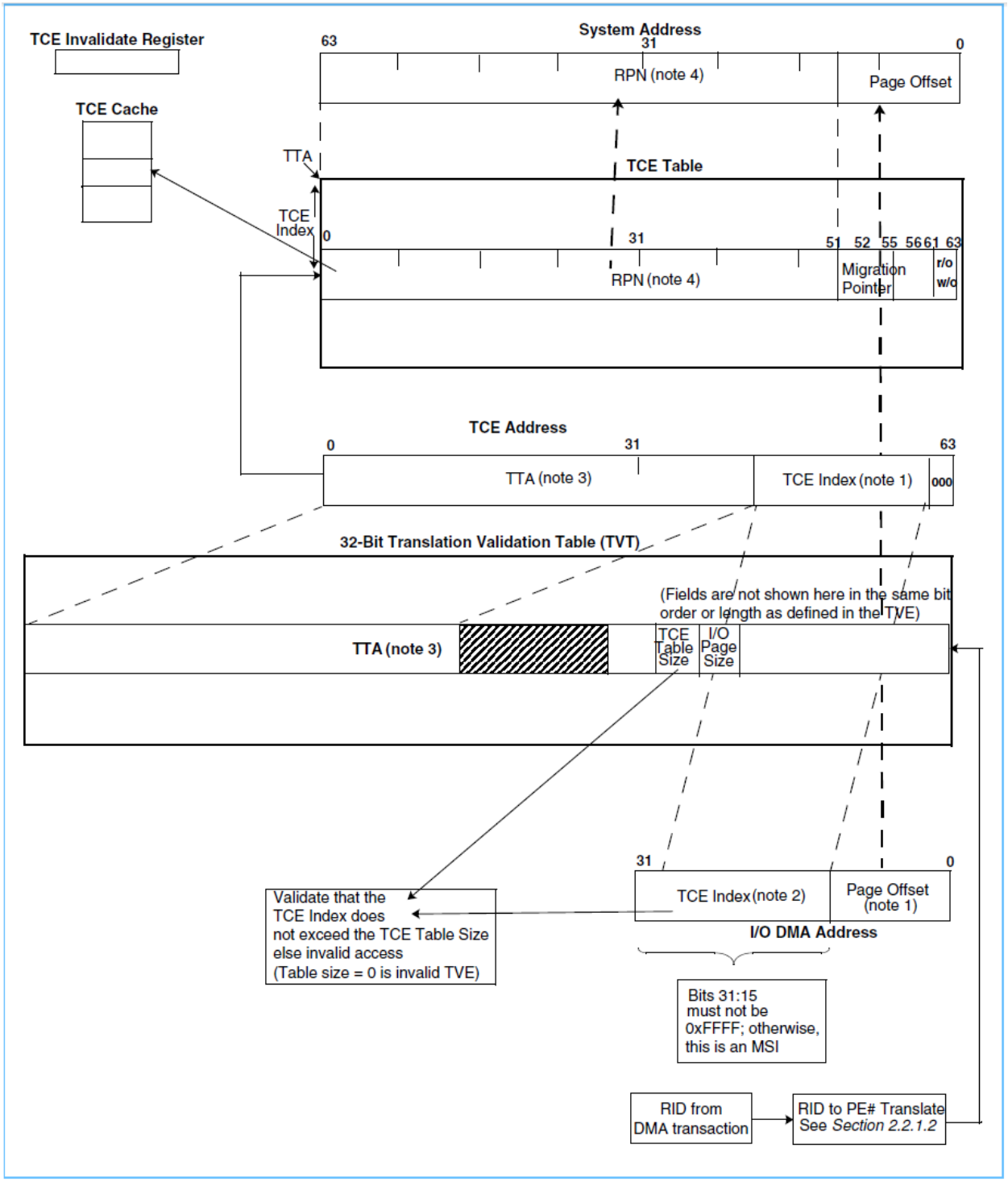
The number of page offset bits is determined by the I/O Page Size. The I/O Page Size is specified in the I/O Page Size field of the TVE (see Table 3.5, “TVE Definition” [23]).

   - The high-order bits are obtained from the Real Page Number (RPN) in the TCE.

For more information, see:

- Figure 3.4, “I/O Address Validation and TCE Translation Implementation for 32-Bit DMA Addresses” [19], which shows how the translation from I/O address to system memory address works for 32-bit I/O addresses
- Figure 3.5, “I/O Address Validation and TCE Translation Implementation for 64-Bit DMA Addresses” [21], which shows how the translation from I/O address to system memory address works for 64-bit I/O addresses

Figure 3.4. I/O Address Validation and TCE Translation Implementation for 32-Bit DMA Addresses



Notes on Figure 3.4, “I/O Address Validation and TCE Translation Implementation for 32-Bit DMA Addresses” [19]and Figure 3.5, “I/O Address Validation and TCE Translation Implementation for 64-Bit DMA Addresses” [21]:

1. The Number of Page Offset bits is determined by the I/O Page Size. That is, the boundary between the TCE Index field and the Page Offset field slides right and left depending on the I/O Page Size. This architecture does not require hardware verification of I/O page sizes of anything other than 4 KB for 32-bit DMA operations (that is, for DMAs with an address less than 4 GB).

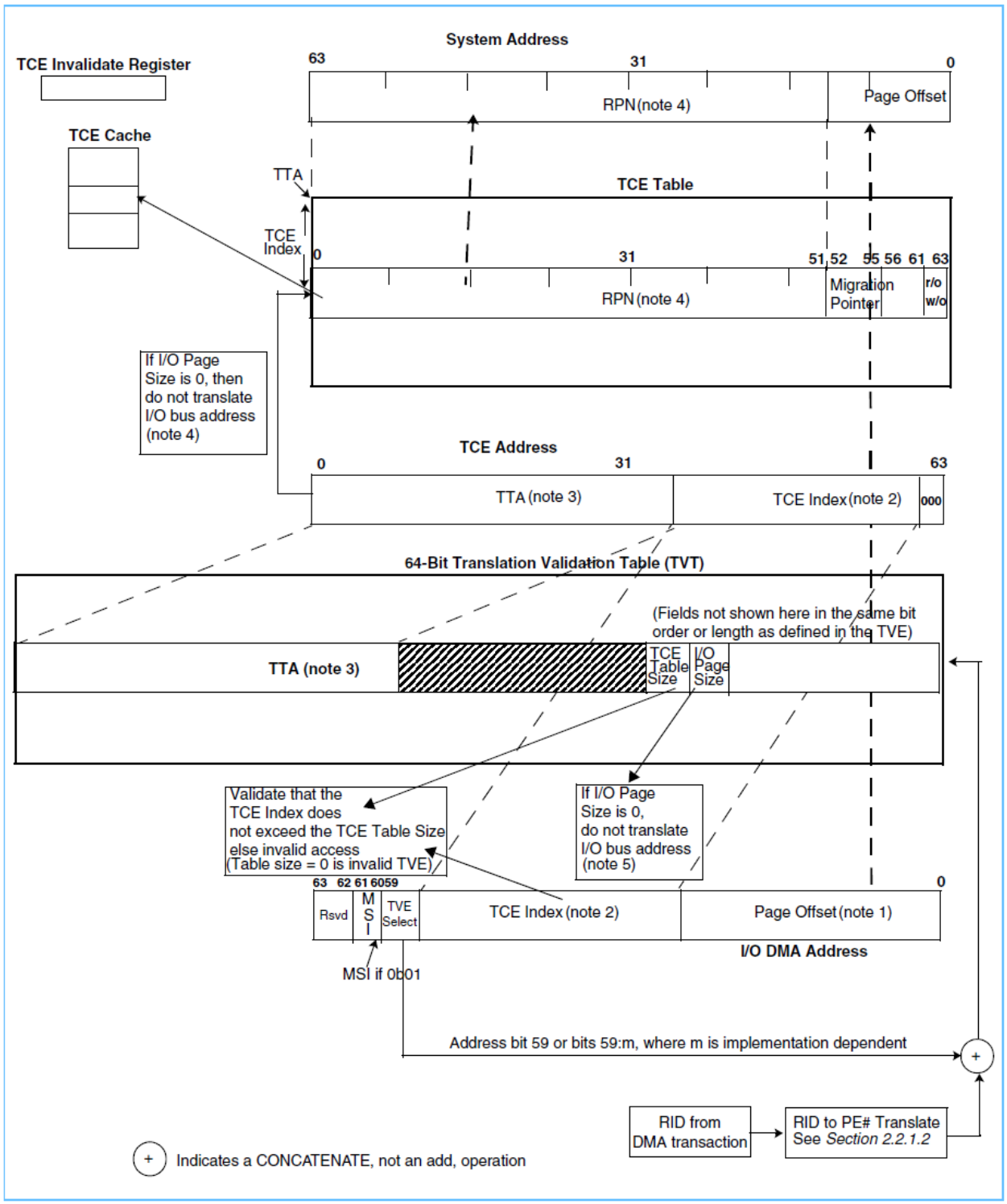
2. The TCE Index field size is based on the number of pages to which the IOA function has access.

3. The Number of TTA bits used in the least-significant bits is determined by the TCE Index field size. The number of TTA bits implemented in the most-significant bits is dependent on the maximum size of system memory to be supported by the platform.

4. The number of most significant RPN bits implemented in the TCE is dependent on the maximum size of system memory to be supported by the platform. The number of least significant RPN bits used depends on the number of Page Offset bits (that is, on the size of the page mapped by the TCE, as determined from the I/O Page Size field in the TVE). Also, if the I/O Page Size is zero in the TVE, the I/O Bus Address is used untranslated to access the system. For more information, see the definition for the “Address range for no translate” field in Table 3.5, “TVE Definition” [23].

5. An implementation can choose to check that the DMA address bits (63:62) are 0b00, that bits 61:60 are not 0b10 or 0b11, and that unused TVE select bits (per mode) are all zero. If any of these conditions are not true, an implementation can choose to abort the operation and set the MMIO and DMA Stopped states for the PE. However, the implementation is not required to do so.

Figure 3.5. I/O Address Validation and TCE Translation Implementation for 64-Bit DMA Addresses



When the TVE entry has an I/O Page Size other than zero, the TVE associates a DMA address range, which is based on the high-order address bits used with the PE# to select the TVE, with a TCE table starting address (TTA), TCE table size, and an I/O page size. When the TVE entry has an I/O Page Size of zero, TCE translation is not performed, and the TTA field becomes an address range check.

R1-3.2.2.1-1 Hardware Requirement:

The PHB hardware must take all of the following actions:

a. Implement the TVE table, as defined by Table 3.5, “TVE Definition” [23], with the TVT being located on the PHB chip.

b. Implement the TCE table, with entries as defined by Table 3.6, “TCE Definition” [24], with the TCE table being in system memory and cached on the PHB chip.

c. Provide a TCE Invalidate Register, as defined in Table 3.7, “TCE Invalidate Register Definition” [25], for invalidating cached TCEs, all TCEs for a particular PE, or the entire cache of TCEs. The hardware must stop using the entry when firmware indicates the invalidate, but it can wait until the TCE is used once by a DMA operation. A *Store* to this register causes the specified operation. Issuing a *Load* to this register causes the last value *Stored* to be returned.

d. Implement the DMA flows as shown in Figure 3.3, “DMA Operation High-Level Diagram - No Page Migration” [16], Figure 3.4, “I/O Address Validation and TCE Translation Implementation for 32-Bit DMA Addresses” [19], and Figure 3.5, “I/O Address Validation and TCE Translation Implementation for 64-Bit DMA Addresses” [21].

R1-3.2.2.1-1 Firmware Requirement:

The firmware must take all of the following actions:

a. Set up the TVEs appropriately.

b. Access the TVE table with 8-byte *Loads* and *Stores*, naturally aligned.

c. Set up the TCEs appropriately.

d. Maintain the coherency between the system memory TCE value and the cached TCE value by using the TCE Invalidate Register to invalidate cached TCEs whenever it changes the value of a TCE.

In Table 3.5, “TVE Definition” [23], [ ] designates optional bits and bytes. Optional bits and bytes that are not implemented must be ignored by that implementation on a *Store* and must be returned as zeros on a *Load*, even when the entire field is not implemented. Implementations that do not implement the full size of the field must treat unused bits and bytes the same as optional bits and bytes. Reserved bits and bytes must be ignored on a *Store* and must be returned as zeros on a *Load*.

Note

A nonzero TCE Table Size field indicates a valid TVE.

Table 3.5. TVE Definition

Bytes	Bits	Field	Definition
0:5	All	TTA or Address range for no translate	When the I/O Page Size field is nonzero and the TVE is valid (TCE Table Size is nonzero), this field is the TCE Table Address (TTA). Bit 0 of the TVE aligns with system address bit 4, bit 1 with system address bit 5, and so on. TCE tables must be aligned on a boundary that is an integer multiple of their size, and, therefore, depend on the size of the table and the TCE size. Some of the low-order bits of this field might not be needed and must be set to zero by the software. Hardware pads this field with zeros, if necessary, in the case of large I/O page sizes where the TCE table is smaller than 4 KB. Likewise, only enough high-order bits need to be implemented by the hardware to match the largest real address in the platform. The minimum alignment is at least 4 KB. When the I/O Page Size field is zero (no translate case) and the TVE is valid (TVE[byte 6, bit 3] = 1), then if PCI Express address [bits 49:24] ≥ (TVE[byte 6, bits 4:5] concatenated with TVE[bytes 0:2]) and PCI Express address [bits 49:24] < (TVE[byte 6, bits 6:7] concatenated with TVE[bytes 3:5]), then use the PCI Express address [bits 49:0], untranslated, as the DMA address. **Notes**: 1. The no-translate case is not valid for 32-bit PCI Express addresses. 2. The no translate and translate cases have different alignment requirements. For the translate case, the size of the area translated by the TCE table dictates the alignment requirements; it must be aligned on an integer multiple of the size. However, for the no translate case, the alignment is 16 MB or larger. 3. See also Appendix B, No-Translate Operation [57].
6	0:2	Number of TCE Table Levels	This field indicates the number of indirect TCE table levels for operations using this TVE, which is the total number of TCE table levels (including the last level) minus 1. When this field is zero, there are no indirect levels. See also Section 3.2.2.3, “Additional DMA Design Details: Multilevel TCE Tables” [30]. The following values are defined by this architecture: 000 - Only one level (direct level) 001 - One indirect level, one direct level 010 - Two indirect levels, one direct level 011 - Three indirect levels, one direct level 100 - Four indirect levels, one direct level 101-111 Reserved
6	3:7	TCE Table Size	A value of zero in this field indicates that the TVE is invalid. When the I/O Page Size field is nonzero (translate case) and the TVE is valid (that is, this field is nonzero), the value of this field defines the number of DMA I/O bus address bits that are used for the TCE index field or fields (see Figure 3.2, “PCIe Non-MSI DMA Operation Address Fields” [15]). The hardware uses the value of this field, along with the Number of TCE Table Levels and I/O Page Size fields of this TVE, to validate the range of the DMA I/O address. That is, it validates that the appropriate number of high-order bits in the DMA I/O bus address are zero. It also uses the value of this field to prevent an IOA function from accessing outside of its address range. Value of 0: Invalid TVE Value of 1: 9 TCE Index bits, 4 KB table size Value of 2: 10 TCEIndex bits, 8 KB table size

Bytes	Bits	Field	Definition
			Value of 3: 11 TCEIndex bits, 16 KB table size Value of 4: 12 TCEIndex bits, 32 KB table size Value of 5: 13 TCEIndex bits, 64 KB table size Value of 6: 14 TCEIndex bits, 128 KB table size Value of 7: 15 TCEIndex bits, 256 KB table size Value of 8: 16 TCEIndex bits, 512 KB table size Value of 9: 17 TCEIndex bits, 1 MB table size Value of 10: 18 TCEIndex bits, 2 MB table size Value of 11: 19 TCEIndex bits, 4 MB table size Value of 12: 20 TCEIndex bits, 8 MB table size Value of 13: 21 TCEIndex bits, 16 MB table size Value of 14: 22 TCEIndex bits, 32 MB table size Value of 15: 23 TCEIndex bits, 64 MB table size Value of 16: 24 TCEIndex bits, 128 MB table size Value of 17: 25 TCE Index bits, 256 MB table size Value of 18: 26 TCE Index bits, 512 MB table size Value of 19: 27 TCE Index bits, 1 GB table size Value of 20: 28 TCE Index bits, 2 GB table size Value of 21: 29 TCE Index bits, 4 GB table size Value of 22: 30 TCE Index bits, 8 GB table size Value of 23: 31 TCE Index bits, 16 GB table size Value of 24: 32 TCE Index bits, 32 GB table size Value of 25: 33 TCE Index bits, 64 GB table size Value of 26: 34 TCE Index bits, 128 GB table size Value of 27: 35 TCE Index bits, 256 GB table size Value of 28: 36 TCE Index bits, 512 GB table size Value of 29: 37 TCE Index bits, 1 TB table size Value of 30: 38 TCE Index bits, 2 TB table size Value of 31: 39 TCE Index bits, 4 TB table size When the I/O Page Size field is zero (no translate case) and the TVE is valid (that is, this field is non-zero), this indicates that the TVE bytes [0:5] are to be used to validate the PCI Express address (see the definition for the “Address range for no translate” field of this table).
7	[0:2]	Reserved	Reserved
	3:7	I/O Page Size	The number of low-order I/O bus address bits to be used as the page offset (see Page Offset field in Figure 3.2, “PCIe Non-MSI DMA Operation Address Fields” [15]) is the value of this field plus 11. Zero has a special meaning of no translate. Value of 0: No address translation (see the definition for the “Address range for no translate” field of this table). Other examples: Value of 1: Use 11 + 1 = 12 bits (4 KB I/O page size) Value of 5: Use 11 + 5 = 16 bits (64 KB I/O page size) Value of 17: Use 11 + 17 = 28 bits (256 MB I/O page size) All page sizes that are supported by the processor used with this PHB must be supported as I/O page sizes, but validation can include only the page sizes that the implementation expects to be used. This architecture does not require hardware verification of I/O page sizes of anything other than 4 KB for 32-bit DMA operations (that is, for DMAs with an address less than 4 GB).

Table 3.6. TCE Definition

Bits	Field	Definition
0:51	RPN	If the Number of TCE Table Levels field of the TVE is zero, or if it is nonzero and this is the final TCE table level to be accessed, and if the Page Mapping and Control field of this TCE is something other than page fault, then these bits contain the RPN to which the bus address is mapped in the system address space. In certain PHB implementations, all of these bits might not be required. However, enough bits must be implemented to match the largest real address in the platform.

Bits	Field	Definition
		If the Number of TCE Table Levels field of the TVE is zero, or if it is nonzero and this is the final TCE table level to be accessed, and if the Migration Descriptor Pointer is non-zero, then this is the RPN for the source page. The destination page RPN comes from the Migration Register pointed to by the Migration Descriptor Pointer. If the Number of TCE Table Levels field of the TVE is nonzero, and this is not the final TCE table level to be accessed, and if the Page Mapping and Control field of this TCE is something other than page fault, then these bits contain the address for the start of the next level of TCE table. For more information, see Section 3.2.2.3, “Additional DMA Design Details: Multilevel TCE Tables” [30].
52:55	Migration Pointer	Used during a migration operation. The meaning is as follows: 0b0000 A migration is not in process for this page. nonzero A migration is in progress, and the value of this field points to which Migration Register in the PHB is used for the operation. For more information, see Section 3.2.2.2, “Additional DMA Design Details: Page Migration” [25].
56:61	Reserved	Reserved
62:63	Page Mapping and Control	These bits define page mapping and read-write authority. They are coded as follows: 00 Page fault (no access) 01 System address space (read only) 10 System address space (write only) 11 System address space (read/write) Code point 0b00 signifies that the page is not mapped. It must be used to indicate a page fault error. Hardware must not change its state based on the value in the remaining bits of a TCE when code point 0b00 is set in this field of the TCE. For accesses to system address space with an invalid operation (a write or PCI Atomic operation to a read-only page or a read or PCI Atomic operation to a write-only page), the HB generates an error. For more information, see the “Error and Event Notification” chapter in the PAPR.

Table 3.7. TCE Invalidate Register Definition

Bits	Field	Definition
0:2	Invalidate Operation	Specifies the scope of the invalidate operation. The following values are valid, where “x” is a don’t care bit: 0b1xx: Invalidate the entire TCE cache. 0b01x: Invalidate all TCEs for the specified PE#. 0b001: Invalidate the TCE for the specified PCIe bus address and PE#.
3	Reserved	Reserved
4:51	Invalidate Address	PCIe address for the specified PE# for which the corresponding direct TCE is to be invalidated, I/O page aligned. Register bit 4 aligns with bit 59 of the PCIe address, and register bit 51 aligns with bit 12 of the PCIe address. Because this field is I/O page aligned, some of the low-order bits of this field must be 0 for I/O page sizes larger than 4 KB. This field is required for Invalidate Operation 0b001, and not required for 0b01x or 0b1xx.
52:55	Reserved	Reserved
56:63	PE#	PE# of the cache entry or entries to invalidate. This field is required for Invalidate Operations 0b01x and 0b001, and not required for 0b1xx.

3.2.2.2. Additional DMA Design Details: Page Migration

The page migration facilities in the PHB hardware give the firmware the tools necessary to keep DMA operations going while copying a memory page from a source location to the target location. These facilities consist of the following components:

- A Migration Pointer field in the TCE. When nonzero, it indicates that a page migration is in progress from the page pointed to by the TCE (called the *source* page). It also points to one of the Migration Registers in the PHB hardware.

- Migration Register in the PHB hardware. It specifies:

  —Target RPN: A field specifying the page number of the destination page to which the source page is being moved (called the *target* page).

  —Target Page Size: A field specifying the I/O page size of the target page, which might be greater than or equal to the source page size. The I/O page size of the source page is in the TVE.

  The Migration Register provides the capability to have one migration descriptor that describes the target, regardless of how many different pages of different sizes are mapped to the source page.

  For example, a 64 KB I/O page with multiple 4 KB I/O pages mapped into that 64 KB I/O page.

  —Read Target: A bit specifying the page to use as the source of DMA data for a DMA read request; it can be the source page or the target page.

On a DMA operation, the hardware does what is specified in Hardware Requirement R1-3.2.2.2-1 (p. 25).

A graphical representation of the memory migration operation is shown in Figure 3.6, “Memory Migration Operation for a 64 KB Page and a 4 KB Page within the 64 KB Page” [27]. A graphical representation of the address translation is shown in Figure 3.7, “Source and Destination Page Address Creation for DMA to a Page Being Migrated” [28].

The target page address translation of “Bits 4:(63-N) of the Migration Register || TCE-translated source page address bits (64-N):63”, as described in Hardware Requirement R1-3.2.2.2-1 (p. 25), can also be described in a different way, as indicated in Figure 3.6, “Memory Migration Operation for a 64 KB Page and a 4 KB Page within the 64 KB Page” [27]and Figure 3.7, “Source and Destination Page Address Creation for DMA to a Page Being Migrated” [28], and as described, below:

- If the Target Page Size in the Migration Register is the same as the TVE I/O Page Size field for the operation, replace the RPN bits from the TCE in the source page address with the corresponding address bits from the Target RPN bits in the Migration Register.

- If the Target Page Size in the Migration Register is larger than the TVE I/O Page Size field for the operation, keep the low-order bits from the RPN from the TCE that represent the source page offset within the target page. Replace the remaining RPN bits from the TCE in the TCE Translated PCIe Address with the corresponding address bits from the Target RPN bits in the Migration Register.

Figure 3.6. Memory Migration Operation for a 64 KB Page and a 4 KB Page within the 64 KB Page

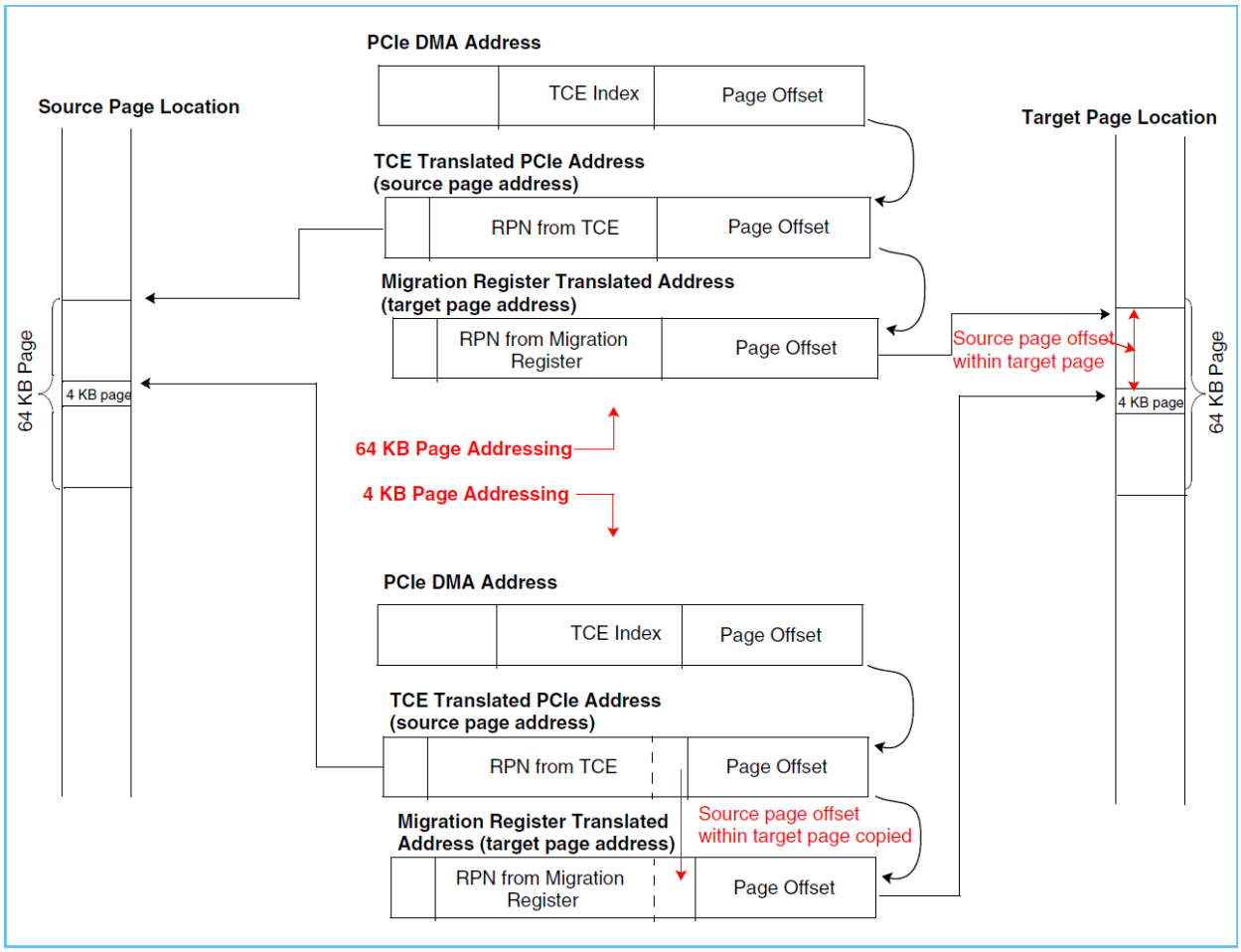
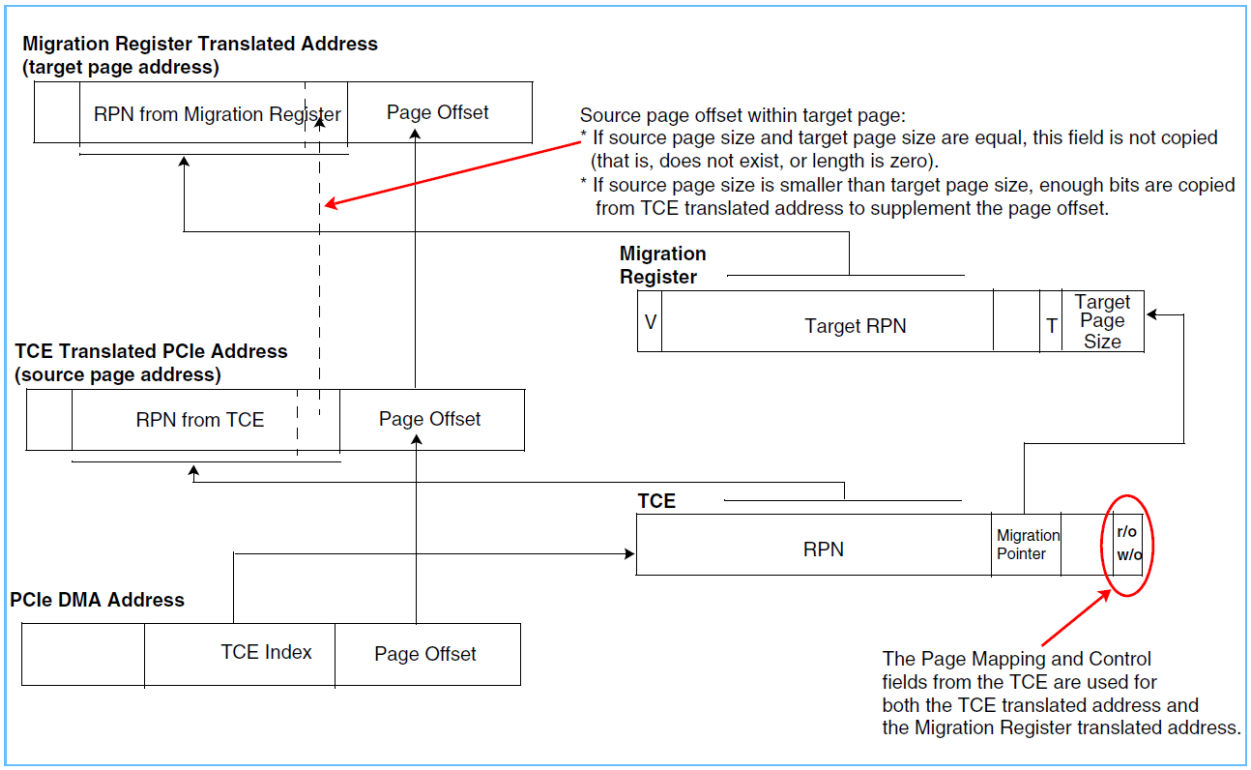


Figure 3.7. Source and Destination Page Address Creation for DMA to a Page Being Migrated



R1-3.2.2.2-1 Hardware Requirement:

The PHB hardware must take all of the following actions:

a. Implement a set of Migration Registers, with the definition in Table 3.8, “Migration Register Definition” [30]. The number of registers implemented is implementation dependent, but the minimum number is 7.

b. Use the Page Mapping and Control field from the TCE for both source page and target page operations.

c. Implement the Migration Pointer field of the TCE, as specified in Table 3.6, “TCE Definition” [24]. When that field is nonzero, perform the operations described by parts d [28]through i [29] of this requirement.

d. Access the TCE for the operation, as for a normal DMA operation (that is, as per Section 3.2.2.1, “DMA Design Details: No Page Migration” [16]). Calculate the source page address as usual.

e. Calculate the migration target page address. If N is the value in the Migration Register Target Page Size field, the address is generated by:

   Bits 4:(63 - N) of the Migration Register || TCE-translated source page address bits (64 - N):63

f. If the operation is a DMA Read and the Read Target bit in the Migration Register = 0, access at the Migration Source page address in the translation page address.

g. If the operation is a DMA Read and the Read Target bit in the Migration Register = 1, access at the Migration Target page address.

h. If the operation is a DMA Write, write the data to the source page at the TCE-translated page address. After this first write is visible to all other processors and mechanisms, write the data to target page address.

i. Prevent a PCIe atomic operation that targets a page being migrated from being performed until the migration operation is completed against the page being targeted by the PCIe atomic operation (that is, until the Migration Pointer in the TCE for the page is set to 0). Perform this atomic operation blocking without blocking DMA Read and DMA write requests.

R1-3.2.2.2-1 Firmware Requirement:

The firmware must take the following actions during a page migration, in basically the following order:

a. Allocate a Migration Descriptor Register in each PHB with TCE accessibility to the source page, for use in the migration.

b. Build the Migration Descriptor Register content for the physical page of the memory area to be migrated, and set the Read Target bit to 0. If there are multiple page mappings that map the physical page, there only needs to be one Migration Descriptor Register set up for all the page mappings (for example, a 64 KB page migration with multiple 4 KB pages mapped within that 64 KB page). However, firmware must assure that the Target Page Size field in the Migration Register is equal to the largest I/O page size being migrated.

c. Store the contents built in part b of this requirement into the Migration Descriptor Register allocated in part a of this requirement.

d. Redirect all TCEs pointing to the memory area to be migrated to relevant Migration Descriptor Registers. Then use the TCE Invalidate facility to invalidate any cached versions of the TCE (as per the usual TCE change process).

e. Copy data from the source to the destination memory region. For each atomically writable quantum of memory:
   - Read quantum from the migration source page.
   - Write quantum to the corresponding offset in the migration destination page.
   - Reread quantum from the migration source page.
   - Compare the first and second reads (this catches a DMA write race).
   - If not equal, branch back and copy again.
   - Else, loop to the next quantum.

f. Set the Read Target Bit to 1 in each Migration Descriptor Register allocated in part a of this requirement.

g. Adjust all TCEs changed in part d of this requirement to directly access their migration destination pages. Set the Migration Pointer in those TCEs to 0. Then, use the TCE Invalidate facility to invalidate any cached versions of the TCE (as per the usual TCE change process).

Table 3.8. Migration Register Definition

Bits	Field	Definition
0	Valid	0: Invalid. Attempts to use causes the PE to be put into the EEH Stopped 1: Valid
1:3	Reserved	Reserved
4:51	Target RPN	The Page Mapping and Control field is inherited from the TCE for the source page. If the Page Mapping and Control field is something other than a page fault, these bits contain the RPN used for the target page translation. In certain PHB implementations, all of these bits might not be required. However, enough bits must be implemented to match the largest real address in the platform.
52:56	Reserved	
57	Read Target	0: DMA Read operations are made to the source page. 1: DMA Read operations are made to the target page.
58:63	Target Page Size	The target page size is $2^{\text{Contents of this field}}$. The values allowed are implementation dependent, but correspond to the same I/O page sizes that are implemented for the I/O Page Size field of the TVE (see Table 3.5, “TVE Definition” [23]).

3.2.2.3. Additional DMA Design Details: Multilevel TCE Tables

This section defines the changes to typical DMA operations described previously when there are multiple levels of TCE tables; that is, when the Number of TCE Table Levels field of the TVE is nonzero.

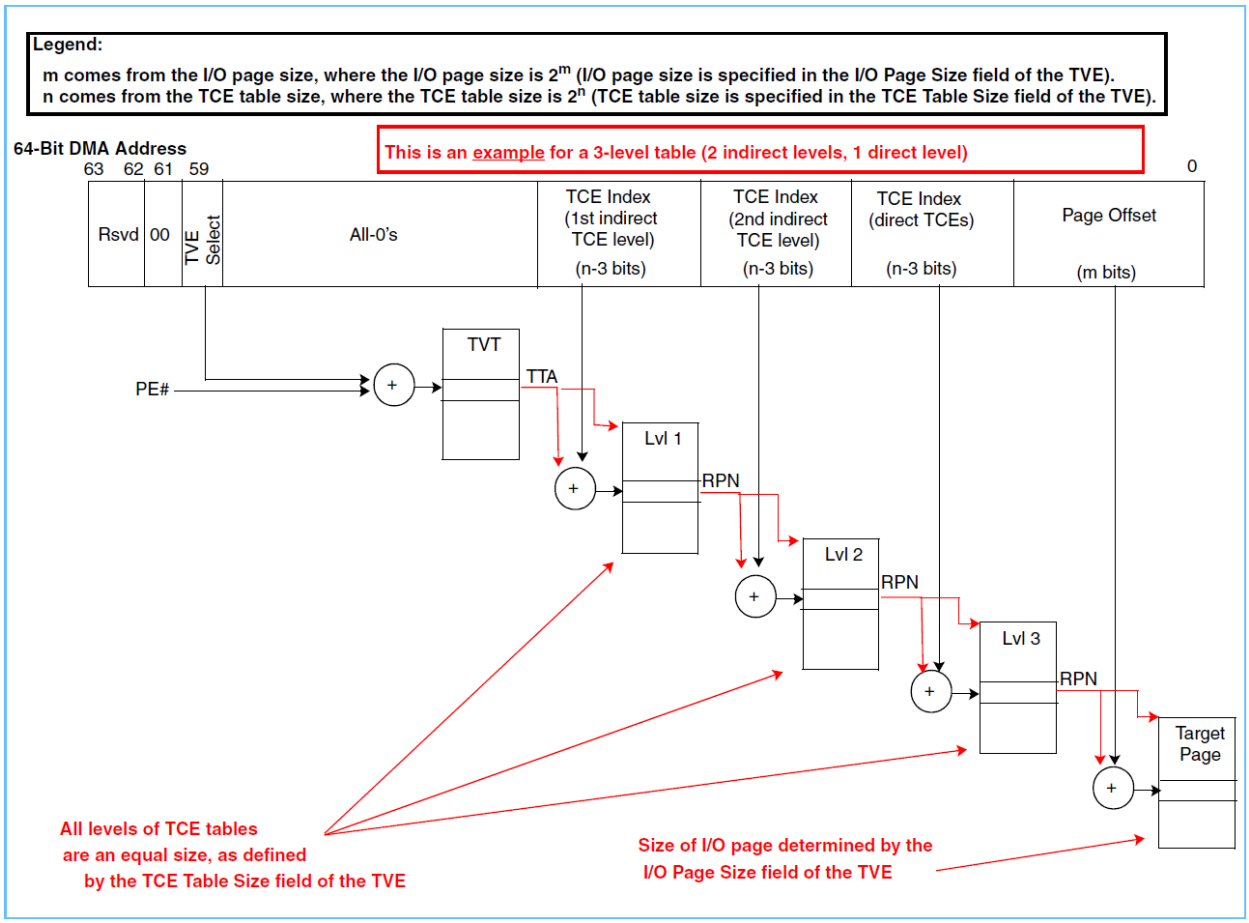
The following terms are used in the description of multilevel TCE tables:

Direct TCE table	Contains direct TCEs.
Direct TCE	A direct TCE contains the real page number (RPN) that points to the page that the DMA operation is trying to access.
Indirect TCE table	Contains indirect TCEs.
Indirect TCE	An indirect TCE is a TCE for which the RPN contains a pointer to the starting address for the next level of the TCE table. This pointer is used along with the TCE Index for the next level of table to access the next TCE in the chain of TCEs. The TCE Index is obtained from the PCIe address, as shown in Figure 3.2, “PCIe Non-MSI DMA Operation Address Fields” [15]. For a description of the use of the indirect TCE, see Section 3.2.2.3.1, “Multilevel Table TCE Fetching” [30].

3.2.2.3.1. Multilevel Table TCE Fetching

When the Number of TCE Table Levels field of the TVE is nonzero, multiple TCE fetches are made by the hardware when that TVE is used. The number of fetches is equal to the Number of TCE Table Levels field of the TVE plus 1, unless one of the TCEs indicates a page fault. In that case, the operation is marked in error and the fetching stops. All but the last level of tables contains what is called indirect TCEs, and the last level contains direct TCEs. The PHB uses the RPN field of indirect TCEs to point to the start of the next level of TCE table. The TCE index is obtained from the PCIe address as shown in Figure 3.2, “PCIe Non-MSI DMA Operation Address Fields” [15]. An example of this process for a three-level table is shown in Figure 3.8, “PCIe Normal DMA Operation for a Three-Level TCE Table” [31].

Figure 3.8. PCIe Normal DMA Operation for a Three-Level TCE Table



R1-3.2.2.3-1 Hardware Requirement:

The PHB hardware must take all of the following actions:

a. Implement multilevel TCE tables as defined by Section 3.2.2.3, “Additional DMA Design Details: Multilevel TCE Tables” [30].

b. Implement the Number of TCE Table Levels field according to the definition in Table 3.5, “TVE Definition” [23]. Use this value to determine when an indirect TCE table is being accessed and when a direct TCE table is being accessed.

c. Treat a Page Fault setting (0b00) of the TCE Page Mapping and Control field the same for both direct and indirect TCEs. That is, fail the operation and set the EEH Stopped state for the PE.

d. Except for the Page Fault setting (0b00) of the TCE Page Mapping and Control field, ignore any read-only or write-only setting for these bits in indirect TCEs. That is, the values of 0b01, 0b10, and 0b11 are to be treated the same for indirect TCEs; they are all valid states for any DMA operation.

e. Treat the TCE Table Size of the TVE as the size of each level of TCE table being accessed by the TTA of the TVE or by the RPN of an indirect TCE.

f. Validate that the All-0's field of the PCIe address, shown in Figure 3.8, "PCIe Normal DMA Operation for a Three-Level TCE Table" [31], is all zeros, based on the TCE Table Size, Number of TCE Table Levels, and I/O Page Size fields of the TVE.

R1-3.2.2.3-2 Firmware Requirement:

The firmware must take all of the following actions:

a. Set up the Number of TCE Table Levels field appropriately for each TVE.

b. Set up the indirect TCE tables appropriately such that each indirect TCE points to the start of the next level TCE table to be accessed. Set the Page Mapping and Control field for used indirect TCEs to something other than the Page Fault (0b00) setting.

3.2.2.3.2. Multilevel Table TCE Caching

To enable the cache to match on accesses from the same device to the same I/O page, the PCIe address must be in the cache, and not the address of the direct TCE itself. By doing this, no intermediate (indirect) fetches are made if the PCIe address hits a cached entry.

Implementations might also want to consider caching other levels. However, the performance gained must be traded off against the silicon area and based on the expected workload.

3.2.2.4. DMA Read Sync Register

The DMA Read Sync Register shown in Table 3.9, "DMA Read Sync Register" [32]is provided to assist firmware in determining when all currently outstanding (in progress relative to the PHB's state machine) DMA read operations are complete. For example, it can be used during a memory-migration operation or during PE-reset operations to assure that in-flight DMAs are complete.

Table 3.9. DMA Read Sync Register

Bits	Field	Access Mode	Definition
0	Start Synchronization	Write only	Writing a 0 to this bit has no effect. Writing a 1 to this bit starts the DMA read synchronization process. Writing a 1 to this bit sets the Synchronization Complete bit (bit 1 of this register) to a 0 at least until the hardware determines if there are currently any incomplete DMA read operations. Reading this bit returns a 0.
1	Synchronization Complete	Read only	0: One or more DMA read operations, which were incomplete (outstanding)[a] when the Start Synchronization bit (bit 0 of this register) was last written to a 1, are still not complete. 1: All DMA read operations, which were incomplete (outstanding)[a] when the Start Synchronization bit (bit 0 of this register) was written to a 1, if any, have been completed. Writes to this bit are ignored.
2:63	Reserved	-	Reserved

[a]A DMA Read operation is considered to be incomplete (outstanding) if processing on it has been started by the PHB's state machine, such that the address translation process has started and cannot be aborted. For example, if the PHB has started the operation to read the TCE from system memory, the DMA read operation is considered outstanding at that point if the operation cannot be stopped and restarted. The operation remains outstanding (and the Synchronization Complete bit remains at 0) until the DMA Read operation data (not just the TCE data) is received by the PHB from system memory.

R1-3.2.2.4-1 Hardware Requirement:

The PHB hardware must provide the DMA Read Sync register, as defined in Table 3.9, “DMA Read Sync Register” [32].

3.2.3. LSI Design

When an LSI is signalled by the Assert_INTx message request (where x is A, B, C, and D for respective PCI interrupt signals) and when that interrupt was not previously signalled, the PHB selects one of four XIVEs from the LSI XIVT to determine the appropriate interrupt to assert [1] to the system. (INTA selects the first entry, INTB the second, and so on.) The interrupt state is also presented in one of four interrupt state entries (ISEs) in the LSI interrupt state table (IST), which is associated with the LSI XIVT. See Section 3.2.3.1, “LSI XIVE Definition” [33]and Section 3.2.3.2, “LSI ISE Definition” [34].

Use of LSI interrupts in systems has the following implications for firmware and hardware:

- There are only four LSIs per PHB, and any specific interrupt cannot be shared across PEs. This limitation is enforced by firmware. Therefore, this limit severely restricts the hardware configurations available.
- No validation is done on the Assert_INTx message requests that come in to the PHB. See also Requirement R1-3.2.3.1- c.

Unlike MSI interrupts, when the PE is in the DMA Stopped state, LSI operations from that PE are not prevented. The firmware can disable the interrupts on detection of an EEH event. On detection of the event, the DD should also disable its IOA function’s interrupt in its function’s PCI configuration space.

3.2.3.1. LSI XIVE Definition

The external interrupt vector tables (XIVTs), which contain the external interrupt vector entries (XIVEs), are located in the PHB chip.

In Table 3.10, “XIVE Definition for LSI Interrupts Only” [33], [ ] designates optional bits and bytes. Optional bits and bytes that are not implemented must be ignored by that implementation on a *Store* and must be returned as zeros on a *Load* even when the entire field is not implemented. Implementations that do not implement the full size of the field must treat unused bits and bytes the same as optional bits and bytes. Reserved bits and bytes must be ignored on a *Store* and must be returned as zeros on a *Load*.

Table 3.10. XIVE Definition for LSI Interrupts Only

Bytes	Bits	Field	Definition
0:2	All	Server #	The number of bits implemented is implementation dependent. At a minimum, it is necessary to have one server per processor thread for a global queue and one server per processor thread for a local queue, plus some room for growth. The suggested minimum is 20 bits. The maximum is 24 bits. It is useful for this field to be writable independent of other fields in XIVE.
3	All	Interrupt Priority	A value of 0x00 is the highest priority, and a value of 0xFE is the lowest priority. A value of 0xFF means the interrupt is disabled and should not be presented at the current time.
[4:7]	All	Reserved	Reserved

[1] Unless the Interrupt Priority field in the XIVE is 0xFF, which means that the interrupt is disabled.

R1-3.2.3.1-1 Hardware Requirement:

The PHB hardware must implement the XIVT for LSI interrupts, and the entries in this table must be as defined in Table 3.10, “XIVE Definition for LSI Interrupts Only” [33].

R1-3.2.3.1-2 Firmware Requirement:

The firmware must take all of the following actions:

a. Access the XIVT for LSI interrupts with 8-byte Loads and Stores, naturally aligned.

b. Not allow interrupts to be shared between PEs.

c. Prevent a partition from setting up an LSI on an IOA owned by it, if that LSI is not owned by that partition.

3.2.3.2. LSI ISE Definition

The interrupt state tables (ISTs), which contain the interrupt state entries (ISEs), are located in the PHB chip.

There is one ISE per XIVE. ISEs are organized in the IST, with the first ISE corresponding to the first XIVE, the second ISE corresponding to the second XIVE, and so on.

In Table 3.11, “ISE Definition for LSI Interrupts Only” [34], [ ] designates optional bits and bytes. Optional bits and bytes that are not implemented must be ignored by that implementation on a *Store* and must be returned as zeros on a *Load* even when the entire field is not implemented. Implementations that do not implement the full size of the field must treat unused bits and bytes the same as optional bits and bytes. Reserved bits and bytes must be ignored on a *Store* and must be returned as zeros on a *Load*.

Table 3.11. ISE Definition for LSI Interrupts Only

Bytes	Bits	Field	Definition
[0:6]	All	Reserved	Reserved
7	[0:3]	Reserved	Reserved
7	4	Reserved	Reserved
7	5	Interrupt Rejected	Hardware sets this bit to a 1 when the interrupt that was presented to the routing layer is rejected (Interrupt Return). It sets this bit to 0 on power-on reset and on an EOI for the interrupt corresponding to this bit. The Rejected bit does not affect the internal state of the interrupt. It is used as a status bit to indicate that the interrupt was rejected at least once. Firmware can set this bit to a 0. 1: Interrupt was sent to the routing layer but was rejected at least once. 0: Interrupt might be pending or presented but not rejected.
7	6	Interrupt Presented	Hardware sets this bit to a 1 when the interrupt is presented to the routing layer. It sets this bit to a 0 on power-on reset and on an EOI or interrupt return for the interrupt corresponding to this bit. Firmware can set this bit to a 0. Setting the XIVE’s priority to 0xFF is the mechanism for disabling an interrupt.

Bytes	Bits	Field	Definition
			Interrupts are remembered if the Interrupt Pending bit is set to 1 (interrupt received) while the priority for the interrupt received is set to 0xFF. If firmware changes the priority field to non-0xFF while the Interrupt Pending bit is a 1, the remembered interrupt is presented if not previously presented (if this bit is 0). This bit is then set to 1. If the Interrupt Pending bit is deasserted (set to 0) before the priority is changed, this bit is not set to 1 and the interrupt is not presented. A state of Interrupt Presented = 0 and Interrupt Pending = 1 indicates that the interrupt request was not sent, but is waiting to be sent. This can be the result if the interrupt was presented but was returned or if the interrupt is pending but is disabled. In the disabled case, the interrupt is presented and the Interrupt Presented bit is set to 1 when the interrupt priority is changed to non-0xFF. In the interrupt return case, the Interrupt Presented bit is set to a 1 after the interrupt re-present timer elapses or when (in legacy mode) an interrupt reissue command is received for the interrupt. 1: Interrupt presented. 0: Interrupt not presented.
7	7	Interrupt Pending	For an LSI interrupt, if this bit has a value of 1, it means that the interrupt signal is active (Assert_INTx was received but not Deassert_INTx). Hardware sets this bit to a 0 on power-on reset. Firmware can set this bit to a 0. Firmware might want to correct for the hardware not presenting a Deassert_INTx (that is, in error recovery clean-up). Firmware might also set this bit to a 1 to generate interrupts for testing purposes. Hardware sets this bit to a 0 for LSI interrupts only when the Deassert_INTx message for the interrupt is received. If an EOI for the interrupt corresponding to this bit is received for an LSI interrupt and the Interrupt Pending bit is set to a 1, another interrupt is presented to the routing layer. The Interrupt Presented bit is set to a 1 in this case as well. 1: Interrupt pending. 0: Interrupt not pending.

R1-3.2.3.2-1 Hardware Requirement:

The PHB hardware must implement the IST for LSI interrupts, and the entries in this table must be as defined in Table 3.11, “ISE Definition for LSI Interrupts Only” [34].

R1-3.2.3.2-2 Firmware Requirement:

The firmware must access the IST for LSI interrupts, with 8-byte *Loads* and *Stores*, naturally aligned.

3.2.3.3. LSI Interrupt Source Number Definition

The interrupt source number is passed to the interrupt presentation layer. How the interrupt source number is generated for LSIs is implementation specific and is beyond the scope of this architecture.

3.2.4. MSI Design

This section gives the requirements for the MSI interrupt source controller that is implemented by the PHB. The term “MSI” is used to refer to “MSI” and “MSI-X”, generically. In cases where there are differences, the distinction will be made in context.

Most per-interrupt information (such as priority and server number) and state information (such as interrupt presented, interrupt queued, and interrupt rejected) is in system memory for scalability rea-

sons. The processing of the interrupt states in system memory is a joint effort between the hardware and the firmware.

The difference between 32-bit MSIs and 64-bit MSIs is how the MSI address range is decoded.

- 64-bit I/O DMA addresses. Bits 61:60 are allocated in the address to indicate, when 0b01, that the address is for an MSI.
- 32-bit I/O DMA addresses. When 32-bit MSIs are enabled, MSI addresses are determined by decoding bus address bits 31:16 as 0xFFFF (a 64 KB region, which limits 32-bit MSIs to the first 4K IVEs PHB). Chip designs can disable 32-bit MSIs by default, but must allow a way for the platform to enable this decode. This allows the platform to control the 64 KB hole in the address space, based on whether 32-bit MSI support is desired or not.

When the PE is in the DMA Stopped state, additional MSI operations that are presented from the PE after the PE enters the Stopped state are blocked by the PHB. This can happen naturally, when the hardware treats the MSI operation as a normal DMA write; that is, the DMA Stopped state is checked before an MSI decode of address is made. A DD determines that its IOA function is stopped by timing operation completions and timing out when its IOA function fails to signal completion after a reasonable period of time. MSI operations that were presented by a PE to the PHB before that PE entered the Stopped state can be presented after the PE enters the Stopped state. For example, an initial presentation was rejected by the presentation layer, and the PHB presents the same interrupt after it was rejected.

Figure 3.9, “MSI Flow” [37]shows the general flow from receipt of an MSI to the generation of the information necessary to present to the system. The details of how this information gets presented to the system is implementation dependent and is beyond the scope of this document. In this figure and section, the following terms and constructs are defined:

- IVE – An entry in the interrupt vector table. It is similar to an XIVE except that it contains state bits and exists in system memory. See also Table 3.13, “MSI IVE Definition” [42].
- IVC – Interrupt vector cache. A cache of IVEs.
- P bit – Presented. There is one P bit per interrupt source. This bit indicates that an interrupt was presented and has not yet returned an EOI. It resides in the IVE for the interrupt source, in a byte of the IVE with only the “G” bits.
- G[0:1] The Generation bits are used to control races in the software cache coherence protocol.
- Q bit – Interrupt requested/queued. There is one Q bit per interrupt source. It resides in the IVE for the interrupt source, in a byte of IVE with no other bits. This bit indicates one of two things (both require firmware to queue up a new interrupt using the firmware force interrupt (FFI) facility:
  - —Indicates that an MSI was received after the previous interrupt was presented and before the P bit is cleared.
  - —Indicates that an MSI was received when the IVE priority was 0xFF (that is, when the interrupt was disabled).

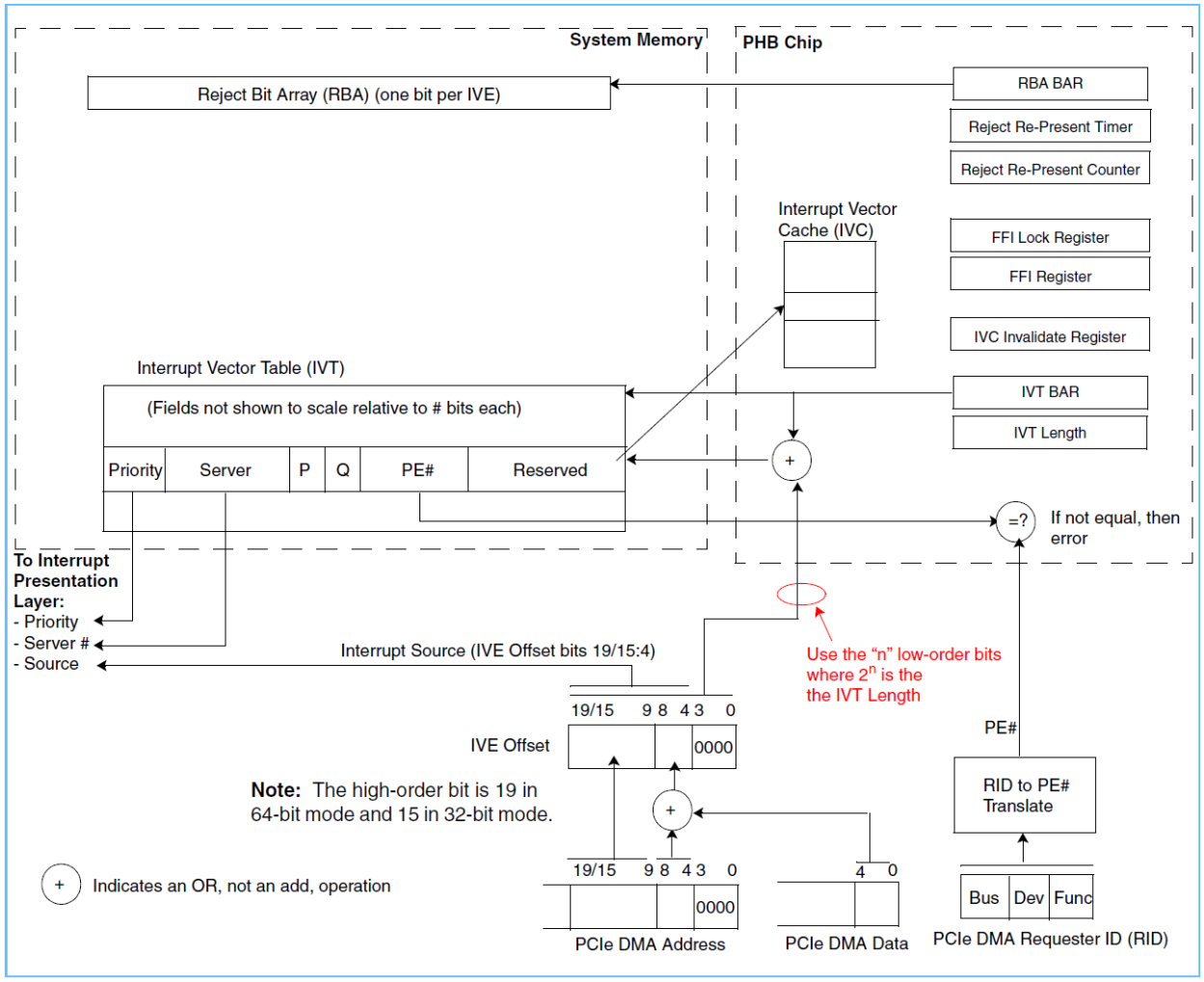
Architecture Note: The P and Q bits are placed into separate bytes for coherency management reasons because the P and Q bits are set by the hardware and cleared by the firmware.

- R bit – Rejected. There is one R bit per interrupt source. It indicates that an interrupt that was previously presented has been returned or rejected by the interrupt presentation layer. It resides

in the R bit array (RBA). Processing of the RBA is done by the PHB hardware, but the firmware might need to clean up bits in the RBA during certain clean-up procedures.

- Firmware force interrupt (FFI) facility - This facility provides firmware with a way to force an interrupt for a given source. It produces the same PHB state change. If the P bit was initially 0, it issues an interrupt to the system just as if the interrupt came from the PCIe link.

Figure 3.9. MSI Flow



Referring to Figure 3.9, "MSI Flow" [37], after the determination is made that this is an MSI operation, the PCIe address and data are combined to form the IVE index. The IVE is read from system memory or the IVC. The PE# is compared to the PE# generated from the RID (see Section 3.2.1.2, "DMA and Error Message PE# Determination, RTT, RTC Invalidate, and PELT-V" [7]). If the PE# in the IVE does not match the PE# generated from the RID, or if the IVE index causes the access to be past the end of the IVE table (as determined by the IVT Length register), the PE# is in error; the PE is placed into the error state (MMIO Stopped and DMA Stopped states). If the PE#s match, the processing continues as described in Table 3.12, "MSI State Table" [38], and as illustrated by example in Figure 3.10, "Example Interrupt State Bit Flow" [39].

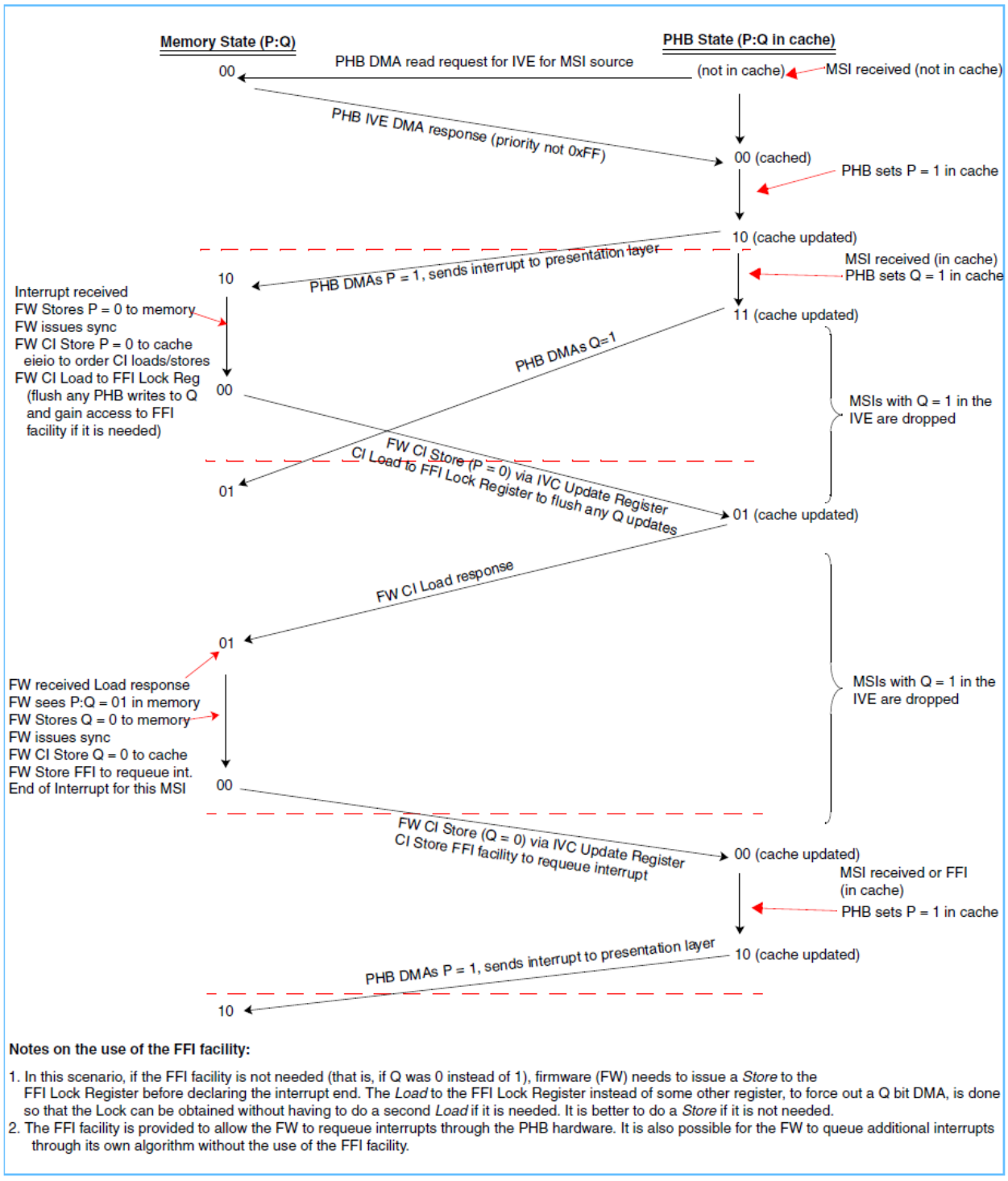
Table 3.12. MSI State Table

P:Q State	Firmware Action During EOI[a]	PHB Action[b]
00	Done with EOI.	• For IVE priority not 0xFF: On receipt of an interrupt from the PCI or from the FFI register, hardware sets P = 1[b] and sends the interrupt to the system. • For IVE priority 0xFF (interrupt disabled): On receipt of an interrupt from the PCI or from the FFI register, hardware sets Q = 1[b].
01 (Q bit processing state)	Firmware sets Q = 0[a]; that is, it transitions to 00. It then uses the FFI registers in PHB to force a hardware interrupt. Firmware first reads the FFI Lock Register until a 0 is returned, which indicates that the thread can use the facility. It then writes the FFI registers. Hardware releases the lock when the interrupt-force operation is complete.	Drop new interrupts for this source. Do not write to Q. • Hardware can reliably drop new interrupts because it knows that the firmware has seen the Q bit set. Otherwise, the P bit would not be 0.
10	Firmware sets P = 0[a]; that is, it transitions to 00. It then does a cache-inhibited (CI) *Load* to force out any PHB write to Q. It might transition to state 00 or 01, or it might transition to 10 or 11 if new interrupts come in and hardware sees P = 0 before a CI *Load*and next step is done. If P:Q is 01 after the previous action, it then goes to state 01. Otherwise, it is done with an EOI.	On receipt of an interrupt from the PCI or from the FFI register, hardware sets Q = 1[b].
11	Firmware sets P = 0[a]. It then goes to state 01.	Drop new interrupts for this source. Do not write to Q.

[a]When firmware changes an IVE in system memory, it is required to synchronize any cached version of that IVE in the IVC in the PHB by using the IVC Update Register in the PHB (see Table 3.15, “IVC Update Register Definition” [43]). Note that there exists an intrinsic race in this synchronization process. Between the time that the firmware changes an IVE and firmware can command an update of the associated IVC entry, the hardware might have fetched and updated the new IVE value due to an incoming interrupt (see Figure 3.11, “Example EOI Update Race” [40]). To assist firmware in controlling this race, the IVE contains a Generation field. Incrementing the Generation field during the EOI process allows the firmware to condition the IVC updates to cache entries with only the old value of the Generation field (see Figure 3.12, “Example EOI Update Race Controlled with Generation Number Field” [40]). This way, the IVC update does not overwrite the values for the next generation of the interrupt state. The change of the IVE in system memory is flushed out of the processor to system memory, by use of a sync instruction, before issuing the CI Store to the IVC Update Register.

[b]When the PHB updates a P or Q bit in a cached copy of an IVE, it is required to DMA that change to that IVE in system memory, thus synchronizing both copies.

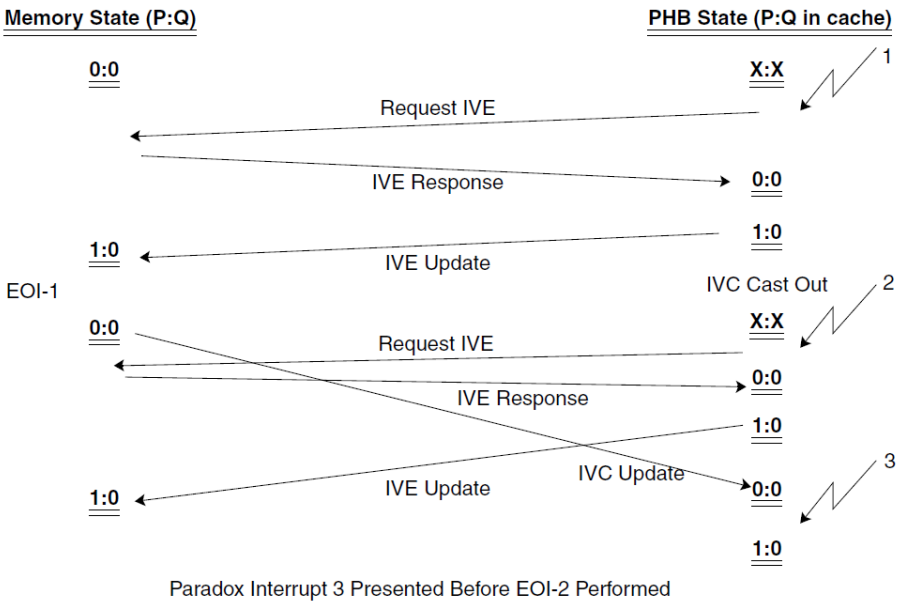
Figure 3.10. Example Interrupt State Bit Flow



Notes on the use of the FFI facility:

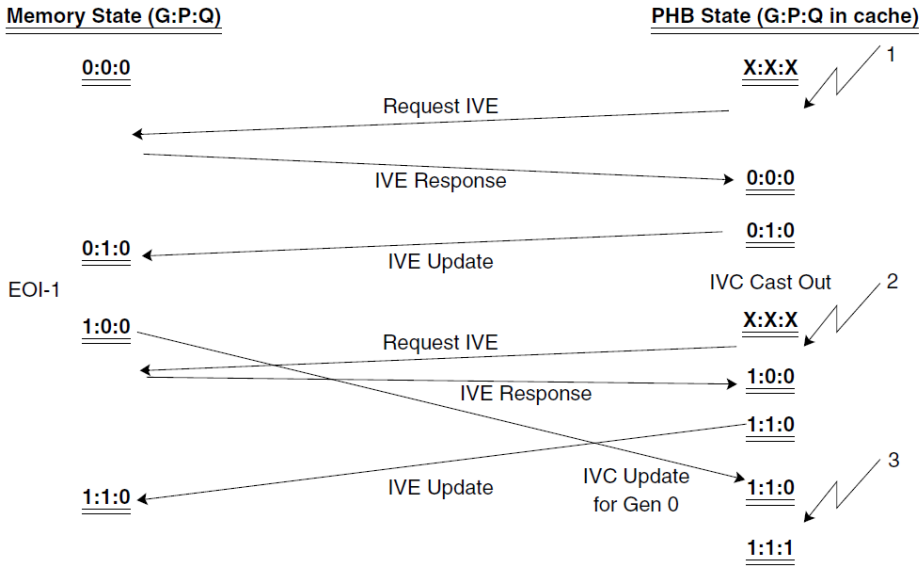
1. In this scenario, if the FFI facility is not needed (that is, if Q was 0 instead of 1), firmware (FW) needs to issue a *Store* to the FFI Lock Register before declaring the interrupt end. The *Load* to the FFI Lock Register instead of some other register, to force out a Q bit DMA, is done so that the Lock can be obtained without having to do a second *Load* if it is needed. It is better to do a *Store* if it is not needed.
2. The FFI facility is provided to allow the FW to requeue interrupts through the PHB hardware. It is also possible for the FW to queue additional interrupts through its own algorithm without the use of the FFI facility.

Figure 3.11. Example EOI Update Race



Paradox Interrupt 3 Presented Before EOI-2 Performed

Figure 3.12. Example EOI Update Race Controlled with Generation Number Field



Paradox Avoided Interrupt 3 Queued waiting EOI-2

R1-3.2.4-1 Hardware Requirement:

The PHB hardware must take all of the following actions:

a. Decode MSI operations by detecting that PCIe addresses bits 61:60 are equal to 0b01, for 64-bit I/O DMAs, or that PCIe address bits 31:16 are equal to 0xFFFF (a 64 KB region), for 32-bit I/O DMA addresses.

b. Disable 32-bit MSIs by default. Provide firmware with a way to enable them.

c. Implement the IVT with entries defined as in Table 3.13, “MSI IVE Definition” [42].

d. Implement IVT BAR and IVT Length registers, writable by firmware, whose contents points to the start and length, respectively, of the IVT in system memory.

e. Create the address to access the IVE for the MSI operation by ORing together the following entities (see also Figure 3.9, “MSI Flow” [37]):

—IVT BAR

—“n” low-order PCIe address bits, where 2^n is the IVT Length, aligned with the low-order IVT BAR bits

—PCIe data bits 4:0 aligned with PCIe address bits 8:4

Architecture Notes:

1. The IVE address generation specified in part e works for both MSI and MSI-X. It is firmware’s responsibility to set the appropriate bits to 0 in the MSI address, MSI-X address and data, and IVT BARs (see Firmware Requirement R1-3.2.4-2).

2. Due to the high-order PCIe address bit truncation in the IVE address generation specified in part e, hardware does not directly detect an address that would have accessed past the end of the IVT. However, the PE# check in part fassures that the device only accesses MSIs assigned to its PE.

f. Access the IVE with the address generated in part e of this requirement. Then compare the PE# field from the IVE with the PE# generated in Requirement R1-3.2.1.2-1. If they are not equal, put the PE into the MMIO and DMA Stopped states, as defined in Section 3.2.1.3, “PE State and EEH” [12].

g. Implement the “PHB Action” column of Table 3.12, “MSI State Table” [38].

h. On an interrupt reject (return) from the interrupt presentation layer, perform the RBA processing defined in Section 3.2.4.2, “MSI Reject (Return) Processing by PHB, the RBA, and Reject Re-Present Timer” [45].

i. Provide an IVC cache for caching IVEs used for MSI operations.

j. Provide an IVC Invalidate Register, as defined in Table 3.14, “IVC Invalidate Register Definition” [42], for invalidating IVC entries. The hardware must stop using the entry when firmware indicates the invalidate. However, it can wait until any invalidated IVE is used once. A *Store* to this register causes the specified IVE or all IVEs in a cache line to be invalidated. Issuing a *Load* to this register causes the last value *Stored* to be returned.

k. Provide an IVC Update Register, as defined in Table 3.15, “IVC Update Register Definition” [43], for updating IVC entries. The hardware must do an immediate update of the entry when firmware issues a *Store* to this register. Issuing a *Load* to this register causes the last value *Stored* to be returned.

R1-3.2.4-2 Firmware Requirement:

The firmware must take all of the following actions:

a. Set up the IVT BAR and IVT Length Registers in the PHB appropriately to point to the IVT. The IVT length must be set to a power of 2. The IVT BAR alignment must be equal to an integer multiple of the size.

b. Set up the IVE for each interrupt source, including marking those not used with a Priority of 0xFF.

c. Disable interrupt sources by setting the Priority field in the IVE, for the interrupt sources to be disabled, to 0xFF.

d. On an EOI, implement the “Firmware Action on EOI” column of Table 3.12, “MSI State Table” [38].

e. When enabling an interrupt source (that is, setting the Priority in the IVE from 0xFF to something other than 0xFF), perform the operations defined in Section 3.2.4.3, “Firmware Action on Enabling an MSI Interrupt” [46].

f. Do not allow interrupts to be shared between PEs.

g. Maintain the coherency between the system memory IVE contents and the cached IVE contents by using the IVC Update Register to update cached IVEs whenever it changes the value of an IVE in system memory. Use the IVC Invalidate Register when an IVE in system memory is no longer valid.

Firmware Implementation Note: The PHB caches the PE# along with the other IVE information in the IVC. Note that the IVC Update Register does not provide a mechanism to change the PE# of an entry. If a PE# is being changed in an IVE, it is because the interrupt source is being re-assigned. In that case, the firmware must use the IVC Invalidation Register to invalidate the entry before re-assignment.

Table 3.13. MSI IVE Definition

Bytes	Field	Definition
0:2	Server Number	The server number for this interrupt source. The number of bits implemented is dependent on the number of servers supported by the hardware. Unimplemented bits are on the left end of the field.
3	Priority	The priority for this interrupt source.
4	Bits 0:4 Reserved	Written as 0.
	Bits 5:6 Generation field	Generation number used to control races in updates of the IVC.
	Bit 7 P bit	Interrupt Presented. See Section 3.2.4, “MSI Design” [35] for the definition.
5	Q bit	Interrupt Queued. MSI received when another one has already been presented and has not EOIed yet. That is, the P bit is still a 1. The Q bit is bit 7 of the byte, and all other bits are Reserved (written as 0). See Section 3.2.4, “MSI Design” [35] for the definition.
6:7	PE#	The number of the PE that is allowed to access this interrupt source. The number of bits implemented is dependent on the number of PEs supported by the PHB hardware. Unimplemented bits are on the left end of the field.
8:15	Reserved	Reserved

Table 3.14. IVC Invalidate Register Definition

Bits	Field	Definition
0	Invalidate All	0: Invalidate the entry in the IVC specified by the Source ID field. 1: Invalidate all entries in the IVC, regardless of the value in the Source ID field.
1:15	Reserved	Reserved

Bits	Field	Definition
16:31	Source ID	The Source ID of the IVC entry to invalidate when the Invalidate All bit is set to a 0. It is possible that some implementations might not implement a full 16-bit Source ID. For those implementations, some number of the high-order bits of this field can be treated as Reserved.
32:63	Reserved	Reserved

Table 3.15. IVC Update Register Definition

Bits	Field	Definition
0	Enable P Bit Update	0: No action against the P bit in the IVC entry specified by the Source ID field. 1: Set the P bit in the IVC entry specified by the Source ID field to the value in the P bit field of this register.
1	Enable Q Bit Update	0: No action against the Q bit in the IVC entry specified by the Source ID field. 1: Set the Q bit in the IVC entry specified by the Source ID field to the value in the Q bit field of this register.
2	Enable Server Number Update	0: No action against the Server Number field in the IVC entry specified by the Source ID field. 1: Set the Server Number field in the IVC entry specified by the Source ID field to the value in the Server Number field of this register
3	Enable Priority Field Update	0: No action against the Priority field in the IVC entry specified by the Source ID field. 1: Set the Priority field in the IVC entry specified by the Source ID field to the value in the Priority field of this register.
4	Enable Generation Field Update	0: No action against the Generation field in the IVC entry specified by the Source ID Field. 1: Set the Generation field in the IVC entry specified by the Source ID field to the value in the Generation Number field of this register.
5	Condition Updates on Generation Match	0: The updates specified by bits 0 - 4 are performed without regard to the Generation field. 1: The updates specified by bits 0 - 4 are performed only if bits 6 - 7 match the value of the Generation field of the IVC entry specified by the Source ID field of this register.
6:7	Generation Number to Match	
8:23	Server Number	See Enable Server Number Update field definition.
24:31	Priority	See Enable Priority Field Update definition.
32:33	Generation Number 0:1	See Enable Generation Field Update definition
34	P Bit	See Enable P Bit Update definition
35	Q Bit	See Enable Q Bit Update definition
36:47	Reserved	Reserved
48:63	Source ID	The Source ID of the IVC entry to update, when the update field bits are set to a 1. It is possible that some implementations might not implement a full 16-bit Source ID. For those implementations, some number of the high-order bits of this field can be treated as Reserved.

3.2.4.1. Firmware Force Interrupt (FFI) and FFI Lock Registers

The Firmware Force Interrupt (FFI) facility provides firmware with a way to force an interrupt for a given source, with the same PHB state change. It issues an interrupt to the system, if the P bit was initially 0, as if the interrupt came in from the PCIe link. The FFI facilities are provided to allow firmware to requeue an interrupt through the PHB mechanisms. It is also possible for firmware to not use the FFI facilities and instead queue up additional interrupts using its own algorithms.

This facility is a shared facility across all partitions that share I/O under the PHB. The use of this facility must therefore be serialized. To do this, the FFI Lock Register is provided to facilitate this locking.

The format of the FFI, as specified in Table 3.16, “FFI Definition” [44], is set up to match the PCIe address of an MSI-X interrupt with data of 0. That is, the format is the same as an MSI-X interrupt received from the PCIe link.

Table 3.16. FFI Definition

Bits	Field	Definition
0:3	Fixed value	0b0001
4:43		0
44:59	Source Number	The 16-bit interrupt source number
60:63	Fixed value	0

Table 3.17. FFI Lock Definition

Bits	Field	Definition
0	Lock State	0: The resource lock has been acquired. 1: The resource is busy; try again. The lock is reset by the PHB hardware when the operation defined by the *Store* to the FFI register is completed. That is, the Lock state is set to 0. Normally this state is read-only, but write access is provided for firmware clean-up operations, if necessary.
1:63	Fixed value	0

R1-3.2.4.1-1 Hardware Requirement:

The PHB hardware must take all of the following actions:

a. Implement the FFI and FFI Lock registers, as defined in Table 3.16, “FFI Definition” [44] and Table 3.17, “FFI Lock Definition” [44].

b. On a *Load* to the FFI Lock register, return the value of 0 if the FFI facility is available for use. Otherwise, return a value of 1.

c. After processing a *Store* to the FFI register (that is, after forcing the requested interrupt), and when the facility is available for re-use, reset the Lock state in the FFI Lock register to 0.

d. Accept a *Store* to the FFI Lock register, setting the Lock state bit to the value specified by the *Store* data.

e. When a *Store* is received to the FFI register, force the requested interrupt into the PCIe operation flow. Use the FFI register as the DMA address with a data value of 0, just as though the MSI for that source had been received. Use the same ordering rules as for a received MSI. When that forced interrupt comes to the top of the queue for processing, process it as though it was an MSI coming in from the PCIe link.

Hardware Implementation Notes:

1. It is permissible for the hardware to ignore all data bits except the Source Number field bits, on a *Store* to the FFI register.
2. Interrupts presented through the FFI facility do not go through the PE Number checking. That is, there is no RID or RID-to-PE-Number lookup associated with the FFI operation.

R1-3.2.4.1-2 Firmware Requirement:

The firmware must take all of the following actions:

a. Before *Storing* to the FFI register, obtain a lock on that facility by *Loading* the FFI Lock register until a zero value is returned.

b. After obtaining the lock through the FFI Lock register, *Store* to the FFI register as soon as possible to prevent FFI register resource contention with other processing threads.

c. Do not write to the FFI Lock register, unless necessary for clean up.

d. *Store* to the FFI register and, if necessary, to the FFI Lock register only with 8-byte *Stores*.

3.2.4.2. MSI Reject (Return) Processing by PHB, the RBA, and Reject Re-Present Timer

The PHB is set up by the firmware to kick off processing of the RBA, when there are any bits set in the RBA, on a specific interval. The facility provided by the PHB to allow this setup is the Reject Re-Present Timer. The definition of this register is implementation specific and is beyond the scope of this document.

R1-3.2.4.2-1 Hardware Requirement:

The PHB hardware must take all of the following actions:

a. Provide a Reject Re-Present Timer that can be set by firmware.

b. Provide a Reject Re-Present Counter that is loadable by the value in the Reject Re-Present Timer.

c. Implement a BAR to point to the start of the RBA that is loadable by the firmware.

d. When the PHB receives an interrupt reject back from the interrupt source layer, update the R bit corresponding to the interrupt source in the RBA.

e. When the PHB hardware writes to any bit in the RBA, and when the Reject Re-Present Counter is 0, load the Reject Re-Present Counter with the value in the Reject Re-Present Timer. Start counting down on an implementation-dependent interval.

f. When the Reject Re-Present Counter reaches 0, if there are any bits set in the RBA, then for each R bit that equals 1; set that R bit to 0. If the interrupt is still enabled, as indicated by the Priority in the interrupt's IVE being not 0xFF, re-present the rejected interrupt to the system. Otherwise, set the Q bit in the IVE for that interrupt to 1.

g. Minimize, as much as possible, the impact of reading the RBA from system memory by reading only cache lines that might have an R bit set.

Hardware Implementation Note: Interrupts presented through the reject processing do not go through the PE Number checking. That is, there is no RID or RID-to-PE-Number lookup associated with the reject processing operation.

R1-3.2.4.2-2 Firmware Requirement:

The firmware must take all of the following actions:

a. Write a value into the Reject Re-Present Timer that provides a system trade-off of interrupt re-present latency versus system resource usage (that is, bandwidth used by the PHB hardware in fetching the RBA).

b. Set up the RBA BAR register to point to the contiguous real memory that the PHB can use to implement the RBA with a size that is a power of 2. It must be large enough to contain one bit per PE# implemented by the PHB and have an alignment equal to the size.

Table 3.18. RBA Definition

Bits	Field	Access Mode	Definition
0:(n-1)	R Bit	Hardware R/W	n = number of IVEs. Firmware only writes to these bits for clean-up reasons (for example, during hot plug operations). Firmware access requires firmware to use a read-modify-write operation with *Load*with reserve and *Store*conditional instructions. Abort the *Store*if the reservation is lost due to the PHB writing to the RBA at the same time and to same address.
n:(m-1)	Reserved		m = width of RBA. m is a power of 2 and the RBA is naturally aligned.

3.2.4.3. Firmware Action on Enabling an MSI Interrupt

Firmware sets the IVE priority for an interrupt to 0xFF to disable the interrupt at the source interrupt controller. During the time that an interrupt source is disabled in this way, if an MSI is received for that source, the PHB hardware sets the Q bit for that interrupt. It is firmware's responsibility to check and process the Q bit when the source's priority is changed from 0xFF to something else. The firmware then uses the FFI facility to force an interrupt, if necessary (that is, if the Q bit is set to a 1 at the time the priority is changed).

R1-3.2.4.3-1 Firmware Requirement:

When the firmware changes the Priority field in the ISE for an interrupt source from 0xFF to something other than 0xFF, it must take all of the following actions:

a. Write the Priority field in the IVE in system memory to the new value. Update the cached version to the new value using the IVC Update register.

b. Issue a CI *Load* to the PHB to flush any Q bit write that is in the PHB's write queue.

c. If Q = 1 in IVE for the source, set Q = 0. Read the FFI Lock register until a 1 is returned indicating that the thread can use the facility. Then, write the FFI registers; hardware releases the lock when the interrupt-force operation is complete. This forces an interrupt to that interrupt source, as in processing of the Q bit during EOI.

3.2.5. PCIe Configuration Space

Firmware needs the capability to access the PCIe configuration space while the other PEs remain in the MMIO Stopped state. Therefore, the PE for the configuration space cannot be shared by any other PE under the PHB.

R1-3.2.5-1 Hardware Requirement:

Configuration access to an IOA function and to the I/O fabric must be available at all times to the firmware, even though the MMIO Stopped state for a PE might be set. That is, the hardware must provide PEs for the configuration space that are separate from the normal MMIO memory space PEs.

3.2.6. PE State Table

The PHB is required to capture certain data relative to PE errors. The PE state table (PEST) is in system memory, and the PE state entry (PESE) is defined by Table 3.19, “PESE Definition” [47].

Table 3.19. PESE Definition

Bits	Field	Definition
0:1	Reserved	
2	MMIO Cause	This bit is set to ‘1’ if an MMIO operation froze the PE. It is set to zero if a DMA operation froze the PE. If both an MMIO and DMA operation attempt to freeze the endpoint in the same cycle, the MMIO operation has priority.
3	CFG Read	The operation that caused the PE to be frozen was a PAPR inject CFG Read.
4	CFG Write	The operation that caused the PE to be frozen was a PAPR inject CFG Write, or a CFG Write with Size or Access error.
5:7	Transaction Type (0:2)	This is an encoding of the transaction type that caused this PE to be frozen. The encoding is as follows: 000 DMA Write 001 MSI Interrupt 010 DMA Read 011 DMA Read Response 100 MMIO *Load* 101 MMIO *Store* 110 Unused 111 Other (internal checker detected error). This encoding is used for most fatal internal error cases. Examples include data parity errors, state machine errors, and so on This encoding is also used where there was no error indicated when a request covered by one of the other encodings was first received, but after the request was issued, some exception condition occurs. For example: a response timeout.
8	CA Return Status or Completion Timeout	An MMIO *Load*, MMIO I/O Write, or other transaction returned from the PCIe link with a status of Completer Abort (CA), or the MMIO operation terminated with a completion timeout.
9	UR Return Status	An MMIO *Load*, MMIO I/O Write, or other transaction returned from the PCIe link with a status of Unsupported Request (UR).
10	NONFATAL_ERROR	A PCIe nonfatal error occurred.
11	FATAL_ERROR	A PCIe fatal error occurred.
12	Reserved	Reserved
13	Parity/ECC UE Error	Any parity error or uncorrectable ECC error.
14	Correctable Error / CORR_ERROR	A correctable error occurred.
15	PCIe Core Interrupt	An error occurred in the PCIe core.
16	Invalid MMIO Address Translation / IODA2 Error	The down-bound MMIO did not match against any BARs or was invalid, or the up-bound DMA request had an error defined by this architecture.
17	Reserved	Reserved
18	TCE Page Fault	A DMA transaction accessed a TCE whose page-access control bits were all zeros.
19	TCE Access Fault	A DMA transaction conflicted with its allowed permissions according to the TCE page-access control bits (includes all cases including page fault).
20	DMA Response Timeout	A timeout occurred while waiting for an outstanding DMA Read Response to return from system memory.
21	AIB Size Invalid	The Size field in an incoming AIB packet was not valid.
22:25	Reserved	Reserved
26:31	LEM Bit Number (0:5)	Bit number in the LEM FIR Accumulator Register for the error that froze this endpoint.

Bits	Field	Definition
32:47	Requester ID (0:15)	This is the PCIe requester ID value in the TLP. PCI Bus (0:7) - Requester ID (0:7) PCI Dev (0:4) - Requester ID (8:12) PCI Func (0:3) - Requester ID (13:15)
48:63	MSI Data (0:15)	Bytes 0 and 1 for an MSI interrupt. **Note:** Only bits 4:0 of byte 0 are used for MSI interrupts in the design. For a PCIe Tag Reuse error, the bits 48:55 contain the PCIe Tag value that was detected as reused.
64:66	Reserved	Reserved
67:127	Fail Address(3:63)	This is the address that was associated with the transaction that froze the endpoint. For MMIOs, the address used is the 48-bit AIB address, right justified. MMIO Fail Address (03:15) = all zeros MMIO Fail Address (16:63) = AIB Address (0:47) For DMAs, the address used is the least significant 61 bits of the PCI address. DMA Fail Address (03:63) = PCI Address (60:0) **Note:** This field might be invalid or all zeros for certain cases like MMIO/DMA response related errors where the address of the original transaction is no longer known or stored. The address is generally valid for all errors detected during the request phase of a transaction.

R1-3.2.6-1 Hardware Requirement:

The PHB hardware must take all of the following actions:

a. Implement the PEST in system memory, with entries defined by Table 3.19, “PESE Definition” [47].

b. Implement a BAR, to point to the start of the PEST (PEST BAR), that is loadable by the firmware.

c. Use the PE#, with four trailing zeros concatenated, to index into the PEST.

d. When a PE is placed into the MMIO Stopped state, write the appropriate error information into the PESE for that PE#.

Implementation Note: Some bits might not make sense for some implementations. However, if possible, they should be implemented and when implemented, appear in the location designated by Table 3.19, “PESE Definition” [47].

R1-3.2.6-2 Firmware Requirement:

The platform firmware must take all of the following actions:

a. Set up the PEST BAR to point to the start of the PEST in contiguous real system memory, with a size that is a power of 2 and with an address alignment on an integer multiple of the size of the table.

b. Clear the contents of a PELE corresponding to a PE# before clearing the MMIO Stopped state for that PE.

Appendix A. Endpoint Partitioning

This appendix describes details of endpoint partitioning in a PCI Express (PCIe) fabric environment. For implementation information, see Section 3.2, “Lower-Level Details” [6].

A.1. Endpoint Partitioning Overview

Logical partitioning (LPAR) is the capability to divide the resources of a computer system among different partitions, which then act independently. In this environment, it is not permissible for resources or programs in one partition to affect another partition’s operations.

To be useful, the granularity of assignment of resources needs to be fine-grained. For example, it is not considered acceptable to assign all resources under a PCI host bridge (PHB)[1] to the same partition. That approach would restrict configurability of the system, including the capability to dynamically move resources between partitions. To be able to partition I/O adapters (IOAs) requires some functionality in the bridges in the system be able to partition the IOAs or individual functions of an IOA to separate partitions. At the same time, one partitionable resource must be prevented from affecting another partition or getting access to another partition’s resources. For example, the following actions must be prevented:

- Addressing the resources directly
- Causing an error that affects other partitions
- Causing false interrupts to another partition in an attempt to cause a denial-of-service attack.

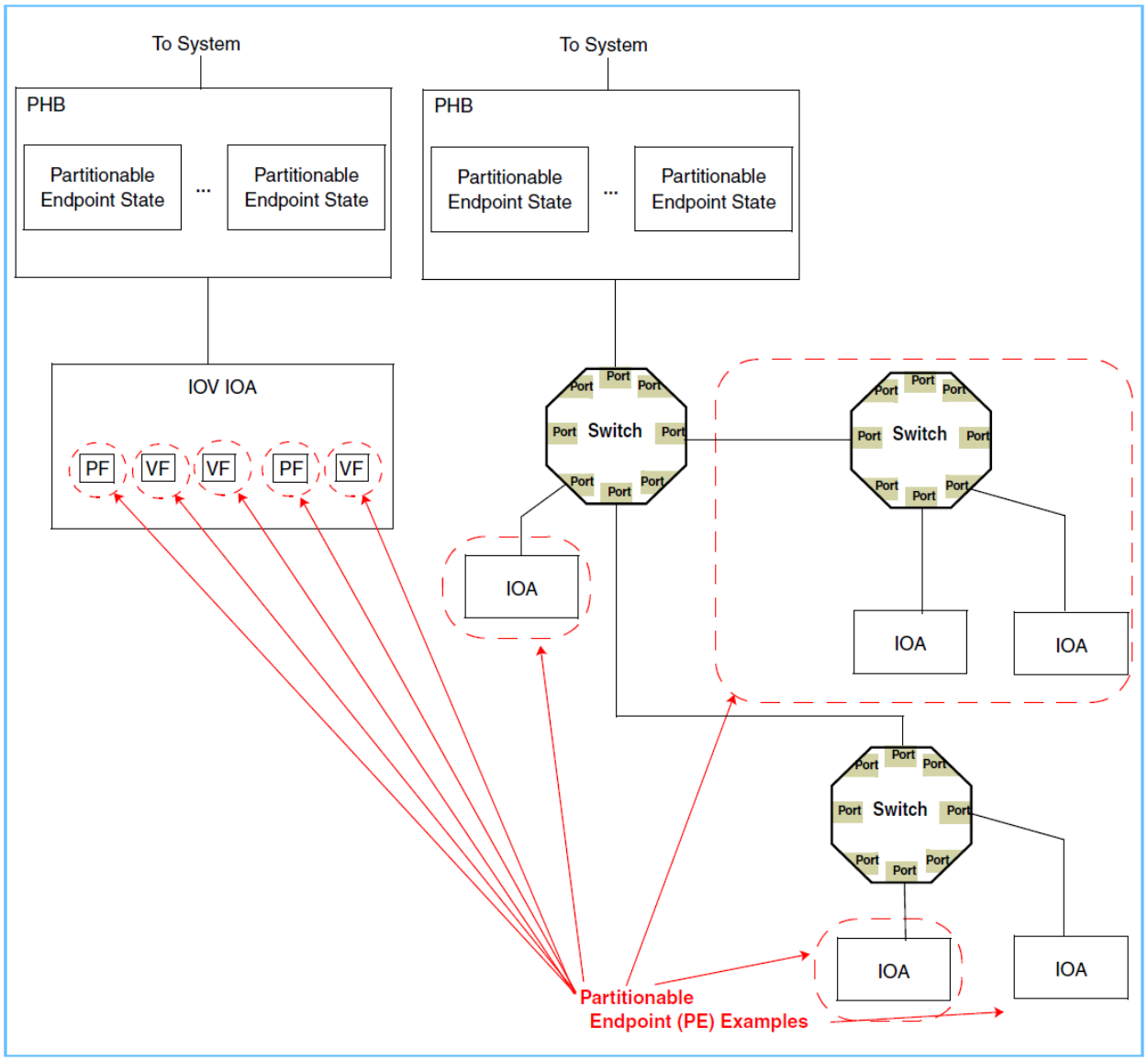
A partitionable endpoint (PE) is a separately assignable I/O unit.[2] That is, a PE is any part of an I/O subsystem that can be assigned to a partition independent of another PE. In Figure A.1, “Example System Configurations: Partitionable Endpoint (PE) Definition” [50], examples of PEs are shown encircled with dotted boxes.

There are also aspects of shared state. Examples include any element that is shared between partitions, such as PHBs and switches, and that detects an error that cannot be isolated to a specific PE. In such cases, the error must be propagated to the state of all PEs that share that element.

[1] PCIe “Root Complex” terminology correlates directly to the “PHB” terminology in this architecture (that is, the reader can substitute the “Root Complex” PCIe terminology, if desired, for “PHB” when the text can relate to PCIe.

[2] PCIe defines an “endpoint” somewhat differently than this architecture defines a “Partitionable Endpoint,” in some cases. PCIe defines an endpoint as “a device with a Type 0x00 Configuration Space header.” That means any entity with a unique Bus/Dev/Func # can be an endpoint. In some cases, a PE does not exactly correspond to this unit.

Figure A.1. Example System Configurations: Partitionable Endpoint (PE) Definition



A.2. Endpoint Partitioning Functional Specifics

Several functions in the PHB are partitioned per PE, and might have to keep state and control separate on a per PE basis:

1. IOA address domains (see Section A.2.2, “Address Domains” [52])

   a. MMIO *Load/Store* address domains (see Section A.2.2.1, “MMIO Load/Store Address Domains (Not Configuration)” [52])

   b. Configuration Space Address Domains (see Section A.2.2.2, “Configuration Space Address Domains” [52])

   c. DMA I/O bus address domains and TCEs (see Section A.2.2.3, “DMA I/O Bus Address Domains and TCEs” [53])

2. IOA Error domains (specifically for EEH) (see Section A.2.3, “IOA Error Domains (Specifically for EEH)” [53])
3. IOA error injection domains (see Section A.2.4, “IOA Error Injection Domains” [54])
4. Interrupts (see Section A.2.5, “Interrupts” [54])
   a. LSI (see Section A.2.5.2, “LSI Information” [55])
   b. MSI (see Section A.2.5.3, “MSI Information” [55])
5. PE Reset domains (see Section A.2.6, “PE Reset Domains” [56])
6. PE Hot Plug and Power domains (see Section A.2.7, “PE Hot Plug and Power Domains” [56])

Some of these functions are partitioned by the PHB, making them unique in some aspects to this architecture. Other functions happen naturally within the definition of the industry standard PCIe architecture. The following sections deal mainly with the former, but touch on the latter.

In addition, as indicated in the following sections, some of the following items can be optional based on the platform needs.

A.2.1. PE Domains

A.2.1.1. General

The PE domain encompasses all the individual domains necessary to hold the state and control information for the PE. The breakdown of the domain into the individual domain components is described in the following sections.

A.2.1.2. Numbering

The PE Domain Number is the bond that associates the various domain components to the same PE. PCI defines several divisions that can differentiate IOAs:

- Bus # (or simply “Bus”). The highest level of division. Each bus or PCIe link under a PHB has a unique Bus #.
- Device # (or simply “Dev”) within the Bus #. Subdivides the IOAs on a bus (the next level of division). For PCIe devices that implement the optional PCIe alternate RID interpretation (ARI), the Dev and Func fields can be combined into one 8-bit Func field.
- Function # (or simply “Func”) within the Device #. Subdivides the IOA into functions. Multifunction IOAs have multiple function numbers, and single function IOAs have only one.

The Bus/Dev/Func is combined into one field for purposes of naming. This field can be called the Routing ID (RID). The RID can take the form of a Requester ID or Completer ID, depending on the context in which it is used (ID of requestor or ID of completer for a request).

The architecture defined here allows for division down to the lowest level for domains (Bus/Dev/Func). This architecture also allows for the granularity to be higher, for implementations that are to

be used in platforms that do not need such fine a granularity or for configurations where the PCI architecture does not allow for such level of granularity (for examples, see Figure A.1, “Example System Configurations: Partitionable Endpoint (PE) Definition” [50]). In general the larger the granularity, the less flexibility in the configuration, the less hardware implementation costs, but potentially higher software implementation costs to deal with the shared aspects. Therefore, there is a trade-off to be made in choosing the granularity for a particular implementation. Some platforms might require a certain flexibility of assignment. Chip designers need to be aware of the requirements of those platforms and the ramification of their implementation choices.

A limitation of the Bus/Dev/Func numbers is that the number space is sparse. Bus/Dev/Func allows for 64K domains (Bus - 8 bits; Dev - 5 bits; Func - 3 bits). However, most PHB implementations will probably only implement fewer than 1K domains for the foreseeable future. Thus, to number the domains by the PCI numbering schemes might be problematic in the implementation. Therefore, this architecture defines a way to correlate an incoming PCI transaction’s Bus/Dev/Func with an internal domain number (PE#), which is compact.

A.2.2. Address Domains

Separation of addressing between PEs is important to keep one partition’s I/O from affecting another partition and to keep from one partition from accessing another partition’s I/O.

A.2.2.1. MMIO Load/Store Address Domains (Not Configuration)

MMIO (I/O and memory space) addresses are typically decoded, as they pass down through the PCI tree, by switches and bridges. As the PE state is moved up in the PCI tree, some decoding must also be moved up. This is necessary to determine which PE state to affect when an MMIO error occurs. It is also required to affect certain functionality, such as relaxed ordering and transaction classes, based on which IOA is being addressed by the MMIO.

Creation of the MMIO address domain is done by decoding the individual PE addresses and associating that decode with a PE. This can be done in several ways. See Section 3.2.1.1, “MMIO PE# Determination” [7].

Thus, each PE MMIO address range that is decoded is associated with a compact PE#. This PE# is tied to the Bus/Dev/Func number of the IOA, as described in Section A.2.2.3, “DMA I/O Bus Address Domains and TCEs” [53].

Besides a compact PE#, the following pieces of state can also be associated with the MMIO decode. They can be tied back to the PE# through indirect association or can be implemented in the same structure as the MMIO decoding.

- EEH state (See Section A.2.3, “IOA Error Domains (Specifically for EEH)” [53]for more information.)
- Error injection domains. (See Section A.2.4, “IOA Error Injection Domains” [54].)
- I/O ordering considerations

A.2.2.2. Configuration Space Address Domains

For PCIe PHBs, the hardware must not disable configuration accesses when the partitionable endpoint goes into the MMIO Stopped state. It must provide a separate PE domain for the configuration space that is used when firmware accesses the configuration space.

A.2.2.3. DMA I/O Bus Address Domains and TCEs

The platform must provide a way for an IOA to get access to all the physical memory that it needs to service the partitions that it needs to service. However, an IOA must not get access to any partitions' memory that it is not supposed to access. Physical memory for different partitions is interspersed throughout the physical memory address range. Given the requirement to prevent access to other partition's memory, it is not realistic, in general, for an IOA to get access directly from the I/O bus address to the physical memory address. This is accomplished by the translation control entry (TCE) mechanism in this architecture. This TCE mechanism also provides an indirection mechanism that allows the hypervisor to hide the physical system memory address from the partitions and, specifically, from the DDs.

This architecture provides structures that are defined to limit an IOA to a range of bus addresses (the DMA I/O bus address domain of the IOA), while using industry-standard bridges. The mechanism to do this is the translation validation table (TVT).

The TVT is a table of entries (translation validation entries, or TVEs), each of which is assigned to a single PE. The index into the TVT is by the PE#, which is looked up in the RID translation table (RTT), and by one or more address bits from the I/O bus address that is generated by the IOA. This is called the TVE Index.

For a definition of the TVE fields, see Table 3.5, "TVE Definition" [23].

Figure 3.5, "I/O Address Validation and TCE Translation Implementation for 64-Bit DMA Addresses" [21] shows the operation of the TVT, including the lookup of the TCE for 64-bit I/O addresses. The picture is similar for 32-bit I/O addresses except that the I/O bus address bits that make up the TVE Index are implied to be 0. Therefore, only one TVE is available for each PE for addresses smaller than 4 GB.

For more information about how this all works, see Section 3.2.1, "PE# Determination, PE State, EEH, and Error Injection" [6] and Section 3.2.2, "DMA Design,TVEs, and TCEs" [15].

A.2.3. IOA Error Domains (Specifically for EEH)

Enhanced error handling (EEH) is a powerful technology developed by IBM to prevent I/O errors from propagating to the system and causing unrecoverable errors, which generally bring down the operating system. EEH is a required technology for LPARed systems, so that an error in the I/O subsystem of one partition does not affect the other LPAR partitions.

EEH in the PHB stops operations to and from a PE when an error is detected (called the Stopped state). Keys to this function are:

1. The PE must be prevented from completing the I/O operation in error:
   - In such a way that the PE does not propagate an error to any partition
   - In such a way that the requester of the I/O operation does not use bad data
2. The stop of operations must appear to a DD to be isolated to just that DD. This implies extra hardware or firmware to support the continuation of I/O operation of other PEs in the face of an error generated from another PE.

Exceptions:

- A plug-in adapter that has multiple IOAs on it under a PCI-to-PCI bridge, and for which there exist multiple DDs (potentially one per function). In this case, the DDs for those multiple devices or multiple functions must coordinate any Stopped state recovery. These cooperating DDs do not necessarily need to be in the same partition, as long as:
  - One partition owns the responsibility for coordinating error recovery, and the cooperating DDs have a communication path between the partitions.
  - The user of such shared IOAs understands that one partition can affect a sharing partition's performance by a denial-of-service type attack through causing of EEH Stopped states and the ensuing EEH recovery of operations.
- An IOA that has multiple functions on it, and for which there exist multiple DDs (potentially one per function) and for which the platform does not provide PE functionality down to the Func # level (only the Bus # level). The same shared restrictions and conditions that are listed for a plug-in adapter with multiple IOAs apply here.

3. Software (DD or above) for one PE must not be able to introduce an error that can cause another PE to enter the Stopped state.
   - Software might, for example, improperly set up the TCEs for an I/O operation, or pass the wrong address to its IOA, causing an access to a TCE that is invalid (TCE not set up, or TCE set to read-only for a write, or PCI Atomic operation or write-only for a read or PCI Atomic operation). This would cause the assertion of the Stopped state.
   - It is acceptable for a platform hardware error to affect multiple PEs, as long as the recovery from it is transparent to the DD. (That is, the platform makes it appear to all DDs and PEs that they have encountered the error condition themselves.) Examples might include:
     - An error in a switch or bridge between the PE and the PHB might cause multiple, or all, PE domains in the PHB to enter the Stopped state. The key here is that there cannot be any of these conditions that can be overtly caused by a partition's software (for example, by a DD). If the hardware cannot determine the source of the error, it must put all PEs under the PHB into the Stopped state.
     - When the firmware requires temporarily suspending all operations under a PHB to recover a PHB or I/O fabric error, the firmware can place all the PE domains in a PHB into the Stopped state. This makes it look to the DD and operating system that the error is just for its IOA.
4. The capturing of fault information for problem determination must be allowed after the Stopped state condition occurs.
5. Firmware must have access to the configuration space below the PHB when any or all of the PEs are in the Stopped state.

A.2.4. IOA Error Injection Domains

Hardware, firmware, DD code, and operating system code development for PE functionality requires the capability during development to be able to programmatically inject errors, to test the hardware, firmware, and software. This functionality, although it does not have to be partitioned to the PE level, at least needs to be able to inject errors that cause a specific PE to see the error injected.

A.2.5. Interrupts

A.2.5.1. Types of Interrupts

Two types of interrupts are supported for PEs:

1. Level-Signalled Interrupt (LSI). This type of interrupt was defined by the original PCI architecture. The IOA activates an LSI interrupt and does not deactivate the interrupt until told to do so by the DD. The DD must tell the IOA to release the LSI before issuing an EOI to the interrupt controller. It must do so in a way that guarantees that the request to release the LSI gets to the IOA and gets signalled to the interrupt controller before the EOI gets to the interrupt controller. Otherwise, the interrupt controller presents the same interrupt again on receiving the EOI. The IOA can try to activate the same interrupt signal for a different operation while it remains activated for a previous interrupt. Therefore, the interrupt processing must assure that all outstanding interrupts have been processed after telling the IOA to release the interrupt.

   Originally, the LSI was signalled by separate signal wires that were wired to the interrupt controller. For PCIe, a message to turn on an LSI and turn off an LSI is packetized across the PCIe bus. PCIe limits the number of these messages per PHB to a total of four per root complex (that is, per PHB). That means that there can be at most four PEs below a PHB that support LSI interrupts because of the interrupt sharing requirement (see R1-3.2.3.1-2 c [34].

2. Message-Signalled Interrupt (MSI). The IOA signals this interrupt by writing data that contains interrupt information to a specific address that can be decoded by the system to be that of an interrupt controller. The interrupt is signalled once per occurrence. It does not have to be “released” by the DD before an EOI is issued to the interrupt controller. This is what is sometimes called an “edge triggered” interrupt. As with LSIs, the IOA can try to activate the same interrupt signal for a different operation before finishing processing of that same interrupt source for the previous operation. The timing requirements are a little different for the MSI case, however. In this case, the DD must assure that, after issuing an EOI to the interrupt controller, the IOA does not have any outstanding interrupts pending.

   This type of interrupt was first defined by later versions of PCI, and was made required by PCI-X. PCIe, a packet-based protocol, takes this further and tries to deprecate the LSI method (leaving the MSI method), by strictly limiting the LSI number of interrupts (see Section A.2.5.2, “LSI Information” [55]).

A.2.5.2. LSI Information

For PCIe, two messages are defined, Assert_INTx and Deassert_INTx, for emulation of PCI INTx signaling, where x is A, B, C, and D for respective PCI interrupt signals. These messages are used to provide “virtual wires” for signaling interrupts across a link. Because switches collect these virtual wires and present a combined set at the switch’s upstream port, there can be only four total LSI interrupts under one PCIe PHB.

LSIs are not validated by the PHB. Therefore, it is firmware’s responsibility to make sure that a partition does not accidentally or intentionally set up the configuration space of an IOA owned by it to access an LSI for another partition.

A.2.5.3. MSI Information

MSI accesses are validated by comparing the PE# of the device accessing the IVE to the PE# in the IVE. Therefore, the IOA cannot be set up accidentally or intentionally to access the MSI belonging to another PE. See also Section 3.2.4, “MSI Design” [35].

A.2.6. PE Reset Domains

For PE error recovery, the PCIe hot reset capability might not be sufficient. Therefore, the PCIe fundamental reset capability might need to be provided for each PE. This implies that a hot plug controller must be provided for each PE, even if the PE is not pluggable.

For IOA functions that implement the optional PCI function level reset (FLR), that is also available to reset down to the function level. It is required when a PE is a single function.

A.2.7. PE Hot Plug and Power Domains

For PEs that are hot pluggable, the hot plug controller and all external bus isolation and power-control electronics must be provided. For PEs that participate in DLPAR but that are not hot pluggable and do not implement FLR, the hot plug controller and any external power control electronics must be provided to power cycle the PE to get it into a known initialized state.

Appendix B. No-Translate Operation

Table B.1, “IODA2 No-Translate Operation” [57]shows the no-translate operation for the I/O Design Architecture, version 2 (IODA2). IODA2 has 2 or n segments that are mappable for each PE (bi-modal). For example, POWER8 implements n = 64 and has 2 or 64 segments that are mappable for each PE (bi-modal). IODA2 also has separate address spaces for each PE.

Table B.1. IODA2 No-Translate Operation

Function	IODA2	Comments
TVE selection • 32-bit • 64-bit	PE# is obtained from the RTT lookup. This is concatenated with either PCI addr 59 or 59:m, depending on PHB configuration bit setting, to form the index into the TVE. For 32-bit, these high-order addr bits are implied to be 0.	For IODA2 addresses below 4 GB, there is only one TVE for any given RID. For example, POWER8 implements “m” = 55, so 1 bit or 5 bits (59:55) are used.
TVE selection validation	Not required because only the RID assigned to a TVE can access the TVE, because the PE# for the RID is used as part of the index to select the TVE.	
TVE validation of address	Uses the address range for the no-translate field of the TVE to validate the address.	
No-translate address capability	A PHB can support a mixture of translated and untranslated device addresses. One TVE is used per contiguous address range, with the number of ranges per RID only limited by the number of TVEs available. The address in the range is specified by the TVE field “Addr range for no translate” with a starting boundary that is 4 KB aligned and a size that is an integer multiple of 4 KB. For IODA2, only those TVEs selectable by the number of address bits concatenated with the PE# are available. Architecturally, any number of address bits can be used, giving access to all the TVEs. However, due to system requirements to support more than one PE# under a PHB for most applications with no-translate, the implementations limit the number of address bits that get concatenated with the PE# for TVE selection. Architecturally, the selection of 1 bit or “n” bits is bi-modal per PHB, meaning one or the other for any particular PHB.	For example, for POWER8, the selection uses 1 bit or 5 bits (n = 5) and is bi-modal per PHB. PHB uses either 1 or 5 bits, configurable by a configuration bit. For POWER8 this gives: • Two TVEs per PE with 256 PEs available • Thirty-two TVEs per PE with 16 PEs available Architecturally, the PAPR interfaces only allow no-translate mode for addresses above 4 GB, with addresses below 4 GB being allocated for address translation enabled. See Figure B.2, “Example Physical Address Map with TCE Bypass Enabled for Some PEs” [59].

For IODA2, the TVE no-translate field adds 2 high-order bits from the TCE Table Size field (TVE[byte6]) to extend the addressing to 50 bits.

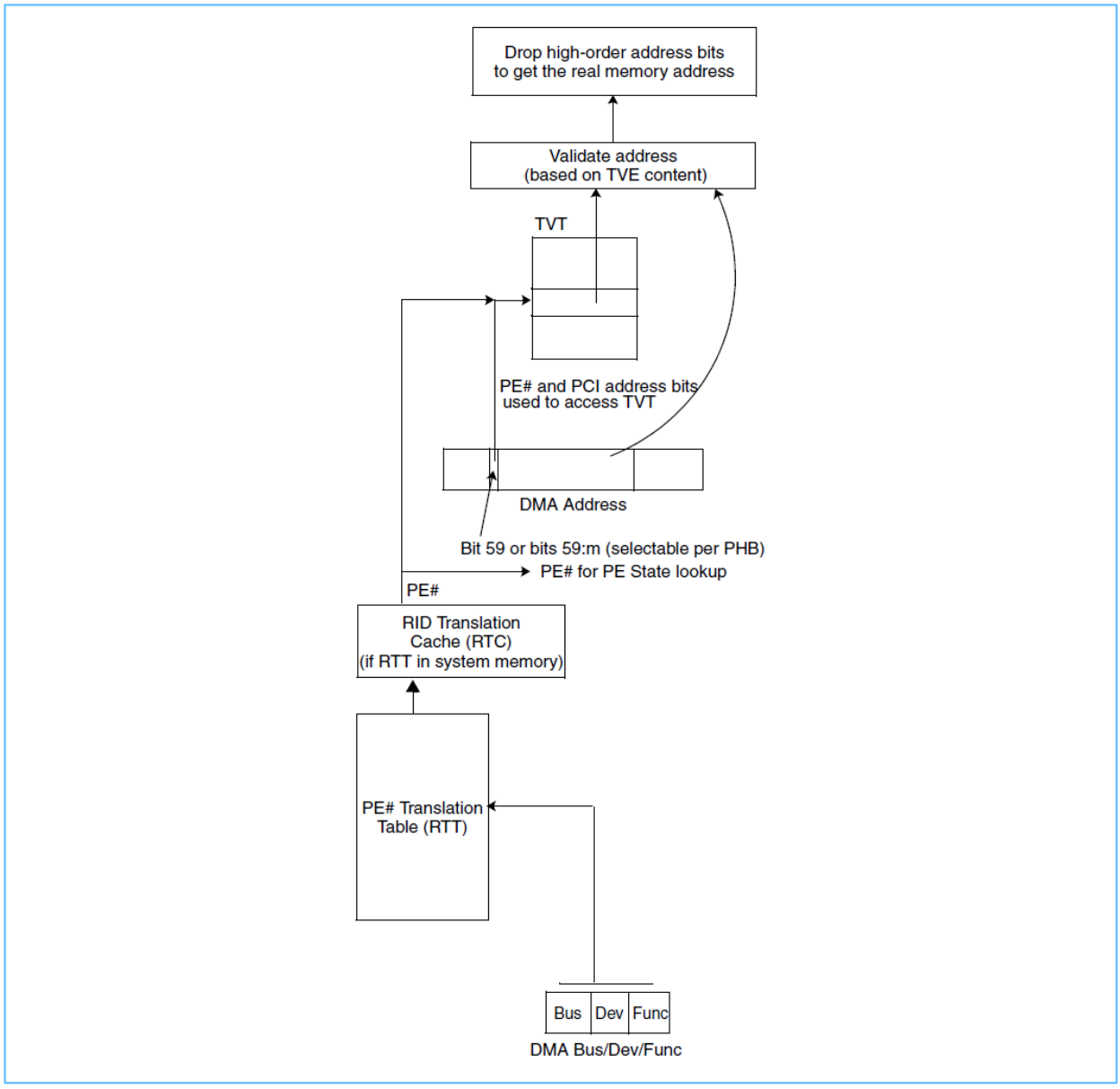
When the I/O Page Size field is zero (no translate case) and the TVE is valid (TVE[byte 6 bit 3] = 1), then if
The PCI Express address[bits 49:24] ≥ (TVE[byte 6 bits 4:5] concatenated with TVE[bytes 0:2]) and
The PCI Express address[bits 49:24] < (TVE[byte 6 bits 6:7] concatenated with TVE[bytes 3:5])
then use the low-order PCI Express address[bits 49:0] untranslated, as the DMA address.

Note

1. The no-translate case is not valid for 32-bit PCI Express addresses.
2. The alignment of the no-translate address range in real address space is 16 MB or larger.

Figure B.1, “IODA2 TVE and PE# Determination” [58]shows graphically the IODA2 TVE and PE# determination.

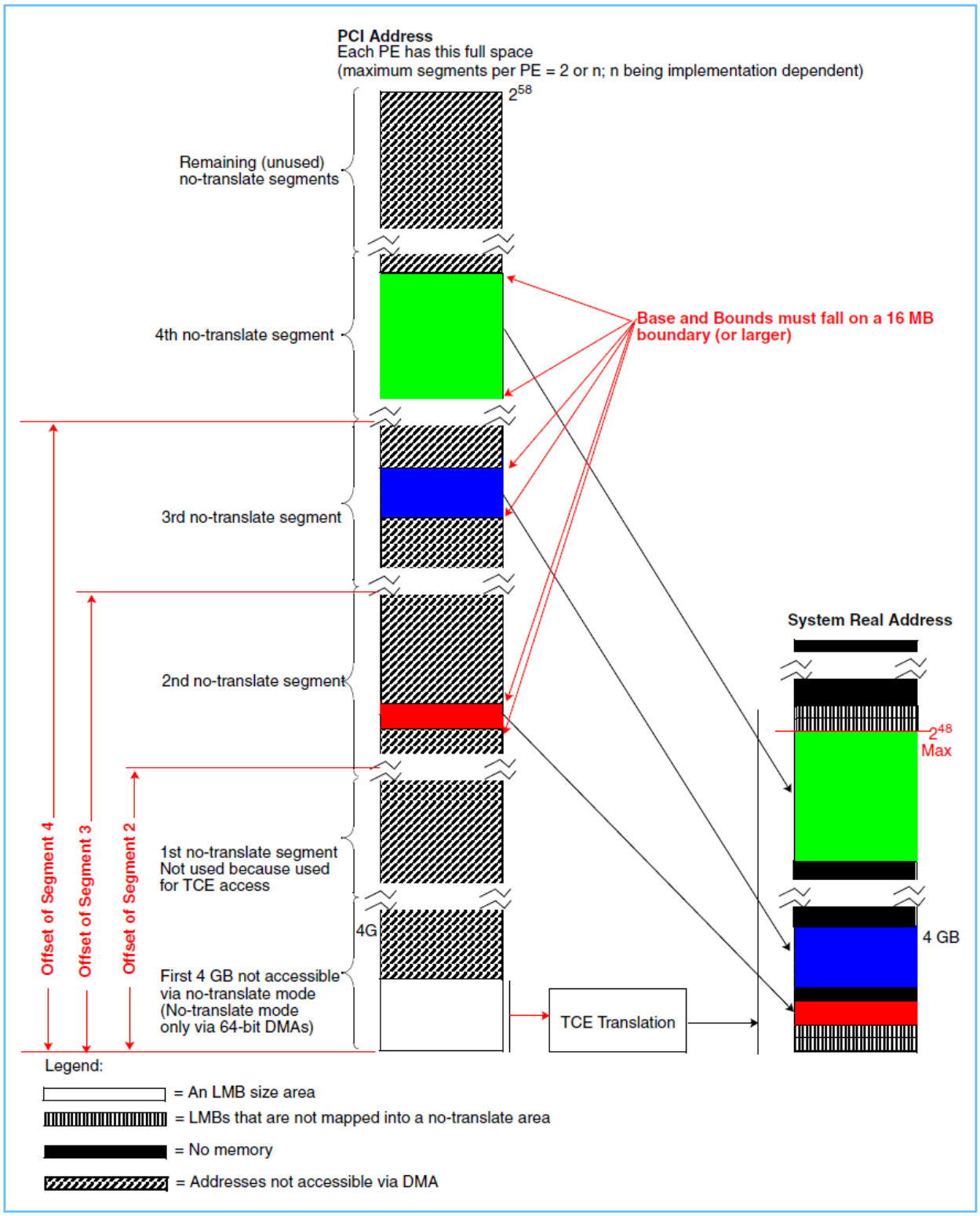
Figure B.1. IODA2 TVE and PE# Determination



B.1. No-Translate Example

An overview of how no-translate maps logical memory blocks (LMBs) is shown in Figure B.2, “Example Physical Address Map with TCE Bypass Enabled for Some PEs” [59].

Figure B.2. Example Physical Address Map with TCE Bypass Enabled for Some PEs



Appendix C. Glossary

AIB	ASIC interconnect bus.
ARI	Alternate RID interpretation.
BAR	Base Address Register.
Bus/Dev/Func	Bus, device, and function. These are three fields in the PCI/PCI-X/PCIe domain that define an IOA function. They come from the Bus (8 bits), Device (5 bits), and Function (3 bits) fields that define the configuration address for the IOA function. In addition, with the implementation of the PCIe alternate requester ID interpretation (ARI) option, the Dev and Func fields can be combined into one 8-bit Func field and a device can consume multiple buses. See also RID and CID.
CFG	Configuration.
CI	Cache-inhibited.
CID	Completer ID. When returning the completion for a transaction, the completer attaches its Bus/Dev/Func to the transaction as a CID. See also RID and BUS/DEV/FUNC.
DD	Device driver. Software that interfaces to and controls an IOA.
DMA	Direct memory access.
DMA Stopped state	See Stopped State.
DLPAR	Dynamic logical partitioning.
DR	Dynamic reconfiguration. The capability of a system to adapt to changes in the hardware/firmware physical or logical configuration, and to be able to use the new configuration, all without having to turn the platform power off or restart the operating system. See the PAPR document for more information.
ECC	Error correction code.
ECRC	End-to-end cyclic redundancy check.
EEH	Enhanced I/O error handling option. See Section 2.2.4, “EEH” [4].
EEH Stopped state	See Stopped State.
Endpoint partitioning	The concept of having a collection of independent domains (addressing, error state, and so on) that relate to a single IOA (that is, a single endpoint). See Appendix A, Endpoint Partitioning [49]. See also PE.
EOI	End of interrupt.
FFI	Firmware force interrupt.
FIR	Fault Isolation Register.
FLR	Function level reset.
FMTC	Firmware-managed TCE coherency.
FW	Firmware.
HB	Host bridge. An entity that attaches an I/O bus to a system. A PHB is a specific HB for a PCI bus. See also PHB.
IOA	I/O adapter (for example, a PCI adapter). These adapters can be built-in (for example, soldered onto a system planar) or plug-in (for example, pluggable into a PCI slot). An IOA can be single function or multiple function.
IOA function	A single function, or specific function, of an IOA.
IODA2	I/O Design Architecture, version 2.
IOV	I/O virtualization. For more information, see the *PCI-SIG I/O Virtualization (IOV) Specifications*.
ISE	Interrupt state entry.
IST	Interrupt state table.
IVC	Interrupt vector cache.
IVE	Interrupt vector entry. For a particular interrupt source, contains the interrupt priority, server number, presented bit, and queued interrupt bit. See Table 3.13, “MSI IVE Definition” [42].
IVT	Interrupt vector table. A table of IVEs.
LEM	Local error macro.
LMB	Logical memory block.

LPAR	Logical partitioning. See the PAPR document for details.
LSI	Level-signalled interrupt. An interrupt signaled by a packet on the PCIe bus.
MMIO	Memory-mapped I/O. Refers to mapping the I/O bus address space (for example, I/O bus memory and I/O address spaces) into the *Load/Store* address space of the processor.
MMIO Stopped state	See Stopped State.
Migration descriptor	A migration descriptor is created by the hypervisor during a memory migrate operation to allow for writing DMA data to two real pages per TCE.
MSI	Message signalled interrupt. An interrupt that is signaled by a write to a particular address with specific data.
MSI-X	Message signalled interrupt - extended.
Page offset	The field in a PCI address used to index into a selected page of memory.
PAPR	Power Architecture Platform Requirements.
PCI®	Peripheral Component Interface.
PCIe®	PCI Express.
PCI-X®	PCI Extended.
PE	Partitionable endpoint. The smallest entity that can be partitioned in endpoint partitioning. See Appendix A, Endpoint Partitioning [49].
PE#	Partitionable endpoint number.
PELE-V	PE lookup entry (vector). The RTT associates a Bus/Device/Function number of an incoming PCIe transaction to either a PE# or an index into the PELT-V table when the operation is an error message. The PELE-V contains a vector of bits indicating which PE numbers are affected by the incoming RID.
PELT-V	PE lookup table (Vector). A table of PELE-Vs. See also PELE-V.
PESE	PE state entry.
PF	The physical function of an IOV adapter. For more information, see the *PCI-SIG I/O Virtualization (IOV) Specifications*.
PHB	PCI host bridge. An entity that attaches a PCIe bus to the system.
PHB chip	The hardware chip where the PHB is implemented. The PHB might only be part of the chip functionality; for example, when the PHB is implemented on the processor chip. In that case, the processor chip becomes the PHB chip for purposes of this architecture.
PTE	Page table entry. Used for processor *Load/Store*address translation like the TVE/TCE is used for I/O address translation. Used to translate MMIO addresses as well.
RBA	Reject bit array. See Table 3.18, “RBA Definition” [46].
RC	Root complex. Connects a PCIe bus into the system. PCIe PHB is used in this document in place of the RC terminology.
RID	Requester ID. A name for the combined Bus/Dev/Func fields. The RID is attached to each PCIe transaction. It uniquely identifies the requester of the transaction. Given the uniqueness of this identifier, it is used by IODA2 to separate facilities in the PHB that are unique to the Func requesting the operation (for example, address translation, interrupt validation, and so on). See also Bus/Dev/Func and CID.
Root complex	“An entity that includes a host bridge and one or more root ports.” (PCI-SIG. *PCI Express Base Specification*. 2003 *)*
Root port	“A PCI express port on a root complex that maps a portion of the hierarchy through an associated virtual PCI-PCI bridge.” (PCI-SIG. *PCI Express Base Specification*. 2003)
RPN	Real page number. The bits in the TCE that are used to replace the high-order I/O bus address bits.
RTAS	Run-Time Abstraction Services.
RTC	RID translation cache. An optional PHB implementation that allows for better PCI transaction performance when the RTT is in system memory. See also RTT.
RTE	RID translation entry. An entry in the RTT. See Table 3.1, “RTE Definition” [10].
RTT	RID translation table. A 64K-entry table that takes the 16-bit RID from a PCI transaction and maps that to a PE# (for DMA and MSI operations) or to a PELE-V (when the operation is an error message). See also RTE.

Stopped state	More informally called the EEH Freeze state, this state occurs after an I/O error. In this state, MMIO *Store*s to the affected IOA are discarded, MMIO *Load*sare returned without error with data of all ones, and DMA operations from the IOA are aborted. This state prevents an IOA, after an error, from causing any damage to the system. Therefore, the IOA can be restarted with the knowledge that there are no data-integrity-type errors caused by the error that caused the Stopped state. In addition, the IOA Stopped state can be broken down into the MMIO Stopped state and the DMA Stopped state. If the DMA Stopped state is set, DMA for that IOA or PE is stopped. If the MMIO Stopped state is set, the MMIO is stopped. When first entering the Stopped state, both the MMIO and DMA Stopped states are set by the hardware. The DD might subsequently reset the MMIO Stopped state while leaving the DMA Stopped state set, to be able to query its IOA and recover it. In this document, if “MMIO” or “DMA” is not specified along with “Stopped state”, the reference is either to the general concept or to both the MMIO and DMA Stopped states.
TC	Traffic class. In PCIe, this defines a priority between PCI transactions within a VC. See also VC.
TCE	Translation control entry. Used to translate an I/O address page number to a real page number in system memory. See Table 3.6, “TCE Definition” [24]for the TCE definition.
TCE index	The field in a PCI address that is used to index into the TCE table to get the TCE.
TCE table	Translation control entry table. The table that contains the TCEs.
TLP	Transaction layer packet.
TTA	TCE table address. The address of the start of the TCE table. It is contained in the TVE.
TVE	Translation validation entry. An entry in a TVT. Used to translate and validate an IOA’s access to a DMA address space. See Table 3.5, “TVE Definition” [23]for the TVE definition.
TVT	Translation validation table (in PHB). Table containing TVEs. See also TVE.
UE	Uncorrectable error.
UR	Unsupported request.
VC	Virtual channel. In PCIe, a virtual channel defines a separate set of resources.
VF	The virtual function of an IOV adapter. For more information, see the *PCI-SIG I/O Virtualization (IOV) Specifications*.
XIVE	External interrupt vector table entry. Provides the interrupt priority and server number for routing an LSI interrupt. A priority of 0xFF is assumed to mean that the interrupt is disabled. See Table 3.10, “XIVE Definition for LSI Interrupts Only” [33]for the XIVE definition.
XIVT	External interrupt vector table. The table that contains the XIVEs. See also XIVE.

Appendix D. OpenPOWER Foundation overview

The OpenPOWER Foundation was founded in 2013 as an open technical membership organization that will enable data centers to rethink their approach to technology. Member companies are enabled to customize POWER CPU processors and system platforms for optimization and innovation for their business needs. These innovations include custom systems for large or warehouse scale data centers, workload acceleration through GPU, FPGA or advanced I/O, platform optimization for SW appliances, or advanced hardware technology exploitation. OpenPOWER members are actively pursing all of these innovations and more and welcome all parties to join in moving the state of the art of OpenPOWER systems design forward.

To learn more about the OpenPOWER Foundation, visit the organization website at openpowerfoundation.org.

D.1. Foundation documentation

Key foundation documents include:

- *Bylaws of OpenPOWER Foundation*
- *OpenPOWER Foundation Intellectual Property Rights (IPR) Policy*
- *OpenPOWER Foundation Membership Agreement*
- *OpenPOWER Anti-Trust Guidelines*

More information about the foundation governance can be found at openpowerfoundation.org/about-us/governance.

D.2. Technical resources

Development resouces fall into the following general categories:

- Foundation work groups
- Remote development environments (VMs)
- Development systems
- Technical specifications
- Software
- Developer tools

The complete list of technical resources are maintained on the foundation Technical Resources web page.

D.3. Contact the foundation

To learn more about the OpenPOWER Foundation, please use the following contact points:

- General information -- <info@openpowerfoundation.org>
- Membership -- <membership@openpowerfoundation.org>
- Technical Work Groups and projects -- <tsc-chair@openpowerfoundation.org>
- Events and other activities -- <admin@openpowerfoundation.org>
- Press/Analysts -- <press@openpowerfoundation.org>

More contact information can be found at openpowerfoundation.org/get-involved/contact-us.