OpenPOWER I/O Design Architecture

Workgroup Specification
Revision 3.0_prd1 (March 23, 2020)
** OpenPOWER Foundation Work Group Confidential **
Abstract

The purpose of the I/O Design Architecture, version 3 (IODA3) specification is to describe the chip architecture for key aspects of PCI Express® (PCIe)-based host bridge (PHB) designs for IBM® POWER9™ systems.

This document is a Standard Track, Work Group Specification work product owned by the Hardware Architecture Workgroup and handled in compliance with the requirements outlined in the OpenPOWER Foundation Work Group (WG) Process document. It was created using the Master Template Guide version 1.0.0. Comments, questions, etc. can be submitted to the public mailing list for this document at <hwarch-ioda@mailinglist.openpowerfoundation.org>.

Acknowledgement to members of the workgroup for their contributions
# Table of Contents

Preface .......................................................................................................................................... vi

1. Conventions ......................................................................................................................................................... vi

2. Document change history ........................................................................................................................................ vii

1. About This Document .............................................................................................................................................. 1

1.1. Purpose ........................................................................................................................................................... 1

1.2. Numbering Conventions ....................................................................................................................................... 1

1.3. Reference Documentation ....................................................................................................................................... 1

2. Introduction ............................................................................................................................................................ 2

2.1. Conformance to this Specification ....................................................................................................................... 2

2.2. General Information ................................................................................................................................................ 2

3. Design Specifics ....................................................................................................................................................... 6

3.1. High-Level Specifics ................................................................................................................................................ 6

3.2. Lower-Level Details ................................................................................................................................................ 6

A. Endpoint Partitioning ........................................................................................................................................... 38

A.1. Endpoint Partitioning Overview ........................................................................................................................ 38

A.2. Endpoint Partitioning Functional Specifics ......................................................................................................... 39

B. No-Translate Operation ............................................................................................................................................ 46

B.1. No-Translate Example ........................................................................................................................................... 48

C. Glossary ................................................................................................................................................................. 50

D. OpenPOWER Foundation overview ....................................................................................................................... 53

D.1. Foundation documentation .................................................................................................................................. 53

D.2. Technical resources ................................................................................................................................................ 53

D.3. Contact the foundation ........................................................................................................................................... 54
List of Figures

3.1. Partitionable Endpoint Number Determination for DMA and Error Messages ........................... 9
3.2. PCIe Non-MSI DMA Operation Address Fields ........................................................................ 15
3.3. DMA Operation High-Level Diagram: No Page Migration ......................................................... 16
3.4. I/O Address Validation and TCE Translation Implementation for 32-Bit DMA Addresses ....... 19
3.5. I/O Address Validation and TCE Translation Implementation for 64-Bit DMA Addresses ...... 20
3.6. Memory Migration Operation for a 64 KB Page and a 4 KB Page within the 64 KB Page ...... 26
3.7. Source and Destination Page Address Creation for DMA to a Page Being Migrated .......... 27
3.8. PCIe Normal DMA Operation for a Three-Level TCE Table ...................................................... 30
3.9. MSI Flow ................................................................................................................................... 33
A.1. Example System Configurations: Partitionable-Endpoint Definition ...................................... 39
B.1. TVE and Partitionable-Endpoint Number Determination ........................................................... 47
B.2. Example Physical Address Map with TCE Bypass Enabled for Some Partitionable End-
points ........................................................................................................................................... 49
List of Tables

3.1. RTE Definition ....................................................................................................................... 10
3.2. RTC Invalidate Register Definition ......................................................................................... 10
3.3. PELE-V Definition ................................................................................................................... 10
3.4. Supported Errors for PCI Express Error Injectors ................................................................. 14
3.5. TVE Definition ....................................................................................................................... 21
3.6. TCE Definition ....................................................................................................................... 23
3.7. TCE Invalidate Register Definition ......................................................................................... 24
3.8. Migration Register Definition ................................................................................................. 28
3.9. DMA Read Sync Register ..................................................................................................... 31
3.10. MSI IVE Definition ............................................................................................................... 35
3.11. IVC Invalidate Register Definition ........................................................................................ 35
3.12. PESE Definition ................................................................................................................... 36
B.1. No-Translate Operation ......................................................................................................... 46
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.

Changes

At certain points in the document lifecycle, knowing what changed in a document is important. In these situations, the following conventions will used.

- **New text will appear like this.** Text marked in this way is completely new.
- **Deleted text will appear like this.** Text marked in this way was removed from the previous version and will not appear in the final, published document.
- **Changed text will appear like this.** Text marked in this way appeared in previous versions but has been modified.

Command prompts

In general, examples use commands from the Linux operating system. Many of these are also common with Mac OS, but may differ greatly from the Windows operating system equivalents.

For the Linux-based commands referenced, the following conventions will be followed:

$ 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.
Document links

Document links frequently appear throughout the documents. Generally, these links include a text for the link, followed by a page number in parenthesis. For example, this link, Preface [vi], references the Preface chapter on page vi.

2. Document change history

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

The following table describes the most recent changes:

<table>
<thead>
<tr>
<th>Revision Date</th>
<th>Summary of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 23, 2020</td>
<td>Rev 3.0_prd1: Marking/version updates for public review</td>
</tr>
<tr>
<td>February 27, 2020</td>
<td>Rev 1.0-pre3: Removed the markups of the 1.0-pre2 version</td>
</tr>
<tr>
<td>February 26, 2020</td>
<td>Rev 1.0-pre2: Fixed typos</td>
</tr>
<tr>
<td>March 1, 2018</td>
<td>Rev 1.0-pre1: Creation based on IBM IODA2 Specification - revision 1.0</td>
</tr>
</tbody>
</table>
1. About This Document

1.1. Purpose

The purpose of the I/O Design Architecture, version 3 (IODA3) specification is to describe the chip architecture for key aspects of PCI Express® (PCIe)-based host bridge (PHB) designs for IBM® POWER9™ 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.

Numbers are generally shown in decimal format, unless designated as follows:

- Hexadecimal values are usually preceded by “0x.”
  For example: 0x000000000204000D
- Binary values in sentences are usually preceded by “0b.”
  For example: 0b1

1.3. Reference Documentation

For additional information, see the following documents:

*IBM® Power Architecture® Platform Requirements (PAPR)*

*External Interrupt Virtualization Engine Specification*
2. Introduction

The purpose of the I/O Design Architecture, version 3 (IODA3) specification is to describe the chip architecture for key aspects of PCI-Express™ (PCIe)-based host bridge (PHB) designs for IBM POWER9™ 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-SIG™ 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 PCIe. 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 Glossary (page 50).

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 mandated by the specific requirement.

2. Firmware Requirement: The platform firmware must implement all the requirements specified as “Firmware Requirements” in this architecture, unless otherwise mandated 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, 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 are called memory-mapped I/O (MMIO).

1There 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, which is used by both MMIO and DMA. Some reference are made to the I/O space as it relates to MMIO.
2.2.2. Translation Control Entries

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 unsupported request that cannot access all of the system memory address space. For example, a 32-bit I/O adapter must have its address expanded for systems with more than 4 GB of system memory. Otherwise, such an I/O adapter must use DMA to transfer its data to and from a buffer in the lower address range. The processor must 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 I/O adapter 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 I/O adapters by providing hardware scatter-gather. This provides the I/O adapter with a contiguous address space instead of one that is broken at every 4K page boundary. This can actually improve the performance of some I/O adapters if the platform can perform the TCE manipulation faster than the I/O adapter can process scatter/gather lists.

- Provide extra protection from I/O adapter hardware, firmware, and device driver errors by limiting the system memory that can be accessed by the adapter to those areas required by the outstanding I/O operations. In addition, limit the type of accesses (read/write) required by providing read-only and write-only (as well as read-write) protection through two control bits in the TCE.

2.2.3. Message-Signalled Interrupts

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

- Through a signal pin. This is called a level-signalled interrupt, or LSI.

- Through a message. This is called a message-signalled interrupt, or MSI.

MSIs have the advantage of pushing an I/O adapter’s DMA data that it has previously written ahead of it. Therefore, when the interrupt is presented, the device driver knows that the data is in the processor’s coherency domain. That is, it is immediately available.

2For PCIe, the LSI interrupts are not signalled by a physical pin (sometimes called out-of-band signaling), but rather through a logical pin that is shipped across the PCIe fabric as a packet (sometimes called in-band signaling).

3The “message” for MSI is really a DMA write operation to a special address with special data, as far as the I/O adapter and the I/O fabric are concerned.
There is no such guarantee with an LSI. Therefore, when the device driver sees an LSI, it must perform a \textit{Load} instruction targeted to its I/O adapter. Then, it must wait for the \textit{Load} data to return before being assured that the previously DMAed data is available to be used. This \textit{Load} is a performance penalty. In addition, PCIe allows only four LSIs per PCI host bridge, which severely restricts usability.

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 I/O adapter. The I/O adapter can have up to eight functions, giving up to 256 interrupts per I/O adapter.
- For MSI-X: Up to 2K interrupts per function of the I/O adapter. The I/O adapter can have up to eight functions, giving up to 16K interrupts per I/O adapter.

However, this flexibility comes at a cost in the scalability of the interrupt controller structure. IODA3 incorporates the interrupt virtualization source engine of the External Interrupt Virtualization Engine Architecture. For more information, see \textit{External Interrupt Virtualization Engine Specification}.

2.2.4. Enhanced Error Handling

Enhanced error handling 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. Enhanced error handling 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.

Enhanced error handling stops operations to and from an I/O adapter when an error is detected with that I/O adapter. This stopped condition is called the Stopped state\textsuperscript{4}. The Stopped state has the following key requirements:

- The I/O adapter 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 device driver to be isolated to just that device driver. This implies extra hardware or firmware to support the continuation of the I/O operation of other I/O adapter functions when an error is generated from another I/O adapter function.

\textbf{Exceptions:}

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

- For a plug-in adapter where the enhanced-error-handling functionality is implemented above the physical plug-in connector and where the plug-in adapter has multiple I/O adapter functions on it under a PCI-to-PCI bridge
- For an I/O adapter that has multiple functions on it, and for which there exist multiple device drivers (potentially one per function)

- Software (device driver or above) must not be able to introduce an error that can cause a Stopped state of other I/O adapter functions. That is, it must not introduce a Stopped-state error to I/O adapter functions other than the ones controlled by the device driver.
  - Software might, for example, improperly set up the TCEs for an I/O operation or pass the wrong address to its I/O adapter. This can cause an access to a TCE that is invalid. (The

\textsuperscript{4}Sometimes this state is also referred to as the “freeze” state or condition. In addition, the I/O adapter 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.
TCE is not set up; or the TCE is set to read-only for a write or PCI atomic operation; or the TCE is set to write-only for a read or PCI atomic operation.) This causes a Stopped state.

- It is acceptable for a platform hardware error, but not for a device driver or I/O adapter function hardware error, to affect multiple I/O adapter functions. However, the recovery from such an error must be transparent to the device driver. That is, the platform makes it appear to all I/O adapter functions that they have encountered the error condition themselves.

- The device driver must be able to detect the Stopped state condition.

- The device driver (and, therefore, the platform) must be able to remove its I/O adapter function from the MMIO Stopped state for MMIO operations, independent of other I/O adapter functions.

- The capturing of fault information for problem determination must be allowed after the Stopped state condition occurs.

- The device driver (and, therefore, the platform) must be able to remove the I/O adapter function from the DMA Stopped state for DMA operations independent of other I/O adapter functions. The device driver is responsible for bringing its I/O adapter function to a known good state before removing it from the DMA Stopped state, to avoid the possibility of improper operations from its I/O adapter function. In many cases, the device driver needs to bring its I/O adapter 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 I/O adapter function while the I/O adapter function is in the DMA Stopped state.
3. Design Specifics

This chapter describes applicable design specifics as they apply to IBM Power Systems™ starting with IBM POWER9™. 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 Power Architecture Platform Requirements.

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 across an I/O fabric. For more information, see Appendix A, Endpoint Partitioning [38].

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 mandated 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 mandated by the specific requirement.

3.2. Lower-Level Details

3.2.1. Partitionable Endpoint Number Details

For any given operation, the PCI host bridge hardware determines the partitionable-endpoint numbers to which the operation belongs. It tracks the state of that partitionable-endpoint number so that it can stop the partitionable endpoint on an error and prevent further operations after the error. It does this on a per partitionable-endpoint number basis so that unaffected partitionable endpoints can continue to operate while the affected partitionable endpoint is recovered. For more information, see Section 3.2.1.3, “Partitionable-Endpoint State and Enhanced Error Handling” [12].

The partitionable-endpoint number 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 partitionable-endpoint number. This is done in an implementation-dependent way. For requirements, see Section 3.2.1.1, “MMIO Partitionable-Endpoint Number 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 Partitionable-Endpoint Number Determination” [7].
– If the operation is a DMA or MSI operation, the partitionable-endpoint number field of the RID translation entry (RTE) indicates the partitionable-endpoint number associated with the RID.

– If the operation is an error message, the partitionable-endpoint number field is the index of the partitionable-endpoint lookup table (vector) (PELT-V). When the PELT-V is accessed, the entry indicates, by a vector of bits, which partitionable-endpoint numbers are affected by the RID. PELT-V entries are generated by firmware for all partitionable-endpoint numbers. That is, the depth of the PELT-V is equal to the number of partitionable-endpoints implemented. Hierarchical RIDs, such as switch RIDs and IOV PFs, have more than one bit set in the partitionable-endpoint look-up entry (vector) (PELE-V). Single RIDs, such as those for VFs, have only one bit set.

### 3.2.1.1. MMIO Partitionable-Endpoint Number Determination

**Note:** For more information about this implementation-dependent facility, see the POWER9 Processor User’s Manual.

PCI host bridges are required to support MMIO address-space decoding and the assignment of partitionable-endpoint numbers 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 PCI host bridge hardware must support the decoding of MMIO addresses, and both of the following conditions must be met:

1. Enough address-space decodes must be provided to support the necessary, probably noncontiguous, Base Address Register (BAR) spaces of the devices to be located below the PCI host bridge, including legacy devices that require an address programmed in their BARs that are below 4 GB.

2. The address decodes must be assigned an appropriate partitionable-endpoint number. When multiple decodes are provided for any given function, the partitionable-endpoint number assigned must be the same.

**Hardware Implementation Note:**

Relative to requirement R1-3.2.1.1-1, how the hardware implements this requirement 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 PCI host bridge, 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 Partitionable-Endpoint Number Determination

The RTT and PELT-V tables are implemented in system memory. The most recently used RTT entries (RTEs) are cached in the PCI host bridge 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 partitionable-endpoint number field.
If the operation is a DMA or MSI operation, the RTE indicates the partitionable-endpoint number associated with the RID. If the operation is an error message, the partitionable-endpoint number field is the index of the PELT-V. When the PELT-V is accessed, the entry indicates, by a vector of bits, which partitionable-endpoint numbers are affected by the RID. PELT-V entries are generated by firmware for all partitionable-endpoint numbers. That is, the depth of the PELT-V is equal to the number of partitionable-endpoints 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 partitionable-endpoint number are stored in a cache on the PCI host bridge chip. The cache is referenced first in the partitionable-endpoint translation process. If the entry is not in the cache, a reference is made to the RTT in system memory.

Figure 3.1 [9] shows how partitionable-endpoint numbers are determined for DMA and error messages. For specific hardware and firmware requirements related to this, see R1-3.2.1.2-1 [10], and R1-3.2.1.2-2 [11].

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 partitionable endpoints implemented, with one entry per partitionable endpoint and one bit width for each partitionable endpoint, and with the width in bytes being a power of 2.
Figure 3.1. Partitionable Endpoint Number 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).
OpenPOWER I/O Design Architecture

March 23, 2020

Revision 3.0_prd1

OpenPOWER Foundation

Work Group Confidential 10

Standard Track

Table 3.1. RTE Definition

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Bits</th>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:1</td>
<td>All</td>
<td>PE#</td>
<td>The partitionable-endpoint number or index into the PELT-V. A partitionable-endpoint number of all ones is invalid. Therefore, firmware must set the partitionable-endpoint number to all ones for RIDs that are not configured.</td>
</tr>
</tbody>
</table>

Table 3.2. RTC Invalidate Register Definition

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invalidate All</td>
<td>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.</td>
</tr>
<tr>
<td>1:15</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>16:31</td>
<td>Requester ID (0:15)</td>
<td>The 16-bit Requester ID field of the RTC entry to invalidate when the Invalidate All bit is set to a zero.</td>
</tr>
<tr>
<td>32:63</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3. PELE-V Definition

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Bits</th>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:[n]</td>
<td>[All]</td>
<td>PE Array</td>
<td>An array of bits, one bit per partitionable endpoint, that indicates which partitionable endpoints are affected. An implementation only needs to support the number of bits necessary to support the number of partitionable endpoints that it supports. Unused bits are at the highest bit numbers (bit 0 of byte 0 corresponds to the first partitionable-endpoint number, bit 1 to the second, and so on). The number of bytes implemented is a power of 2.</td>
</tr>
</tbody>
</table>

R1-3.2.1.2.1 Hardware Requirement: The PCI host bridge 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 partitionable endpoints.
2. In e of this requirement, it is best for the optional caching for MSI operations to be controllable on a per-PCI host bridge 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 to invalidate, but it 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 Note: Relative to requirement f, the hardware is required to provide firmware with a way to clean up cache entries when they are no longer needed or when the RID to partitionable-endpoint number 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 PCI host bridge, and read the RTE. If this is a DMA operation, cache the entry. For MSI operations and for error messages, optionally cache the entry. For error messages, if the partitionable-endpoint number 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 partitionable-endpoint numbers that are affected by the error.

Architecture Note: DMAs from entities such as PCIe IOV PFs, and MSIs from PFs, and switch RIDs have only one partitionable-endpoint number associated with them. Error messages from PFs and switch RIDs are likely to point to multiple partitionable-endpoint numbers 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 PCI host bridge, 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 PCI host bridge (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 operations or that issue MSIs, set up the partitionable-endpoint number in the RTT to the partitionable-endpoint number that is associated with the RID. There might be more than one RID associated with the same partitionable-endpoint number.

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

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 PCI host bridge chip appropriately at all times, including during hot plug and DLPAR operations.

### 3.2.1.3. Partitionable-Endpoint State and Enhanced Error Handling

The partitionable-endpoint state includes, but is not limited to, the following states:

- The Enhanced-Error-Handling Enablement state: Indicates whether enhanced error handling is enabled for the partitionable endpoint or not.
- The MMIO Stopped state: Indicates whether MMIO operations are frozen for the partitionable endpoint or not. If MMIO is stopped for the partitionable endpoint, the partitionable endpoint 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 partitionable endpoint or not. If DMA is stopped for the partitionable endpoint, the partitionable endpoint is said to have its DMA Stopped state set or to be in the DMA Stopped state.

**Note:** For enhanced-error-handling-enabled device drivers, on the detection that their partitionable endpoint 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 partitionable endpoint 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 partitionable endpoint or by separately removing the I/O adapter function from the DMA Stopped state. The latter approach might not be possible for some I/O adapter functions or under certain circumstances.

**R1-3.2.1.3-1 Hardware Requirement:** Each partitionable endpoint’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 partitionable endpoints
- Atomically with any other errors that might be occurring at the time

**R1-3.2.1.3-2 Hardware Requirement:** The PCI host bridge hardware must take all of the following actions:

- For any detected failure to or from a partitionable endpoint, set both the MMIO Stopped and DMA Stopped states for the partitionable endpoint.
  
  **Exception:** Not required if the error that caused the failure can be reported to the I/O adapter function in a way that enables it to report the error to its device driver while avoiding any data corruption.

- 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 partitionable endpoint, put all partitionable endpoints that are downstream of the failure into the MMIO Stopped and DMA Stopped states if they might be affected by the failure.

- From the time that the MMIO Stopped state is entered for a partitionable endpoint, prevent the partitionable endpoint from responding to Load and Store operations including the operation that caused the partitionable endpoint to enter the MMIO Stopped state. A Load operation must
return all ones with no error indication. A Store operation must be discarded until the firmware directs the hardware otherwise or until the PCI host bridge 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 partitionable endpoint, prevent the partitionable endpoint from initiating a new DMA request or completing a DMA request that caused the partitionable endpoint to enter the DMA Stopped state, including MSI or MSI-X DMA operations, until the firmware directs the hardware otherwise or until the PCI host bridge 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 partitionable endpoint being in the DMA Stopped state (for example, a read request or an atomic request), return an unsupported request (UR) to the requester. DMA read response data that is returned to the PCI host bridge 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-partitionable-endpoint basis, that a failure occurred which caused the partitionable endpoint 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 partitionable endpoint without disturbing other partitionable endpoints on the platform. The hardware must provide this capability without requiring a partitionable-endpoint reset and must do so through normal processor *Store* instructions. Firmware enabling of MMIO or DMA Stopped states must have the same, immediate effect as if a PCI host bridge-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 changing 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 changing the reset.

i. When a partitionable endpoint 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 PCI host bridge and I/O fabric PCI configuration registers when any or all of the partitionable endpoints are in the MMIO Stopped state.

**Hardware Implementation Notes:**

1. The type of error information trapped by the hardware when a partitionable endpoint 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 an error status, must reach the PCI host bridge 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 PCI host bridge 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 partitionable endpoint into the MMIO and DMA Stopped states, implementations can block DMA operations that were kicked off after a failing
DMA operation. Implementations can 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 enhanced-error-handling option in that bridge. Specifically, the Power Architecture Platform Requirements (PAPR) `ioa-bus-error` and `ioa-bus-error-64` functions of the `ibm,errinjct` RTAS call are used to inject errors onto each partitionable-endpoint primary bus. This, in turn, causes certain actions on the bus and certain actions by the partitionable endpoint, by the enhanced-error-handling 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 PCI host bridge hardware must take all of the following actions:

a. Provide a way to inject the required errors for each partitionable-endpoint 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 partitionable endpoint.

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
   - Partitionable-endpoint 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>
<thead>
<tr>
<th>Operation</th>
<th>PCI Address Spaces</th>
<th>Errors</th>
<th>Other Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>Memory, I/O, Config</td>
<td>TLP ECRC Error</td>
<td>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.</td>
</tr>
<tr>
<td>Store</td>
<td>Memory, I/O, Config</td>
<td>TLP ECRC Error</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.4. Supported Errors for PCI Express Error Injectors
3.2.2. DMA Design, Translation Validation Entries, and Translation Control Entries

This section describes the constructs for DMA operations, except when those DMA operations 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, “Message-Signalled Interrupt Design” [32].

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” [21]).

**Figure 3.2. PCIe Non-MSI DMA Operation Address Fields**

![Figure 3.2. PCIe Non-MSI DMA Operation Address Fields](image)

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, “DMA Design Details: Page Migration” [24].

The differences for multilevel tables are described in Section 3.2.2.3, “DMA Design Details: Multilevel TCE Tables” [29].

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

1. The I/O adapter 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 partitionable-endpoint number.

2. The partitionable-endpoint number is determined from a 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 partitionable-endpoint number is in the DMA Stopped state, abort the operation (see Section 3.2.1.3, “Partitionable-Endpoint State and Enhanced Error Handling” [12]).
   - If the operation is a DMA Read Request, return an unsupported request (UR) error to the requester.
   - If the operation is a DMA Write, discard the data.

4. The partitionable-endpoint number and one or more high-order address bits are used to access the correct TVE for the partitionable endpoint.
   - 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). PCI host bridges operate bimodally relative to the whole PCI host bridge, 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 I/O adapter 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 [46].

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 partitionable endpoint:
     - 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 Defini-
   tion” [21]).

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

For more information, see:

   • Figure 3.4 [19], which shows how the translation from I/O address to system memory address
     works for 32-bit I/O addresses

   • Figure 3.5 [20], which shows how the translation from I/O address to system memory address
     works for 64-bit I/O addresses

Notes on Figure 3.4 and Figure 3.5:

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 I/O adapter 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 depends on the maximum
   size of system memory 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” [21].

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 partitionable endpoint. However, the implementation is not required to do so.

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

![Diagram of I/O Address Validation and TCE Translation Implementation](image)

*Note:* Some addresses and registers are shown in big-endian notation to match the registers in the tables. There is no bit swapping.
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 partitionable-endpoint number 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 PCI host bridge hardware must take all of the following actions:

a. Implement the TVE table, as defined by Table 3.5, “TVE Definition” [21], with the TVT being located on the PCI host bridge chip.

b. Implement the TCE table, with entries as defined by Table 3.6, “TCE Definition” [23], with the TCE table being in system memory and cached on the PCI host bridge chip.

c. Provide a TCE Invalidate Register, as defined in Table 3.7, “TCE Invalidate Register Definition” [24], for invalidating cached TCEs, all TCEs for a particular partitionable endpoint, or the entire cache of TCEs. The hardware must stop using the entry when firmware indicates to 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” [20].

R1-3.2.2.1-2. **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” [21], [ ] 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

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Bits</th>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:5</td>
<td>All</td>
<td>TTA or Address range for no translate</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 [46].

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Bits</th>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
</table>
| 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, “DMA Design Details: Multilevel TCE Tables” [29].  
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 (cont.) | 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 I/O adapter 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 TCE Index bits, 8 KB table size  
Value of 3: 11 TCE Index bits, 16 KB table size  
Value of 4: 12 TCE Index bits, 32 KB table size  
Value of 5: 13 TCE Index bits, 64 KB table size  
Value of 6: 14 TCE Index bits, 128 KB table size  
Value of 7: 15 TCE Index bits, 256 KB table size  
Value of 8: 16 TCE Index bits, 512 KB table size  
Value of 9: 17 TCE Index bits, 1 MB table size  
Value of 10: 18 TCE Index bits, 2 MB table size  
Value of 11: 19 TCE Index bits, 4 MB table size  
Value of 12: 20 TCE Index bits, 8 MB table size  
Value of 13: 21 TCE Index bits, 16 MB table size  
Value of 14: 22 TCE Index bits, 32 MB table size  
Value of 15: 23 TCE Index bits, 64 MB table size  
Value of 16: 24 TCE Index 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 |
Bytes | Bits | Field | Definition
--- | --- | --- | ---
7 | [0:2] | Reserved | The number of low-order I/O bus address bits to be used as the page offset (see the Page Offset field in Figure 3.2, “PCIe Non-MSI DMA Operation Address Fields” [15]) is the value of this field plus 11.

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).

3:7 | I/O Page Size | The number of low-order I/O bus address bits to be used as the page offset (see the 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 PCI host bridge 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

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
</table>
| 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 PCI host bridge implementations, all of these bits might not be required. However, enough bits must be implemented to match the largest real address in the platform.

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 nonzero, 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, “DMA Design Details: Multilevel TCE Tables” [29].

52:55 | Migration Pointer | Used during a migration operation. The meaning is as follows:

0b0000 A migration is not in process for this page.
Bits | Field | Definition
---|---|---
nonzero | A migration is in progress, and the value of this field points to which Migration Register in the PCI host bridge is used for the operation. For more information, see Section 3.2.2.2, “DMA Design Details: Page Migration” [24].

56:61 | 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 a 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 host bridge generates an error. For more information, see the “Error and Event Notification” chapter in the Power Architecture Platform Requirements (PAPR).

### Table 3.7. TCE Invalidate Register Definition

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
</table>
| 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 partitionable-endpoint number.
- 0b001: Invalidate the TCE for the specified PCIe bus address and partitionable-endpoint number.

| 3 | Reserved |

| 4:51 | Invalidate Address | PCIe address for the specified partitionable-endpoint number 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 |

| 56:63 | PE# | Partitionable-endpoint number 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. DMA Design Details: Page Migration

The page-migration facilities in the PCI host bridge 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 PCI host bridge hardware.

- Migration Register in the PCI host bridge 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 can have 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 [27].

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” [26]. 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” [27].

The target page address translation of “Bits 4:(63-N) of the Migration Register || TCE-translated source page address bits (64-N):63” is described in Requirement R1-3.2.2.2-1 [27]. It can also be described in a different way, as follows and shown in Figure 3.6, “Memory Migration Operation for a 64 KB Page and a 4 KB Page within the 64 KB Page” [26] and Figure 3.7, “Source and Destination Page Address Creation for DMA to a Page Being Migrated” [27]:

- 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
R1-3.2.2.2-1. **Hardware Requirement:** The PCI host bridge 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" [28]. 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” [23]. When that field is nonzero, perform the operations described by d through i 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:

   \[
   \text{Bits } 4:(63 - N) \text{ of the Migration Register } \parallel \text{TCE-translated source page address bits } (64 - N):63
   \]

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.

R1-3.2.2.2-2. **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 PCI host bridge 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 zero. 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 ensure 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 b of this requirement into the Migration Descriptor Register allocated in a of this requirement.

d. Redirect all TCEs pointing to the memory area to be migrated to the 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 a of this requirement.

g. Adjust all TCEs changed in 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

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Valid</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Invalid. Attempts to use causes the partitionable endpoint to be put into the enhanced-error-handling Stopped state.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid.</td>
</tr>
</tbody>
</table>
### 3.2.2.3. 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, "Multilevel Table TCE Fetching" [29].

#### 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 as an error and the fetching stops. All but the last level of tables contains what are called indirect TCEs, and the last level contains direct TCEs. The PCI host bridge 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" [30].
R1.3.2.2.3-1. **Hardware Requirement**: The PCI host bridge hardware must take all of the following actions:

a. Implement multilevel TCE tables as defined by Section 3.2.2.3, "DMA Design Details: Multilevel TCE Tables" [29].

b. Implement the Number of TCE Table Levels field according to the definition in Table 3.5, "TVE Definition" [21]. 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 enhanced-error-handling Stopped state for the partitionable endpoint.

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” [30], is all zeros, based on the TCE Table Size, Number of TCE Table Levels, and I/O Page Size fields of the TVE.

R1-3.2.2.3-1. 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.

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” [31] is provided to assist firmware in determining when all currently outstanding (in progress relative to the PCI host bridge’s state machine) DMA read operations are complete. For example, it can be used during a memory-migration operation or during partitionable-endpoint-reset operations to assure that in-flight DMAs are complete.

Table 3.9. DMA Read Sync Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Access Mode</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Start Synchronization</td>
<td>Write only</td>
<td>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.</td>
</tr>
<tr>
<td>1</td>
<td>Synchronization Complete</td>
<td>Read only</td>
<td>0 One or more DMA read operations, which were incomplete (outstanding) 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) 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.</td>
</tr>
<tr>
<td>2:63</td>
<td>Reserved</td>
<td>-</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

Note: A DMA Read operation is considered to be incomplete (outstanding) if processing on it has been started by the PCI host bridge’s state machine, such that the address translation process has started and cannot be aborted. For example, if the PCI host bridge 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 PCI host bridge from system memory.
R1-3.2.2.4-1. **Hardware Requirement:** The PCI host bridge hardware must provide the DMA Read Sync register, as defined in Table 3.9, “DMA Read Sync Register” [31].

### 3.2.3. Level-Signalled Interrupt Design

When a level-signalled interrupt (LSI) is signalled by the Assert_INTx message request (where x is A, B, C, and D for the respective PCI interrupt signals), it is processed as described in the *External Interrupt Virtualization Engine Specification*.

Use of level-signalled interrupts (LSIs) in systems has the following implications for firmware and hardware:

- There are only four LSIs per PCI host bridge, and any specific interrupt cannot be shared across partitionable endpoints. 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 into the PCI host bridge.

Unlike MSI interrupts, when the partitionable endpoint is in the DMA Stopped state, LSI operations from that partitionable endpoint are not prevented. The firmware can disable the interrupts on detection of an enhanced error handling event. On detection of the event, the device driver should also disable its I/O adapter interrupt in its PCI configuration space.

### 3.2.4. Message-Signalled Interrupt Design

This section describes the requirements for the conversion of PCIe message-signalled interrupt (MSI) messages into Interrupt Virtualization Source Engine event triggers. The further processing of these resulting event triggers is covered in the *External Interrupt Virtualization Engine Specification*.

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 PCI host bridge).

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 partitionable endpoint is in the DMA Stopped state, additional MSI operations that are presented from the partitionable endpoint after the partitionable endpoint enters the Stopped state are blocked by the PCI host bridge. 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 device driver determines that its I/O adapter function is stopped by timing operation completions and timing out when its I/O adapter function fails to signal completion after a reasonable period of time. MSI operations that were presented by a partitionable endpoint to the PCI host bridge before that partitionable endpoint entered the Stopped state can be presented after the partitionable endpoint enters the Stopped state. For example, an initial presentation was rejected by the presentation layer, and the PCI host bridge presents the same interrupt after it was rejected.
Figure 3.9, “MSI Flow” [33] 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. The following terms are used in Figure 3.9, “MSI Flow” [33].

- IVE – An entry in the interrupt vector table.
- IVC – Interrupt vector cache. A cache of IVEs.

**Figure 3.9. MSI Flow**

Referring to Figure 3.9, “MSI Flow” [33], 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 partitionable-endpoint number is compared to the partitionable-endpoint number generated from the RID (see Section 3.2.1.2, “DMA Partitionable-Endpoint Number Determination” [7]). If the partitionable-endpoint number in the IVE does not match the partitionable-endpoint number 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 partitionable-endpoint number is in error. The partitionable endpoint is placed into the error state (MMIO Stopped and DMA Stopped states). If the partitionable-endpoint numbers match, the processing continues as described in the *External Interrupt Virtualization Engine Specification*.

**R1-3.2.4-1. Hardware Requirement:** The PCI host bridge 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.

**Note:** It is permissible for implementations to decode, under configuration control, additional PCIe address ranges as MSI operations.

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

c. Implement the IVT with the entries defined as in Table 3.10, “MSI IVE Definition” [35].

d. Implement IVT BAR and IVT Length registers, writable by firmware, whose contents point 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” [33]):

   - 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 step 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 [34]).

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

f. Access the IVE with the address generated in e of this requirement. Then compare the PE# field from the IVE with the partitionable-endpoint number generated in Requirement R1-3.2.1.2-1 [10]. If they are not equal, put the partitionable endpoint into the MMIO and DMA Stopped states, as defined in Section 3.2.1.3, “Partitionable-Endpoint State and Enhanced Error Handling” [12].

g. Forward the event trigger to the Interrupt Virtualization Source Engine.

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

i. Provide an IVC Invalidate Register, as defined in Table 3.11, “IVC Invalidate Register Definition” [35], for invalidating IVC entries. The hardware must stop using the entry when firmware indicates to 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.

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 PCI host bridge 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 and initialize the corresponding Event State Buffer entries to the Off state. For more information, see the External Interrupt Virtualization Engine Specification.


d. Do not allow interrupts to be shared between partitionable endpoints.

e. Maintain the coherency between the system memory IVE contents and the cached IVE contents by using the IVC Invalidate Register to invalidate cached IVEs whenever it changes the value of an IVE in system memory.

**Firmware Implementation Note:** The PCI host bridge caches the partitionable-endpoint number. If a partitionable-endpoint number 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>
<thead>
<tr>
<th>Table 3.10. MSI IVE Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bytes</strong></td>
</tr>
<tr>
<td>0:5</td>
</tr>
<tr>
<td>6:7</td>
</tr>
<tr>
<td>8:15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3.11. IVC Invalidate Register Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bits</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1:15</td>
</tr>
<tr>
<td>16:31</td>
</tr>
<tr>
<td>32:63</td>
</tr>
</tbody>
</table>

3.2.5. PCIe Configuration Space

Firmware must be able to access the PCIe configuration space while the other partitionable endpoints remain in the MMIO Stopped state. Therefore, the partitionable endpoint for the configuration space cannot be shared by any other partitionable endpoint under the PCI host bridge.

**R1-3.2.5-1. Hardware Requirement:** Configuration access to an I/O adapter function and to the I/O fabric must be available at all times to the firmware, even though the MMIO Stopped state for a partitionable endpoint might be set. That is, the hardware must provide partitionable endpoints for the configuration space that are separate from the normal MMIO memory space partitionable endpoints.
3.2.6. Partitionable-Endpoint State Table

The PCI host bridge is required to capture certain data relative to partitionable-endpoint errors. The partitionable-endpoint state table (PEST) is in system memory, and the partitionable-endpoint state entry (PESE) is defined by Table 3.12, “PESE Definition” [36].

### Table 3.12. PESE Definition

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:1</td>
<td>Reserved</td>
<td>This bit is set to one if an MMIO operation froze the partitionable endpoint. It is set to zero if a DMA operation froze the partitionable endpoint. If both an MMIO and DMA operation attempt to freeze the endpoint in the same cycle, the MMIO operation has priority.</td>
</tr>
<tr>
<td>2</td>
<td>MMIO Cause</td>
<td>The operation that caused the partitionable endpoint to be frozen was a Power Architecture Platform Requirements (PAPR) inject MMIO.</td>
</tr>
<tr>
<td>3</td>
<td>CFG Read</td>
<td>The operation that caused the partitionable endpoint to be frozen was a Power Architecture Platform Requirements (PAPR) inject CFG Read.</td>
</tr>
<tr>
<td>4</td>
<td>CFG Write</td>
<td>The operation that caused the partitionable endpoint to be frozen was a Power Architecture Platform Requirements (PAPR) inject CFG Write, or a CFG Write with Size or Access error.</td>
</tr>
<tr>
<td>5:7</td>
<td>Transaction Type (0:2)</td>
<td>This is an encoding of the transaction type that caused this partitionable endpoint to be frozen. The encoding is as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000  DMA Write</td>
</tr>
<tr>
<td></td>
<td></td>
<td>010  DMA Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100  MMIO Load</td>
</tr>
<tr>
<td></td>
<td></td>
<td>110  Unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Examples include data parity errors, state machine errors, and so on.)</td>
</tr>
<tr>
<td>8</td>
<td>CA Return Status or</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Completion Timeout</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>UR Return Status</td>
<td>An MMIO Load, MMIO I/O Write, or other transaction returned from the PCIe link with a status of Unsupported Request (UR).</td>
</tr>
<tr>
<td>10</td>
<td>NONFATAL_ERROR</td>
<td>A PCIe nonfatal error occurred.</td>
</tr>
<tr>
<td>11</td>
<td>FATAL_ERROR</td>
<td>A PCIe fatal error occurred.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Parity/ECC UE Error</td>
<td>Any parity error or uncorrectable ECC error.</td>
</tr>
<tr>
<td>14</td>
<td>Correctable Error /</td>
<td>A correctable error occurred.</td>
</tr>
<tr>
<td></td>
<td>CORR_ERROR</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>PCIe Core Interrupt</td>
<td>An error occurred in the PCIe core.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid MMIO Address</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Translation / IODA3 Error</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>TCE Page Fault</td>
<td>A DMA transaction accessed a TCE whose page-access control bits were all zeros.</td>
</tr>
<tr>
<td>19</td>
<td>TCE Access Fault</td>
<td>A DMA transaction conflicted with its allowed permissions according to the TCE page-access control bits (includes all cases including page fault).</td>
</tr>
<tr>
<td>20</td>
<td>DMA Response Timeout</td>
<td>A timeout occurred while waiting for an outstanding DMA Read Response to return from system memory.</td>
</tr>
<tr>
<td>21</td>
<td>AIB Size Invalid</td>
<td>The Size field in an incoming AIB packet was not valid.</td>
</tr>
</tbody>
</table>
### Bits | Field | Definition
--- | --- | ---
22:25 | Reserved |  
26:31 | LEM Bit Number (0:5) | Bit number in the LEM FIR Accumulator Register for the error that froze this endpoint.  
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 |  
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 or 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 PCI host bridge hardware must take all of the following actions:

a. Implement the PEST in system memory, with entries defined by Table 3.12, “PESE Definition” [36].

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

c. Use the partitionable-endpoint number, with four trailing zeros concatenated, to index into the PEST.

d. When a partitionable endpoint is placed into the MMIO Stopped state, write the appropriate error information into the PESE for that partitionable-endpoint number.

**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 3.12, “PESE Definition” [36].

**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 partitionable-endpoint number before clearing the MMIO Stopped state for that partitionable endpoint.
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 requires some functionality in the bridges in the system be able to partition the I/O adapters or individual functions of an I/O adapter 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 partitionable endpoint is any part of an I/O subsystem that can be assigned to a partition independent of another partitionable endpoint. In Figure A.1, “Example System Configurations: Partitionable-Endpoint Definition” [39], examples of partitionable endpoints are shown encircled with dotted boxes.

There are also aspects of shared state. Examples include any element that is shared between partitions, such as PCI host bridges and switches, and that detects an error that cannot be isolated to a specific partitionable endpoint. In such cases, the error must be propagated to the state of all partitionable endpoints 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 partitionable endpoint does not exactly correspond to this unit.
A.2. Endpoint Partitioning Functional Specifics

Several functions in the PCI host bridge are partitioned per partitionable endpoint, and might have to keep state and control separate on a per partitionable-endpoint basis:

1. I/O adapter address domains (see Section A.2.2 [41])
   a. MMIO Load/Store address domains (see Section A.2.2.1 [41])
   b. Configuration space address domains (see Section A.2.2.2 [41])
   c. DMA I/O bus address domains and TCEs (see Section A.2.2.3 [42])
2. I/O adapter error domains (specifically for enhanced error handling) (see Section A.2.3 [42])
3. I/O adapter error injection domains (see Section A.2.4 [43])
4. Interrupts (see Section A.2.5 [44])
   a. LSI (see Section A.2.5.2 [44])
   b. MSI (see Section A.2.5.3 [45])

5. Partitionable-endpoint reset domains (see Section A.2.6 [45])

6. Partitionable-endpoint hot plug and power domains (see Section A.2.7 [45])

Some of these functions are partitioned by the PCI host bridge, 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. Partitionable-Endpoint Domains**

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

**A.2.1.1. Numbering**

The partitionable-endpoint domain number is the bond that associates the various domain components to the same partitionable endpoint. PCI defines several divisions that can differentiate I/O adapters:

- **Bus #** (or simply “Bus”). The highest level of division. Each bus or PCIe link under a PCI host bridge has a unique Bus #.
- **Device #** (or simply “Dev”) within the Bus #. Subdivides the I/O adapters 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 I/O adapter into functions. Multifunction I/O adapters have multiple function numbers, and single-function I/O adapters 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 Definition” [39]). In general, the larger the granularity:

- The less flexibility in the configuration
• The lower hardware implementation costs

• 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 ramifications 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 PCI host bridge implementations will probably 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 (partitionable-endpoint number), which is compact.

A.2.2. Address Domains

Separation of addressing between partitionable endpoints 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 partitionable-endpoint state is moved up in the PCI tree, some decoding must also be moved up. This is necessary to determine which partitionable-endpoint 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 I/O adapter is being addressed by the MMIO.

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

Thus, each partitionable-endpoint MMIO address range that is decoded is associated with a compact partitionable-endpoint number. This partitionable-endpoint number is tied to the Bus/Dev/Func number of the I/O adapter, as described in Section A.2.2.3, “DMA I/O Bus Address Domains and TCEs” [42].

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

• Enhanced-error-handling state (See Section A.2.3, “I/O Adapter Error Domains for Enhanced Error Handling” [42] for more information.)

• Error injection domains (See Section A.2.4, “I/O Adapter Error-Injection Domains” [43].)

• I/O ordering considerations

A.2.2.2. Configuration Space Address Domains

For PCIe host bridges, the hardware must not disable configuration accesses when the partitionable endpoint goes into the MMIO Stopped state. It must provide a separate partitionable-endpoint 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 I/O adapter to get access to all the physical memory that it needs to service the partitions that it needs to service. However, an I/O adapter 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 I/O adapter 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 device drivers.

This architecture provides structures that are defined to limit an I/O adapter to a range of bus addresses (the DMA I/O bus address domain of the I/O adapter), 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 partitionable endpoint. The index into the TVT is by the partitionable-endpoint number, 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 I/O adapter. This is called the TVE Index.

For a definition of the TVE fields, see Table 3.5, “TVE Definition” [21].

Figure 3.5, “I/O Address Validation and TCE Translation Implementation for 64-Bit DMA Addresses” [20] 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 partitionable endpoint for addresses smaller than 4 GB.

For more information about how this works, see Section 3.2.1, “Partitionable Endpoint Number Details” [6] and Section 3.2.2, “DMA Design, Translation Validation Entries, and Translation Control Entries” [15].

A.2.3. I/O Adapter Error Domains for Enhanced Error Handling

Enhanced error handling 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. Enhanced error handling 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.

Enhanced error handling in the PCI host bridge stops operations to and from a partitionable endpoint when an error is detected (called the Stopped state). Keys to this function are:

1. The partitionable endpoint must be prevented from completing the I/O operation in error:
   • In such a way that the partitionable endpoint 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 device driver to be isolated to just that device driver. This implies extra hardware or firmware to support the continuation of I/O operation of other partitionable endpoints in the face of an error generated from another partitionable endpoint.
Exceptions:

- A plug-in adapter that has multiple I/O adapters on it under a PCI-to-PCI bridge, and for which there exist multiple device drivers (potentially one per function). In this case, the device drivers for those multiple devices or multiple functions must coordinate any Stopped state recovery. These cooperating device drivers 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 device drivers have a communication path between the partitions.
  - The user of such shared I/O adapters understands that one partition can affect a sharing partition's performance by a denial-of-service type attack through causing of enhanced-error-handling Stopped states and the ensuing enhanced-error-handling recovery of operations.

- An I/O adapter that has multiple functions on it, and for which there exist multiple device drivers (potentially one per function) and for which the platform does not provide partitionable-endpoint 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 I/O adapters apply here.

3. Software (device driver or above) for one partitionable endpoint must not be able to introduce an error that can cause another partitionable endpoint 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 I/O adapter, 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 partitionable endpoints, as long as the recovery from it is transparent to the device driver. (That is, the platform makes it appear to all device drivers and partitionable endpoints that they have encountered the error condition themselves.) Examples might include:
  - An error in a switch or bridge between the partitionable endpoint and the PCI host bridge might cause multiple, or all, partitionable-endpoint domains in the PCI host bridge to enter the Stopped state. The key here is that none of these conditions can be overtly caused by a partition's software (for example, by a device driver). If the hardware cannot determine the source of the error, it must put all partitionable endpoints under the PCI host bridge into the Stopped state.
  - When the firmware requires temporarily suspending all operations under a PCI host bridge, to recover a PCI host bridge or an I/O fabric error, the firmware can place all the partitionable-endpoint domains in a PCI host bridge into the Stopped state. This makes it look to the device driver and operating system that the error is just for its I/O adapter.

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 PCI host bridge when any or all of the partitionable endpoints are in the Stopped state.

A.2.4. I/O Adapter Error-Injection Domains

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

A.2.5. Interrupts

A.2.5.1. Types of Interrupts

Two types of interrupts are supported for partitionable endpoints:

1. Level-signalled interrupt (LSI). This type of interrupt was defined by the original PCI architecture. The I/O adapter activates an LSI interrupt and does not deactivate the interrupt until told to do so by the device driver. The device driver must tell the I/O adapter 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 I/O adapter 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 I/O adapter 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 ensure that all outstanding interrupts have been processed after telling the I/O adapter 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 PCI host bridge to a total of four per root complex (that is, per PCI host bridge). That means that there can be at most four partitionable endpoints below a PCI host bridge that support LSI interrupts because of the interrupt sharing requirement.

2. Message-signalled interrupt (MSI). The I/O adapter 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 device driver before an EOI is issued to the interrupt controller. This is what is sometimes called an “edge triggered” interrupt. As with LSIs, the I/O adapter 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 device driver must assure that, after issuing an EOI to the interrupt controller, the I/O adapter 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, “Level-Signalled Interrupts” [44]).

A.2.5.2. Level-Signalled Interrupts

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

**A.2.5.3. Message-Signalled Interrupts**

MSI accesses are validated by comparing the partitionable-endpoint number of the device accessing the IVE to the partitionable-endpoint number in the IVE. Therefore, the I/O adapter cannot be set up accidentally or intentionally to access the MSI belonging to another partitionable endpoint. See also Section 3.2.4, “Message-Signalled Interrupt Design” [32].

**A.2.6. Partitionable-Endpoint Reset Domains**

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

For I/O adapter 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 partitionable endpoint is a single function.

**A.2.7. Partitionable-Endpoint Hot-Plug and Power Domains**

For partitionable endpoints that are hot pluggable, the hot-plug controller and all external bus isolation and power-control electronics must be provided. For partitionable endpoints 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 partitionable endpoint to get it into a known initialized state.
Appendix B. No-Translate Operation

Table B.1, “No-Translate Operation” [46] shows the no-translate operation for the I/O Design Architecture, version 3 (IODA3). IODA3 has 2 or n segments that are mappable for each partitionable endpoint (bimodal). For example, POWER8 implements n = 64 and has 2 or 64 segments that are mappable for each partitionable endpoint (bimodal). IODA3 also has separate address spaces for each partitionable endpoint.

Table B.1. No-Translate Operation

<table>
<thead>
<tr>
<th>Function</th>
<th>IODA3</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVE selection</td>
<td>The partitionable-endpoint number is obtained from the RTT lookup. This is concatenated with either PCI address 59 or 59:m, depending on the PCI host bridge configuration bit setting, to form the index into the TVE. For 32 bits, these high-order address bits are implied to be 0. For addresses below 4 GB, there is only one TVE for any given RID.</td>
<td>For example, POWER8 implements “m” = 55, so 1 bit or 5 bits (59:55) are used.</td>
</tr>
<tr>
<td>• 32-bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• 64-bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TVE selection validation</td>
<td>Not required because only the RID assigned to a TVE can access the TVE, since the partitionable-endpoint number for the RID is used as part of the index to select the TVE.</td>
<td></td>
</tr>
<tr>
<td>TVE validation of address</td>
<td>Uses the address range for the no-translate field of the TVE to validate the address.</td>
<td></td>
</tr>
<tr>
<td>No-translate address capability</td>
<td>A PCI host bridge 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. Only those TVEs selectable by the number of address bits concatenated with the partitionable-endpoint number are available. Architecturally, any number of address bits can be used, giving access to all the TVEs. However, there are system requirements to support more than one partitionable-endpoint number under a PCI host bridge for most applications with no-translate. Therefore, the implementations limit the number of address bits that get concatenated with the partitionable-endpoint number for TVE selection. Architecturally, the selection of 1 bit or “n” bits is bimodal per PCI host bridge, meaning one or the other for any particular PCI host bridge.</td>
<td>For example, for POWER8, the selection uses 1 bit or 5 bits (n = 5) and is bimodal per PCI host bridge. The PCI host bridge uses either 1 or 5 bits, configurable by a configuration bit. For POWER8 this gives: • Two TVEs per partitionable endpoint with 256 partitionable endpoints available • Thirty-two TVEs per partitionable endpoint with 16 partitionable endpoints available Architecturally, the Power Architecture Platform Requirements (PAPR) interfaces only allow no-translate mode for addresses above 4 GB. Addresses below 4 GB are allocated for address translation enabled. See Figure B.2, “Example Physical Address Map with TCE Bypass Enabled for Some Partitionable Endpoints” [49].</td>
</tr>
</tbody>
</table>

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. Implementations can append high-order bits to position the PCI host bridge within the system address space.

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.

Notes:

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, “TVE and Partitionable-Endpoint Number Determination” [47] shows graphically the TVE and partitionable-endpoint number determination.

Figure B.1. TVE and Partitionable-Endpoint Number 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 Partitionable Endpoints” [49].
Figure B.2. Example Physical Address Map with TCE Bypass Enabled for Some Partitionable Endpoints

PCI Address
Each partitionable endpoint has this full space
(maximum segments per PE = 2 or n; n being implementation dependent)

Remaining (unused) no-translate segments

4th no-translate segment

3rd no-translate segment

2nd no-translate segment

1st no-translate segment
Not used because used for TCE access

First 4 GB not accessible via no-translate mode
(No-translate mode only via 64-bit DMAs)

Base and Bounds must fall on a 16 MB boundary (or larger)

System Real Address

48
Max

Legend:

= An LMB size area

= LMBs that are not mapped into a no-translate area

= No memory

= Addresses not accessible via DMA
## Appendix C. Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIB</td>
<td>ASIC interconnect bus.</td>
</tr>
<tr>
<td>ARI</td>
<td>Alternate RID interpretation.</td>
</tr>
<tr>
<td>BAR</td>
<td>Base Address Register.</td>
</tr>
<tr>
<td>Bus/Dev/Func</td>
<td>Bus, device, and function. These are three fields in the PCI/PCI-X/PCIe domain that define an I/O adapter function. They come from the Bus (8 bits), Device (5 bits), and Function (3 bits) fields that define the configuration address for the I/O adapter 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 [51] and CID [50].</td>
</tr>
<tr>
<td>CFG</td>
<td>Configuration.</td>
</tr>
<tr>
<td>CID</td>
<td>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 [51] and Bus/Dev/Func [50].</td>
</tr>
<tr>
<td>DD</td>
<td>Device driver. Software that interfaces to and controls an I/O adapter.</td>
</tr>
<tr>
<td>DLPAR</td>
<td>Dynamic logical partitioning.</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct memory access.</td>
</tr>
<tr>
<td>DMA Stopped state</td>
<td>See Stopped state [52].</td>
</tr>
<tr>
<td>DR</td>
<td>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 Power Architecture Platform Requirements (PAPR) document for more information.</td>
</tr>
<tr>
<td>ECC</td>
<td>Error correction code.</td>
</tr>
<tr>
<td>ECR</td>
<td>End-to-end cyclic redundancy check.</td>
</tr>
<tr>
<td>EEH</td>
<td>Enhanced error handling option. See Section 2.2.4, “Enhanced Error Handling” [4].</td>
</tr>
<tr>
<td>EEH Stopped state</td>
<td>See Stopped state [52].</td>
</tr>
<tr>
<td>Endpoint partitioning</td>
<td>The concept of having a collection of independent domains (addressing, error state, and so on) that relate to a single I/O adapter (that is, a single endpoint). See Appendix A, Endpoint Partitioning [38]. See also PE [51].</td>
</tr>
<tr>
<td>EOI</td>
<td>End of interrupt.</td>
</tr>
<tr>
<td>FFI</td>
<td>Firmware force interrupt.</td>
</tr>
<tr>
<td>FIR</td>
<td>Fault Isolation Register.</td>
</tr>
<tr>
<td>FLR</td>
<td>Function level reset.</td>
</tr>
<tr>
<td>FMTC</td>
<td>Firmware-managed TCE coherency.</td>
</tr>
<tr>
<td>Host bridge</td>
<td>An entity that attaches an I/O bus to a system. A PCI host bridge is a specific host bridge for a PCI bus. See also PHB [51].</td>
</tr>
<tr>
<td>IOA</td>
<td>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.</td>
</tr>
<tr>
<td>IOA function</td>
<td>A single function, or specific function, of an IOA.</td>
</tr>
<tr>
<td>IODA3</td>
<td>I/O Design Architecture, version 3.</td>
</tr>
<tr>
<td>IOV</td>
<td>I/O virtualization. For more information, see the PCI-SIG I/O Virtualization (IOV) Specifications.</td>
</tr>
<tr>
<td>IVC</td>
<td>Interrupt vector cache.</td>
</tr>
<tr>
<td>IVE</td>
<td>Interrupt vector entry. See Table 3.10, “MSI IVE Definition” [35].</td>
</tr>
<tr>
<td>IVT</td>
<td>Interrupt vector table. A table of IVEs.</td>
</tr>
<tr>
<td>LEM</td>
<td>Local error macro.</td>
</tr>
<tr>
<td>LMB</td>
<td>Logical memory block.</td>
</tr>
<tr>
<td>LPAR</td>
<td>Logical partitioning. See the Power Architecture Platform Requirements (PAPR) document for details.</td>
</tr>
<tr>
<td>LSI</td>
<td>Level-signalled interrupt. An interrupt signaled by a packet on the PCIe bus.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Migration descriptor</td>
<td>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.</td>
</tr>
<tr>
<td>MMIO</td>
<td>Memory-mapped I/O. Refers to mapping the I/O bus address space (for example, I/O memory and I/O address spaces) into the Load/Store address space of the processor.</td>
</tr>
<tr>
<td>MMIO Stopped state</td>
<td>See Stopped state [52].</td>
</tr>
<tr>
<td>MSI</td>
<td>Message signalled interrupt. An interrupt that is signaled by a write to a particular address with specific data.</td>
</tr>
<tr>
<td>MSI-X</td>
<td>Message signalled interrupt - extended.</td>
</tr>
<tr>
<td>Page offset</td>
<td>The field in a PCI address used to index into a selected page of memory.</td>
</tr>
<tr>
<td>PAPR</td>
<td>Power Architecture Platform Requirements.</td>
</tr>
<tr>
<td>PCI</td>
<td>Peripheral Component Interface.</td>
</tr>
<tr>
<td>PCI-X</td>
<td>PCI Extended.</td>
</tr>
<tr>
<td>PCIe</td>
<td>PCI Express.</td>
</tr>
<tr>
<td>PE</td>
<td>Partitionable endpoint. The smallest entity that can be partitioned in endpoint partitioning. See Appendix A, Endpoint Partitioning [38].</td>
</tr>
<tr>
<td>PE#</td>
<td>Partitionable-endpoint number.</td>
</tr>
<tr>
<td>PELE-V</td>
<td>Partitionable-endpoint lookup entry (vector). The RTT associates a Bus/Device/Function number of an incoming PCI transaction to either a partitionable-endpoint number 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 partitionable-endpoint numbers are affected by the incoming RID.</td>
</tr>
<tr>
<td>PESE</td>
<td>Partitionable-endpoint state entry.</td>
</tr>
<tr>
<td>PF</td>
<td>The physical function of an IOV adapter. For more information, see the PCI-SIG I/O Virtualization (IOV) Specifications.</td>
</tr>
<tr>
<td>PHB</td>
<td>PCI host bridge. An entity that attaches a PCIe bus to the system.</td>
</tr>
<tr>
<td>PHB chip</td>
<td>The hardware chip where the PCI host bridge is implemented. The PCI host bridge might only be part of the chip functionality; for example, when the PCI host bridge is implemented on the processor chip. In that case, the processor chip becomes the PCI host bridge chip for purposes of this architecture.</td>
</tr>
<tr>
<td>PTE</td>
<td>Page table entry. Used for processor Load/Store address translation like the TVE and TCE are used for I/O address translation. Used to translate MMIO addresses as well.</td>
</tr>
<tr>
<td>RBA</td>
<td>Reject bit array.</td>
</tr>
<tr>
<td>RC</td>
<td>Root complex. Connects a PCIe bus into the system. PCIe host bridge is used in this document in place of the RC terminology.</td>
</tr>
<tr>
<td>RID</td>
<td>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 IODA3 to separate facilities in the PCI host bridge that are unique to the Func requesting the operation (for example, address translation, interrupt validation, and so on). See also Bus/Dev/Func [50] and CID [50].</td>
</tr>
<tr>
<td>Root complex</td>
<td>“An entity that includes a host bridge and one or more root ports.” (PCI-SIG. PCI Express Base Specification. 2003 )</td>
</tr>
<tr>
<td>Root port</td>
<td>“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 )</td>
</tr>
<tr>
<td>RPN</td>
<td>Real page number. The bits in the TCE that are used to replace the high-order I/O bus address bits.</td>
</tr>
<tr>
<td>RTAS</td>
<td>Run-Time Abstraction Services.</td>
</tr>
<tr>
<td>RTC</td>
<td>RID translation cache. An optional PCI host bridge implementation that allows for better PCI transaction performance when the RTT is in system memory. See also RTT [51].</td>
</tr>
<tr>
<td>RTE</td>
<td>RID translation entry. An entry in the RTT. See Table 3.1, “RTE Definition” [10].</td>
</tr>
<tr>
<td>RTT</td>
<td>RID translation table. A 64K-entry table that takes the 16-bit RID from a PCI transaction and maps that to a partitionable-endpoint number (for DMA and MSI operations) or to a PELE-V (when the operation is an error message). See also RTE [51].</td>
</tr>
</tbody>
</table>
Stopped state

More informally called the Enhanced-Error-Handling Freeze state, this state occurs after an I/O error. In this state, MMIO Stores to the affected I/O adapter are discarded. MMIO Loads are returned without error with data of all ones. DMA operations from the I/O adapter are aborted. This state prevents an I/O adapter, after an error, from causing any damage to the system. Therefore, the I/O adapter 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 I/O adapter 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 I/O adapter or partitionable endpoint 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 device driver might subsequently reset the MMIO Stopped state while leaving the DMA Stopped state set, to be able to query its I/O adapter 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 [52].

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” [23] 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 I/O adapter’s access to a DMA address space. See Table 3.5, “TVE Definition” [21] for the TVE definition.

TVT
Translation validation table (in the PCI host bridge). A table containing TVEs. See also TVE [52].

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.
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 resources fall into the following general categories:

- *Technical Steering Committee*
- *Foundation work groups*
- *OpenPOWER Ready documentation, products, and certification criteria*
- *Resource Catalog*

To find all OpenPOWER resources of the following types, select the specified combination of Resource Type/Main Category/Sub-category in the Resource Catalog:

Specifications | Developer Resources/OpenPOWER Documents/SPECIFICATIONS
Work Group Notes | Developer Resources/OpenPOWER Documents/Work Group Notes
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.