PCIe AMBA Integration Guide

Document number:	ARM DEN 0114
Release Quality:	ALP
Issue Number:	1.0
Confidentiality:	Non-Confidential
Date of Issue:	10/02/2022

Contents

About this document

Release Information

The change history table lists the changes that have been made to this document.

Date	Version	Confidentiality	Change
Feb 2022	1.0	Non-Confidential	First release of 1.0 ALP version – This is a comprehensive update of the 0.7 version dated 04/06/2013 (ARM-EPM-033524). Key changes: 1. Updated the requirements to be in accordance with the specifications listed in References section. 2. Updated terminology to be in accordance with the specifications listed in References section. 3. Added IDO and RO related recommendations and requirements. 4. Restructured the content to bring all Arm memory model related content into section 4 and all PCIe ordering model related content into section 5.

PCIe AMBA Integration Guide

copies of the Document in its possession. Upon termination of this Licence, all terms shall survive except for the licence grants.

Any breach of this Licence by a Subsidiary shall entitle Arm to terminate this Licence as if you were the party in breach. Any termination of this Licence shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This Licence may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this Licence and any translation, the terms of the English version of this Licence shall prevail.

The validity, construction and performance of this Licence shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: LES-PRE-21585 version 4.0

References

This document refers to the following documents.

Ref	Document Number	Title
1	ARM IHI 0022G (ID073019)	AMBA® AXI and ACE Protocol Specification
2	ARM DDI 0487E.a (ID070919)	Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile
3		PCI Express® Base Specification Revision 5.0 Version 1.0
4		PCI Local Bus Specification Revision 3.0

Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning

Potential for change

The contents of this specification are subject to change.

In particular, the following may change:

- Feature addition, modification, or removal
- Parameter addition, modification, or removal
- Numerical values, encodings, bit maps

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://infocenter.arm.com

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`.

In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Current status and anticipated changes

Alpha version.

Changes expected.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to errata@arm.com . Give:

- The title (PCIe AMBA integration guide).
- The number and issue (ARM DEN 0114 ALP 1.0).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 Introduction

1.1 Introduction

This document is intended to provide guidance on PCIe interface integration into an AMBA based System-On - Chip (SoC). It is assumed that the PCIe interface is connected to the rest of the SoC through an AXI or ACE protocol-based interconnect. The reader is assumed to be conversant with the PCIe, AMBA AXI and AMBA ACE protocols and Arm architecture.

The document covers the following topics:

- Description of Terminologies used in this document.
- Guidance on Arm memory type usage for PCIe transactions.
- How to comply with Arm Memory Model Requirements for PCIe transactions from Arm Processing Elements (PEs).
- How to comply with PCIe ordering model in an AMBA based system.
- Topology considerations while integrating PCIe interface into an AMBA based system.

2 Terminology

2.1 Transaction Terminology

A transaction is typically a read or write. The term transaction describes all aspects including the request, any data, and any responses.

In PCIe a transaction is made up of one or more packets. Each packet travels in one direction only.

In AMBA a transaction is made up of one or more transfers. Each transfer travels in one direction only.

An AMBA transfer is synonymous with a PCIe packet.

2.2 Arm Terminology

This document will use terminology as defined in the Arm Architecture Reference Manual [2] for describing the memory type and other memory attributes. This document does not attempt to give a full definition of all the memory types that can be used but gives an overview and describes the different terminology that is used. In keeping with the AMBA AXI and ACE focus of this document, the mapping of Arm memory types and memory attributes to AXI and ACE attributes is provided in sections 3.2 and 3.3.3.

Note: In this document, the term “AMBA interconnect” stands for an AXI or ACE protocol-based interconnect.

In Arm architecture, memory types can be broadly divided into Device memory types and Normal memory [see section B2.7 in [2]].

2.2.1 Device memory

Arm architecture defines the following attributes for Device memory:

Early Write Acknowledgment -	An early acknowledgement for a write transaction can be given before the transaction reaches the peripheral.
Reordering –	Transactions to the same peripheral can be re-ordered with respect to each other.
Gathering –	Multiple transactions can be merged into a single transaction.

Note: Speculative access to Device memory type is never permitted.

The Device memory types in ARMv8 are:

ARMv8 Memory Type	Definition

Device-nGnRnE	No Early Write Acknowledgement No Reordering No Gathering
Device-nGnRE	Early Write Acknowledgement No Reordering No Gathering
Device-nGRE	Early Write Acknowledgement Reordering No Gathering
Device-GRE	Early Write Acknowledgement Reordering Gathering

For accesses to Device memory, the table below shows the different terminology used in ARM v8 and AMBA.

ARMv8	AMBA AXI and ACE	Key characteristics of the Arm memory type.
Device-nGnRnE	Device Non-bufferable (AxCACHE = 0b0000). Same AXI ID. (See Note below)	Response comes from target peripheral. Transactions to the same peripheral must remain in order.
Device-nGnRE	Device Bufferable (AxCACHE = 0b0001) Same AXI ID. (See Note below)	Early response is permitted before the transaction has reached the peripheral. (Typically, this only relates to write transactions.) Transactions to the same peripheral must remain in order.
Device-nGRE	Device Bufferable (AxCACHE = 0b0001) Re-ordering is not explicit. It is implied by different AXI ID.	Early response is permitted. Transactions to the same peripheral can be re-ordered.
Device-GRE	Not supported. Will be treated as Device-nGRE.	Early response is permitted. Transactions to the same peripheral can be re-ordered. Multiple transactions to the same peripheral can be merged into a single transaction. Note: Since AXI/ACE does not support gathering, any merging must be done prior to the transaction entering the AXI/ACE interconnect.

Note: In AXI using the same AXI ID does not ensure ordering between read and write transactions. To ensure this ordering it is necessary to wait for a response to an early transaction before issuing the later transaction.

Note: Where it is stated that order is achieved using the same AXI ID, it is also possible to obtain order by ensuring that the later transaction is only issued once the earlier transaction has completed. The later transaction can then be issued using a different AXI ID.

2.2.2 Normal memory

Normal memory has two key attributes:

Shareability – Determines the agents/observers in a system that access the same address region.

Cacheability – Determines if it is permitted to allocate a memory location in a given cache and, if it can be allocated, whether the line is permitted to be dirty (Write-Back) or must be clean (Write-Through).

Shareability can be one of:

- Non-shareable
- Inner Shareable
- Outer Shareable

Cacheability can be one of:

- Normal Non-cacheable
- Normal Write-Through Cacheable
- Normal Write-Back Cacheable

Cacheability can be defined for both Inner and Outer caches. Inner caches are expected to be within a PE cluster and only the Outer cache memory type is exposed on an AMBA interface. The terms Inner and Outer when used for cacheability are different from the terms Inner and Outer when used for shareability – this is a common source of confusion.

For the purposes of this document Inner Cacheability is assumed to be the same as Outer Cacheability. No further use will be made of the terms Inner and Outer to refer to Cacheability and any reference to Inner and Outer later in this document refers to shareability.

For the purposes of this document it is assumed that Write-Through Cacheable is not supported. This is common for most Arm processor implementations.

The combination of Inner Shareable and Non-cacheable is not a supported combination. This is because Inner Shareable implies that the location can be held in the cache of another component, so the memory type (that must be compatible between components) cannot be non-cacheable.

If all observers (i.e. requesters) in an Outer Shareable domain are also in the same Inner Shareable domain, then there is no difference between Inner Shareable and Outer Shareable.

For the rest of this document, Inner Shareable is not considered due to the following reasons:

- The difference between Inner and Outer Shareable is deprecated in ACE5, ACE5-Lite, ACE5-LiteDVM, ACE5-LiteACP [see section F5.1 of [1]].
- The requirements for Inner Shareable and Outer Shareable are identical and the difference between Inner and Outer Shareable is only in the number of agents that need to be snooped.
- It is expected that for majority of systems, the Inner and Outer Shareability domains will have the same set of observers. In such cases, the software visible behaviour would be the same regardless of whether shareability attribute is programmed as Inner Shareable or Outer Shareable.

Note: In some AMBA documents, the word “Snoopable” is used instead of “Shareable”. In this document, the word "Shareable” is used throughout.

The table below shows the supported combinations of cacheability and shareability that need to be considered.

Cacheability	Shareability
Non-cacheable	Outer Shareable
Write-Back Cacheable	Non-Shareable
Write-Back Cacheable	Inner Shareable
Write-Back Cacheable	Outer Shareable

The mapping of permitted combinations of Arm memory type, cacheability and shareability attribute to AMBA AXI attributes [see table A4-5 in section A4.4 of [1]] and ACE attributes [see table D3-3 in section D3.1.1 of [1]] is given in section 3.3 for Inbound transactions and in section 3.2 for Outbound transactions.

2.3 PCIe Terminology

The PCIe specification [3] defines the following terminology:

Term	Definition
Downstream	1.The relative position that is farther from the Root Complex. 2.A direction of information flow where the information is flowing away from the Root Complex.
Upstream	1.The relative position that is closer to the Root Complex. 2. A direction of information flow where the information is flowing towards the Root Complex.
Transmitting Port	The Port that transmits the Packet on a given Link.
Receiving Port	The Port that receives the Packet on a given Link.
Egress Port	The Transmitting Port: that is, the Port that sends outgoing traffic.
Ingress Port	The Receiving Port: that is, the Port that accepts incoming traffic.
Completer	The PCIe function that completes a request (usually the destination of the request).
Relaxed Ordering (RO) and ID based ordering (IDO)	These are packet attribute fields defined in the PCIe specification. The value of these attributes influences the order in which requests and completions arrive at the destination. Refer to the PCIe specification for details [3].

Process Address Space ID (PASID).	This is an ID that used to identify the address space of a transaction. Refer to the PCIe specification for details [3].
Fabric	A PCIe fabric is composed of point-to-point Links that interconnect a set of components (Endpoints, Switches and Root Complex) [section 1.3 of [3]].

This document will use the following terminology to describe the different aspects of a transaction.

- PCIe Interface is used to describe the IP block that interfaces between an on-chip AMBA system and a PCIe Link. A PCIe Interface will have both AMBA Manager (M) and Subordinate (S) ports. A PCIe Interface could be either a Root Complex or an Endpoint in the PCIe domain.
- Root-SoC is used to describe the SoC that contains the PCIe Root Complex.
- Endpoint-SoC is used to describe an SoC that contains a PCIe Endpoint.

Figure 1 below illustrates the PCIe interface in Root-SoC and Endpoint-SoC

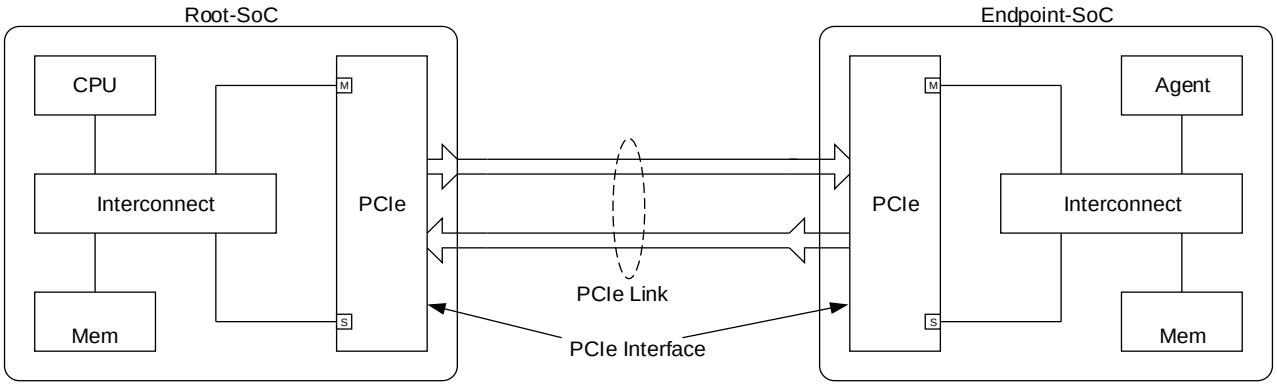


Figure 1 PCIe Interface in Root-SoC and Endpoint-SoC

For this document, PCIe switches are not considered.

2.3.1 Inbound and Outbound

The terms Outbound and Inbound are used to describe a transaction.

An Outbound transaction has an Outbound Request and (optionally) an Outbound Completion, which returns in the opposite direction. Note that in the case of a completion “Outbound” refers to the direction of the original transaction request, not the direction of the completion.

Likewise, an Inbound transaction has an Inbound Request and (optionally) an Inbound Completion, which returns in the opposite direction. Note that in the case of a completion “Inbound” refers to the direction of the original transaction request, not the direction of the completion.

For a Root-SoC, containing the PCIe Root Complex:

An Outbound Request travels downstream to the Endpoint.

An Outbound Completion returns upstream to the Root Complex.

An Inbound Request travels upstream to the Root Complex.

An Inbound Completion returns downstream to the Endpoint.

For an Endpoint-SoC:

An Outbound Request travels upstream to the Root Complex.

An Outbound Completion returns downstream to the Endpoint.

An Inbound Request travels downstream to the Endpoint.

An Inbound Completion returns upstream to the Root Complex.

Note: Care is required when using the terms “upstream” and “downstream”. For example, “arrives from upstream” is the same as “travels downstream”.

The Figure 2 below shows Inbound and Outbound traffic for a Root-SoC.

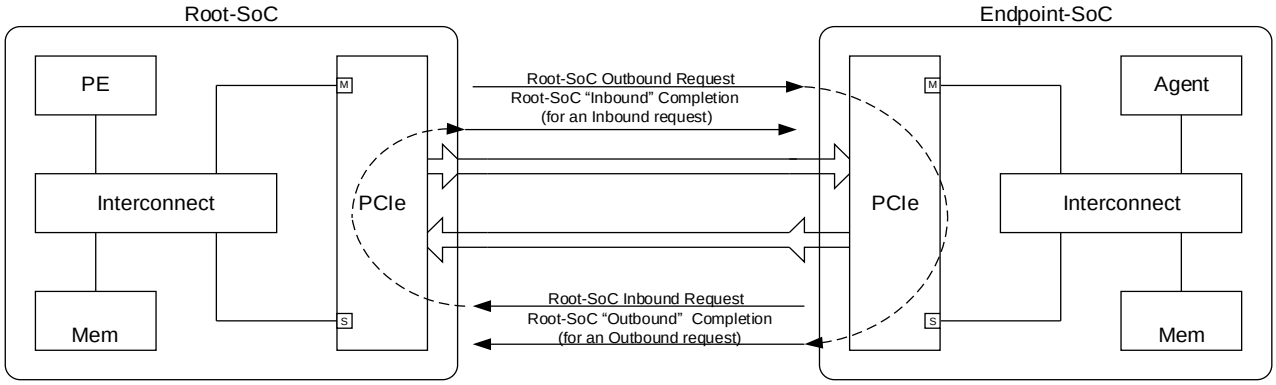


Figure 2 Inbound and Outbound traffic for a Root-SoC

All the examples given in this document are described from the perspective of the Root-SoC.

3 ARM Memory Type USAGE FOR INBOUND AND OUTBOUND PCIE transactions

3.1 Memory Type Assumptions

The following assumptions are made about the memory type for all transactions.

- The cacheability of the memory type is the same for all levels of cache. The allocation hints may differ, but the architectural requirements of the memory type are the same.
- Write-Through Cacheable is not supported.

3.2 Outbound Transaction Memory Type

All Outbound transactions destined for PCIe will have one of the following Arm memory type attributes:

- Device-nGnRnE
- Device-nGnRE
- Device-nGRE
- Device-GRE
- Normal Non-cacheable

Use of any cacheable memory type is ***not*** expected for Outbound transactions. Cache maintenance would be required to ensure visibility of transactions.

Use of cacheable memory is expected for Inbound transaction, see section 3.3 for further details.

Table below shows the mapping of Arm memory type to AMBA AXI and ACE memory attributes for Outbound transactions.

Arm memory type and cacheability attribute of the Outbound transaction	**Outbound Transaction AXI Memory attributes (AxCACHE)**	**Outbound Transaction ACE memory attributes (AxCACHE) and shareability domain (AxDOMAIN)**
Device-nGnRnE	Device Non-bufferable	Device Non-bufferable, System
Device-nGnRE or Device-nGRE or Device-GRE	Device Bufferable	Device Bufferable, System
Normal, Non-cacheable	Normal Non-cacheable Bufferable	Normal Non-cacheable Bufferable, System

3.3 Inbound Transaction Memory Type

Inbound transactions from the PCIe interface must have appropriate AXI or ACE memory attributes. It is expected that these attributes are derived in a three-step process as follows:

- Step 1: The Arm memory type, Cacheability and Shareability attributes of the Inbound transaction is determined.
- Step 2: If required, the Cacheability attribute determined in step 1 is overridden based on the transaction's No Snoop bit value.
- Step 3: The Arm memory type and Shareability attribute from step 1 and the cacheability attribute from step 2 is used to determine the AXI or ACE memory attributes of the Inbound transaction.

3.3.1 Step 1: Determining the Arm memory type, cacheability and shareability attributes

For deriving the Arm memory type, cacheability and shareability, the following configurations need to be considered:

1. The PCIe interface has an associated System Memory Management Unit (MMU) and the System MMU is enabled to provide translations:

In this case, the System MMU must be accessed to get the Arm memory type, cacheability and shareability attributes of the transaction.

2. PCIe interface does not have an associated System MMU or the System MMU is bypassed or disabled:

In this configuration, the PCIe interface must be able to determine the Arm memory type, cacheability and shareability attributes of the location an Inbound transaction is targeting. Typically, this can be done using a primitive address decode to associate specific address ranges with specific Arm memory type, cacheability and shareability attributes. It is expected that this association is configurable to allow maximum flexibility.

It is expected that the Arm memory type for a location that is in a peripheral is determined based on the properties of the location, as defined in the Arm Architecture reference manual [see section B2.7.1 and B2.7.2 of [2]].

It is strongly recommended that DRAM memory is mapped as Normal memory with Write-Back as the cacheability attribute.

Note: The constraint to be considered while setting the memory type of a location is that the weakest memory type that can be used is determined by the properties of the location. It is permitted to use a stronger memory type although it might not be performance optimal.

3.3.2 Step 2: Modifying the cacheability attribute based on the transaction's No Snoop bit value

The "No Snoop" bit is a transaction attribute provided by PCIe specification for a requester to indicate whether hardware coherency management is expected or not [see section 2.2.6.5 and 7.5.3.4 of [3]].

For an SoC with software visible caches, the cacheability attribute of the transaction must be forced to Non-cacheable if "No Snoop" bit for the transaction is 1 and the Arm memory type (determined in step 1) is Normal.

For an SoC where all caches are software invisible, it is strongly recommended that cacheability attribute of the transaction is set to Non-cacheable if "No Snoop" bit for the transaction is 1 and the Arm memory type (determined in step 1) is Normal.

In both cases, if the "No Snoop" bit is 0, then the cacheability attribute from step 1 is retained.

3.3.3 Step 3: Mapping the Arm memory type, cacheability and shareability attributes to AXI/ACE memory attributes.

In this step, Arm memory type and shareability attributes from Step 1 and the cacheability attribute from step 2 is mapped to AXI or ACE memory attributes prior to issuing into the AMBA interconnect.

The table below provides the mapping information:

- AMBA AXI memory attribute [see table A4-5 in section A4.4 of [1]] for the transaction given its final Arm memory type, cacheability attribute, shareability attribute.
- AMBA ACE memory attributes [see table D3-3 in section D3.1.1 of [1]] for the transaction given its final Arm memory type, cacheability attribute, shareability attribute.

Note: As explained in section 2.2.2, this section only considers Outer Shareable.

Inbound PCIe transaction: Arm memory type, cacheability and shareability attributes (after Step 2)	**PCIe Transaction AXI Memory attribute (excluding the allocate attributes)**	**PCIe Transaction ACE memory attribute (excluding the allocate attributes) and shareability domain**
Device-nGnRnE	Device Non-bufferable	Device Non-bufferable, System
Device-nGnRE or Device-nGRE or Device-GRE	Device Bufferable	Device Bufferable, System
Normal, Write-Back, Outer Shareable	WriteBack	WriteBack, Outer Shareable
Normal, Non-cacheable, Outer Shareable	Normal Non-cacheable, Bufferable	Normal Non-cacheable Bufferable, System

3.3.4 Coherency management for Inbound transactions

For an Inbound PCIe transaction, the table below provides the following information:

- For a given location, possible combinations of the Inbound PCIe transaction attributes (after step 2 – see section 3.3.2) and PE side attributes for exchanging data between Software running on a PE and a PCIe Requester.
- Whether or not software cache maintenance is required to maintain coherency.
- The AXI and ACE attributes corresponding to the Inbound PCIe transaction attributes.
    - Note: As explained in section 2.2.2, this section only considers Outer Shareable.

PE transaction: Arm memory type, cacheability and shareability attributes	Inbound PCIe transaction: Arm memory type, cacheability and shareability attributes (after step 2 -i.e. after considering No Snoop attribute)	Coherency Management	PCIe Transaction AXI Memory attribute (excluding the allocate attributes)	PCIe Transaction ACE memory attribute (excluding the allocate attributes) and shareability domain
Normal, Write-Back, Outer Shareable	Normal, Write-Back, Outer Shareable	Hardware managed.	WriteBack	WriteBack, Outer Shareable
Normal, Write-Back, Non-shareable	Normal, Write-Back, Outer Shareable	Software cache maintenance (This assumes that PE transaction looks up the cache in which PCIe transaction might be cached).	WriteBack	WriteBack, Outer Shareable
Normal, Non-cacheable, Outer Shareable	Normal, Write-Back, Outer Shareable	Not permitted unless all caches in the system are invisible to software. With software invisible caches, hardware would ensure coherence.	WriteBack	WriteBack, Outer Shareable
Normal, Write-Back, Outer Shareable	Normal, Non-cacheable, Outer Shareable	Software cache maintenance unless all caches in the system are invisible to software. With software invisible caches, hardware would ensure coherence.	Normal Non-cacheable, Bufferable	Normal Non-cacheable Bufferable, System
Normal, Write-Back, Non-shareable	Normal, Non-cacheable, Outer Shareable	Software cache maintenance	Normal Non-cacheable, Bufferable	Normal Non-cacheable Bufferable, System
Normal, Non-cacheable, Outer Shareable	Normal, Non-cacheable, Outer Shareable	Not cached.	Normal Non-cacheable, Bufferable	Normal Non-cacheable Bufferable, System

For a location which is mapped as one of the Device memory types for both the PE transaction and the PCIe Inbound transaction, there is no hardware or software cache maintenance required as Device memory type locations are never cached. The Device memory type assigned to the location on the PE side can be different from the Device memory type assigned to the location on the PCIe Inbound side. However, software (or a combination of software and firmware) is responsible for ensuring that both PE transaction and PCIe Inbound transaction uses a Device memory type that matches with the properties of the target location.

It is expected that software (or the combination of software and firmware) will not map a location as Normal memory for the PE transaction and map the same location as Device memory for the PCIe Inbound transaction. Similarly, it is expected that software (or the combination of software and firmware) will not map a location as Device memory for the PE transaction and map the same location as Normal memory for the PCIe Inbound transaction. If software (or the software-firmware combination) behaves contrary to these expectations, then it is software's (or the software-firmware combination's) responsibility to manage the errors that can happen due to such a mapping.

Note: With software invisible caches, the assumption is that hardware would ensure coherence by accessing the caches regardless of the cacheability and shareability attributes.

4 COMPLYING TO ARM memory model FOR PE GENERATED PCIE TRANSACTIONs

PE generated PCIe transactions are Outbound transactions as shown in Figure 3 for a Root-SoC and as shown in Figure 4 for an Endpoint-SoC.

This section describes the following:

- Arm memory type mapping options for PCIe address spaces.
- Requirements that the PCIe interface, the SoC interconnect and the PE must meet to preserve Arm memory model's guarantees.
- End to end ordering guarantees provided by the system for a PE request targeting a PCIe completer.

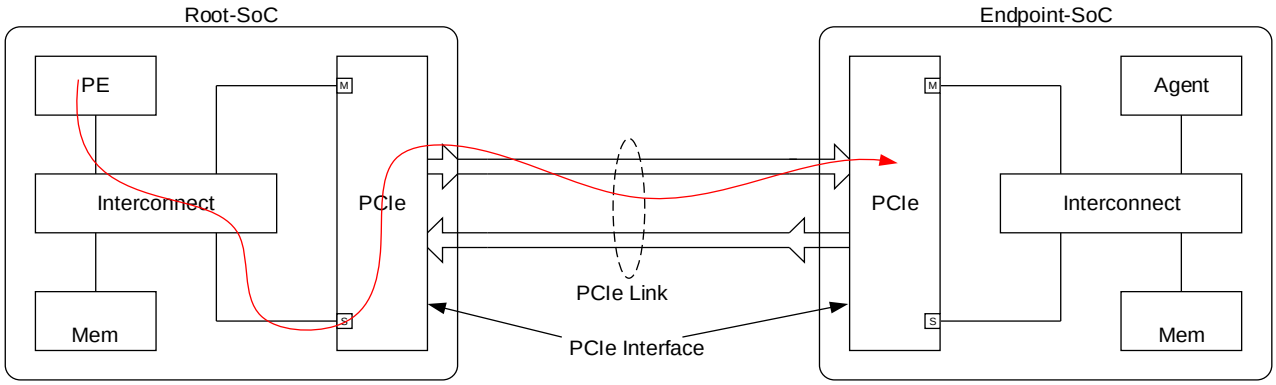


Figure 3 PE generated PCIe transaction in a Root-SoC

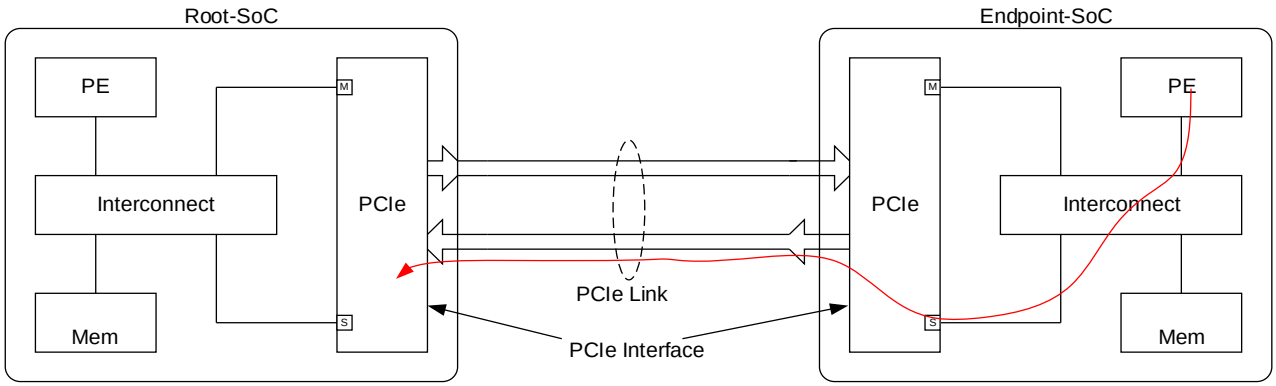


Figure 4 PE generated PCIe transaction in an Endpoint-SoC

From a PE perspective, all memory locations in a PCIe completer will be mapped in the PE's translation tables as one of the following:

- Device-nGnRnE.
- Device-nGnRE.
- Device-nGRE.
- Device-GRE.
- Normal with Non-cacheable as the cacheability attribute.

Sections 4.1,4.2 and 4.3 details the requirements to be met for preserving the guarantees given to software for each Arm memory type.

4.1 Device-nGnRnE and Device-nGnRE

For Device-nGnRnE and Device-nGnRE memory types, the Endpoint arrival order must match program order and read speculation is not permitted.

The nGnRnE memory type can be used to target the configuration, prefetchable and non-prefetchable PCIe address spaces.

The nGnRE memory type can be used to target the prefetchable and non-prefetchable PCIe address spaces.

Note: Writes to prefetchable and non-prefetchable memory space of a PCIe completer are posted- i.e. the completer will not send a response back to the requester. Despite the lack of an explicit response back from the completer, the Arm architecture permits mapping the prefetchable and non-prefetchable memory spaces as Device-nGnRnE [see section E.2.7.2 of [2]]. There are no restrictions on mapping the prefetchable and non-prefetchable memory spaces as Device-nGnRE.

4.1.1 PE and Interconnect requirements for handling Outbound requests to Device-nGnRnE or Device-nGnRE mapped locations

For ensuring that Device-nGnRnE and Device-nGnRE memory type works correctly, the PE and the interconnect needs to meet the following requirements:

- Transactions to a location mapped as Device-nGnRnE must have “Device Non-bufferable” as the AXI memory type. For ACE, the memory type is “Device Non-bufferable” with “System” as the shareability domain.
- Transactions to a location mapped as Device-nGnRE must have “Device Bufferable” or “Device Non-bufferable” as the AXI memory type. For ACE, the memory type is “Device Bufferable” or “Device Non-bufferable” with “System” as the shareability domain.
- If there is a read request outstanding to a peripheral, then any new read request to the same peripheral must use the same AXI ID as that of the outstanding read.
- If there is a write request outstanding to a peripheral, then any new write request to the same peripheral must use the same AXI ID as that of the outstanding write.
- Determination of which peripheral region an Outbound PE request falls into is done by implementation defined methods. If such a determination is not possible, then it must be assumed that all Device Bufferable and Device Non-bufferable transactions from a PE are to the same peripheral region and must obey the AXI ID requirements stated above.
- AMBA interconnect does not guarantee any order between read and write transactions. Therefore, to preserve program order between reads and writes, the PE needs to wait for completion of previously issued read transactions to a given peripheral before issuing write transactions to the same peripheral and vice versa.

4.1.2 PCIe interface requirements for handling Outbound requests to Device-nGnRnE or Device-nGnRE mapped locations

For the PCIe interface, requests to a Device-nGnRnE mapped location would be received with “Device Non-bufferable" attribute from the AMBA interconnect. Requests to a Device-nGnRE mapped location would be received with “Device Bufferable” attribute from the AMBA interconnect.

On top of being compliant with the PCIe ordering model (see section 5 for the rules) and the AMBA specification [1], the PCIe interface needs to meet the following requirements:

- Read transactions can get re-ordered with respect to program order while traversing the PCIe fabric. Therefore, the PCIe interface must send out a Device Non-bufferable or Device Bufferable read only after completion has been received for earlier read transactions with the same AXI ID targeting the same peripheral region [for definition of peripheral region see section A6.2 of [1]].
- Configuration writes can get re-ordered with respect to program order while traversing the PCIe fabric. Therefore, the PCIe interface must send out a Device Non-bufferable configuration write only after completion has been received for earlier configuration writes with the same AXI ID targeting the same peripheral region.
- Posted writes with Device Bufferable or Device Non-bufferable attribute targeting the same peripheral region (i.e. having the same AXI ID) must not be allowed to overtake each other while transiting through the PCIe interface to the PCIe link.
    - Note that this applies even if one write has Device Bufferable attribute and the second write has Device Non-bufferable attribute.
- For configuration writes, the PCIe interface must return the write completion response to the requester only after completion from the PCIe completer has been received.
- RO must not be set for transactions with the Device Bufferable or Device Non-bufferable attribute.
- IDO attribute is not set for any transaction from a PE or if IDO is set for transactions from a PE, then all requests from that PE has the same PCIe Requester ID and same PASID.
    - It is expected that PASID will be set only for Outbound transactions from an Endpoint-SoC.
    - It is expected that all transactions from a Root-SoC PE would have the same PCIe Requester ID.

4.1.3 Example use cases for Device-nGnRnE and Device-nGnRE mapping of PCIe address spaces

The most important use case for Device-nGnRnE mapping is configuration writes. This is illustrated by the following example:

If we imagine a case where software writes to a configuration register and then reads a prefetchable BAR space location whose value depends on the configuration register, then it is important that the software does the read only after it knows that the configuration write has reached the target and is complete. The only way for software to know that a configuration write is complete is to make sure that the configuration write uses the Device-nGnRnE attribute and have a DSB between the configuration write and the read.

Secondly, the Device-nGnRnE and Device-nGnRE types provide the ability to pipeline transactions to PCIe targets even if those transactions need to arrive in program order at the target. This illustrated by the following examples:

- Two read transactions, the first of which is a read from the receive buffer and the second of which is a read from the buffer status register, are required to remain in program order to give the correct behaviour. If the transactions are re-ordered on the PCIe fabric then it would be possible to read a status register which indicates that the receive buffer still contains entries when, in reality, the buffer will be emptied by the read from the receive buffer. In this scenario, if both transactions have the Device-nGnRnE or Device-nGnRE attribute, then they will arrive in program order at the target as desired even if they are pipelined.
- A similar situation exists for two Outbound configuration write requests. An example would be software needing to write to Device Control register to set the Max Payload size of write requests by a Device prior to doing a second configuration write to set the Bus Master Enable bit in the Device's Command register. In this case, it is necessary that the first configuration write arrives at the Device before the second configuration write for correct operation. In this scenario, if both transactions have the Device-nGnRnE attribute, then they will arrive in order program order at the target as desired even if they are pipelined.

4.2 Device-nGRE and Device-GRE

For Device-nGRE and Device-GRE memory types, read speculation is not permitted. However, it is permitted for the arrival order of transactions at the destination peripheral to be different from program order except for transactions targeting the same location (see section 4.4).

Any gathering permitted by the Device-GRE memory type must occur prior to entering the AXI or ACE interconnect. This is because AXI and ACE does not support signalling of gathering via attributes.

Device-nGRE memory type can be used to target non-prefetchable and prefetchable PCIe address spaces.

Device-GRE memory type can be used to target prefetchable PCIe address space.

On AXI interconnect, transactions targeting locations mapped with these types will have "Device Bufferable" as the memory type. For ACE, the memory type is "Device Bufferable" and the shareability domain is "System".

The only restriction on usage of AXI IDs is that transactions to the same address must have the same AXI ID if multiple such transactions are outstanding concurrently. The rules for handling two transactions to the same address is given in 4.4.

4.2.1 PCIe interface requirements for handling Outbound requests to Device-nGRE and Device-GRE mapped locations

Requests to a Device-nGRE or Device-GRE mapped location would be received with "Device Bufferable" attribute from the AMBA interconnect by the PCIe interface.

On top of being compliant with the PCIe ordering model (see section 5 for the rules) and the AMBA specification [1], the requirements in section 4.4, 4.5 and 4.6 apply.

Note: It is a software error to map configuration space as Device-nGRE or Device-GRE (see section 4.8, note 1). With such a mapping, program order arrival of configuration accesses will not be guaranteed by the system. If configuration space is mapped as Device-nGRE or Device-GRE, the PCIe interface is not expected to send out configuration reads or configuration writes one at a time unless they have the same AXI ID.

4.3 Normal with Non-cacheable as the cacheability attribute

For the Normal memory type, read speculation is permitted and the Endpoint arrival order of transactions can be different from program order except for transactions targeting the same location (see section 4.4). It is expected that this memory type is only used to target an address space that is prefetchable.

Transactions targeting Normal memory type locations with Non-Cacheable attribute will have “Normal Non-cacheable Bufferable” as the AXI memory type. For ACE, the memory type is “Normal Non-cacheable Bufferable” and the Shareability domain is “System”.

The only restriction on usage of AXI IDs is that transactions to the same address must have the same AXI ID if multiple such transactions are outstanding concurrently. The rules for handling two transactions to the same address is given in 4.4.

4.3.1 PCIe interface requirements for handling Outbound requests to Normal Non-cacheable mapped memory locations

Requests to a Normal memory type location with the Non-cacheable attribute would be received with “Normal Non-cacheable Bufferable” attribute from the AMBA interconnect by the PCIe interface.

On top of being compliant with the PCIe ordering model (see section 5 for the rules) and the AMBA specification [1], the requirements in section 4.4,4.5 and 4.6 apply.

Note: It is a software error to map configuration space as Normal Non-cacheable (see section 4.8, note 1). With such a mapping, program order arrival of configuration accesses will not be guaranteed by the system. If configuration space is mapped as Normal Non-cacheable, the PCIe interface is not expected to send out configuration reads or configuration writes one at a time unless they have the same AXI ID.

Note: If software maps non-prefetchable space as Normal Non-cacheable, then it is the responsibility of software to deal with the consequences of speculative accesses to non-prefetchable space. The PCIe interface is not expected to flag an error if a request with “Normal Non-cacheable Bufferable” attribute is targeting the non-prefetchable space. It is expected that the PCIe interface would forward such requests to the completer in the same way as it would forward a “Normal Non-cacheable Bufferable” request targeting the prefetchable space.

4.4 Complying with internal visibility requirement of the Arm memory model

Internal visibility requirement of the Arm memory model [see section B2.3.3 of [2]] requires that all accesses from a PE to the same location happens in program order.

To comply with this requirement, the PE and the interconnect must comply with the following requirements:

- If there is a write request outstanding from a PE to a given location, then any new write request to that location from the same PE must use the same AXI ID as that of the outstanding write.
- If there is a read request outstanding from a PE to a given location, then any new read request to that location from the same PE must use the same AXI ID as that of the outstanding read.
- If there is a write request outstanding from a PE to a given location, then any new read request to that location from the same PE must not be issued until the response for the outstanding write request has been received.

- If there is a read request outstanding from a PE to a given location, then any new write request to that location from the same PE must not be issued until the response for the outstanding read request has been received.

4.4.1 PCIe interface requirements for complying with internal visibility requirement for Outbound transactions from PEs

The PCIe interface must comply with the following requirements:

- For ensuring that multiple reads from the same location happens in program order:

  In this case, the PCIe interface will receive all such reads with the same AXI ID from the AMBA interconnect. Therefore, the rules in section 4.5 applies.

- For ensuring that program order is maintained for multiple writes to the same location:

  In this case, the PCIe interface will receive all such writes with the same AXI ID from the AMBA interconnect. Therefore, the rules in section 4.5 applies.

- For ensuring that program order is maintained for a read after write sequence to the same location:
  - IDO attribute is not set for any transaction from a requester (e.g. from a PE) or if IDO is set for transactions from a requester, then all requests from that requester has the same PCIe Requester ID and same PASID.
    - Note: It is expected that PASID will be set only for Outbound transactions from an Endpoint-SoC.
    - Note: It is expected that all transactions from a Root-SoC PE would have the same PCIe Requester ID.

  If program order is maintained for a read after write sequence for the same location, then the PE can issue a read transaction before a posted write transaction to the same location has reached its endpoint, as long as the PE has received the write completion response from the PCIe Interface.

- For ensuring that program order is maintained for a write after read sequence to the same location:

  The PCIe interface does not need to do anything for ensuring that program order is maintained for a write after read sequence to the same location. The PE must wait till the earlier read has completed before issuing the write.

4.5 PCIe interface requirements for handling transactions with the same AXI ID

- Read transactions can get re-ordered with respect to program order while traversing the PCIe fabric. Therefore, an AMBA read transaction must be sent out on PCIe only after completion has been received from the PCIe side for earlier AMBA reads with the same AXI ID.
- Writes with the same AXI ID must not be allowed to overtake each other while transiting through the PCIe interface to the PCIe link.
- RO must not be set for write requests.
- IDO attribute is not set for any transaction from a requester (e.g. from a PE) or if IDO is set for transactions from a requester, then all requests from that requester has the same PCIe Requester ID and same PASID.
  - Note: It is expected that PASID will be set only for Outbound transactions from an Endpoint-SoC.
  - Note: It is expected that all transactions from a Root-SoC PE would have the same PCIe Requester ID.

4.6 Interconnect requirements for preserving barrier-ordered-before ordering relation

The PCIe interface effectively contains a PoS (Point of Serialization) as it controls the ordering of transactions by not giving a write transaction response until the write transaction is ordered with respect to later transactions.

Therefore, an interconnect must not provide an early write response for transactions that are destined for a PCIe Interface. The write response for writes targeting PCIe completers must always come from the PCIe interface.

A PE must not consider pre-barrier write and read transactions to be ordered until a completion response has been received from the PCIe interface.

4.7 Setting RO and IDO for PE transactions

4.7.1 Setting RO for Root-SoC PE transactions

It is strongly recommended that RO=0 for PE transactions.

This is because setting RO=1 for PE transactions can make the system non-compliant with the Arm memory model in the following ways:

- Writes to a Device-nGnRnE/Device-nGnRE region in a PCIe device may not arrive in program order.
- Multiple writes to the same location in a PCIe device may not arrive in program order (see section 4.4.1 for more details).
- Barrier enforced order of writes to memory in a PCIe device may not be preserved.
- Inter-thread ordering set up by software between writes to memory in a PCIe device may not be preserved.

Correct operation with RO=1 can only be achieved by avoiding the need for these ordering guarantees. This would require customized software or a restricted usage model. Therefore, the use of RO=1 is incompatible with standard software.

4.7.2 Setting IDO for Root-SoC PE transactions

It is strongly recommended that IDO=0 for transactions generated by an agent within the Root-SoC.

This is because setting IDO for Root-SoC generated transaction is discouraged by the PCIe specification itself [see section E.5.2 and E.4.2 of [3]].

Note: If the PCIe interface assigns the same PCIe Requester ID for transactions from all PEs, setting IDO to 1 will not break the Arm memory model compliance of the system.

4.7.3 Setting RO for Endpoint-SoC PE transactions

It is strongly recommended that RO=0 for PE transactions if the Endpoint-SoC is running off the shelf OSes and applications.

If RO needs to be 1 for PE transactions, then software must be customized, or usage model must be restricted to ensure correct execution.

This is because setting RO=1 for PE transactions can make the system non-compliant with the Arm memory model in the following ways:

- Writes to a Device-nGnRnE/Device-nGnRE region in a PCIe completer may not arrive in program order.
- Multiple writes to the same location in a PCIe completer may not arrive in program order.
- Barrier enforced order of writes to memory in a PCIe completer may not be preserved.
- Inter-thread ordering set up by software between writes to memory in a PCIe completer may not be preserved.

Correct execution with RO set to 1 can only be achieved by avoiding the need for these ordering guarantees. This would require customized software or a restricted usage model. Therefore, the use of RO=1 is incompatible with standard software.

4.7.4 Setting IDO for Endpoint-SoC PE transactions

IDO can be set to 1 for PE transactions while remaining compliant with the Arm memory model if the following conditions are met:

1. The same PCIe Requester ID and PASID is used for all transactions from a PE.

   Meeting this condition ensures that the following Arm memory model guarantees are available:

   - Program order is preserved for multiple accesses to the same location in a PCIe completer (see section 4.4.1 for more details).
   - Accesses to a Device-nGnRnE or Device-nGnRE mapped region in a PCIe completer will arrive in program order.
   - Barrier ordered accesses to a PCIe completer will arrive in the specified order.

2. If the memory on a PCIe completer is shared between two different PEs, then accesses to that memory from both the PEs must use the same PASID and PCIe Requester ID.

   Meeting this condition ensures that the inter-thread order set up by software between accesses to the shared memory is preserved. If IDO is set to 1 with condition 1 or 2 not met, correct execution can be achieved only by avoiding the need for these ordering guarantees. This requires customized software or a restricted usage model.

4.8 Ordering guarantees available to software for accesses targeting PCIe destinations

A PE generated transaction targeting a PCIe completer (an example is shown in Figure 5 below) will have ordering rules from both Arm memory model and PCIe applied to it as it flows from source to destination. This section details the end to end guarantees that the system can provide for such a transaction.

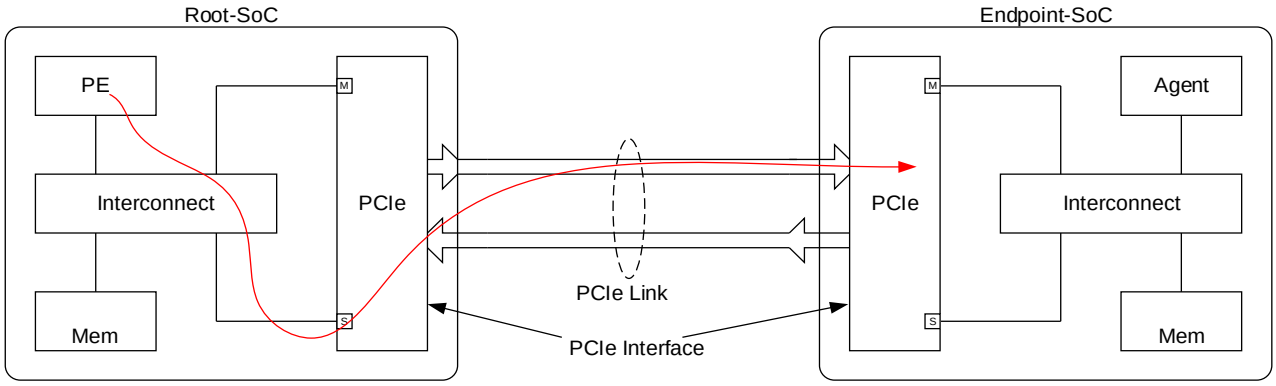


Figure 5 PE transaction targeting PCIe completer

Table below shows the end to end ordering guarantee provided by the system for a PE generated PCIe transaction.

The table makes the following assumptions:

- Software is executing on a single PE that uses the Arm memory model.
- The PE is connected via AMBA interconnect to the PCIe interface.
- All accesses are to a specific PCIe address space in a specific PCIe target.
- RO and IDO attributes are not set to 1 for transactions from the PE.

The table is read in the following way:

For each row, two accesses RW1 and RW2 from the same PE to the PCIe address space given in column 1 (e.g. configuration space), when mapped to the Arm memory type(s) given in column 2 (e.g. Device-nGnRnE) will arrive at the destination as per the ordering guarantee given in column 3 (In the example case of configuration space mapped to Device-nGnRnE, RW1 and RW2 will arrive in program order)

PCIe address space	**Arm memory type to which the PCIe address space is mapped**	**Destination arrival order guarantee provided by system to software**
Configuration space	Device-nGnRnE (See note 1 below)	All accesses from the PE will arrive in program order.
Non-prefetchable memory space	Device-nGnRE or Device- nGnRnE (See note 3 below)	All accesses from the PE will arrive in program order (See note 2 below).
	Device-nGRE	No arrival order guarantees provided by system except for accesses to the same location (See note 4 below).
Prefetchable memory space	Device-nGnRE or Device- nGnRnE (See note 3 below)	All accesses from the PE will arrive in program order (See note 2 below).

	Normal Non-cacheable or Device-GRE or Device-nGRE	No arrival order guarantees provided by system except for accesses to the same location (See note 4, 5 and note 6 below).

Notes:

1. PCIe specification states that there must be a way for software to know that a configuration write is complete at the target [see implementation note in section 7.2.2 of [3]]. This can be achieved only by using the Device-nGnRnE memory type. Therefore, the only memory type that can be used for mapping configuration space is Device-nGnRnE.
2. Program order arrival coupled with the PCIe interface enforcing PCIe ordering rules ensures that producer consumer ordering requirements are met [see section E of [4] for more details on producer consumer ordering model]. Therefore, if software requires producer consumer ordering without using barrier instructions or other ordering techniques, it must use Device-nGnRE or Device-nGnRnE.
3. Writes to non-prefetchable memory space and prefetchable memory space are posted. This means that the PCIe interface will give a write completion response to the requester without getting a completion back from the target. Therefore, even if the software maps the prefetchable/non-prefetchable space as Device-nGnRnE, the completion will not come from the destination of the request. Consequently, DSB is not enough to determine that a write has reached the target for prefetchable and non-prefetchable memory spaces.
4. All accesses from a PE to the same location must arrive in program order as detailed in section 4.4.
5. PCIe specification permits devices to expose memory regions with read side effects as part of its prefetchable address space [see implementation note titled “Additional Guidance on the Prefetchable Bit in Memory Space BARs” in section 7.5.1.2.1 in [3]]. In such cases, software must ensure that prefetchable space with read side effects is not mapped as Normal Non-cacheable so that any unwanted side effect due to read speculation is prevented.
6. PCIe specification permits devices to expose memory regions that cannot tolerate byte merging for writes as part of its prefetchable address space [see implementation note titled “Additional Guidance on the Prefetchable Bit in Memory Space BARs” in section 7.5.1.2.1 in [3]]. Software must ensure that prefetchable space that cannot tolerate byte merging is not mapped as Device-GRE or Normal Non-cacheable.

4.8.1 Achieving producer consumer ordering for transactions from PE to PCIe destinations

As mentioned in section 4.8, the system can guarantee PCIe Producer / Consumer Ordering for Outbound PE generated PCIe transactions only if the target location(s) are mapped as Device-nGnRnE or Device-nGnRE. All other Arm memory types would require barrier instructions or other instruction ordering techniques to be used to enforce the necessary arrival order at the destination.

4.8.2 Multiple PCIe address spaces mapped as Device-nGnRnE or Device-nGnRE

Arm memory model does not give any ordering guarantees between accesses to different Device-nGnRnE or Device-nGnRE peripherals. Additionally, there is no restriction on mapping various PCIe address spaces of the same PCIe function as different Device-nGnRnE or Device-nGnRE peripherals. Consequently, software cannot assume that program order will be maintained between accesses to two different PCIe address spaces, even though both spaces are mapped as Device-nGnRnE or Device-nGnRE. Therefore, for maximum software portability, ordering requirements between accesses to different PCIe address spaces must be handled explicitly in software using appropriate ordering instructions.

5 COMPLYING TO PCIE ORDERING MODEL

PCIe ordering requirements are specified in the form of an ordering rules table in the PCIe specification [see section 2.4 in [3]]. The primary purpose of the PCIe ordering model is to achieve producer-consumer ordering and to avoid deadlock. This section details the requirements that the PCIe interface in a Root-SoC or Endpoint-SoC must meet for complying with PCIe ordering model.

The organization of rest of this section is as follows:

In section 5.1, the rules in the table are discussed in brief and the coverage of the rules in this document is listed.

In section 5.2, the table is broken down in terms of the rules for Ordering and Overtaking. Ordering rules describe when requests and completions must be kept in the order in which they were received by the PCIe interface to ensure that producer consumer ordering model is not violated. The Overtaking rules describe when certain requests or completions must be able to pass other requests or completions that are blocked to avoid deadlock.

In section 5.3, the requirements for complying with the overtaking rules identified in section 5.2 are discussed in detail.

In section 5.4, the requirements for complying with the ordering rules identified in section 5.2 are discussed in detail.

Note: Unless explicitly stated, a given requirement must be met by the PCIe interface in both Root-SoC and Endpoint-SoC.

5.1 Ordering table coverage

PCIe ordering rules are given in the table below [as per section 2.4 in [3]].

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

Entries A2b, B2b, C2b and D2b do not need to be considered for functional correctness. These entries all permit one request to pass another when certain conditions (such as RO and/or IDO permit). The system is always functionally correct if the ability for one request to pass another is ignored, but it may be lower performance. Once it has been established that a system is completely functionally correct, the relaxations permitted by these entries can be used to improve performance. This document covers the requirements for correct implementation of functional correctness rules (A2a, B2a, C2a, D2a) and performance enhancement rules (A2b, B2b, C2b, D2b).

Note: It is worth noting that the relaxations permitted by these entries are not symmetric.

Posted requests do not overtake posted requests - Except for RO or IDO. (Rule A2b)

Read requests do not overtake posted requests - Except for IDO. (Rule B2b)

Non-posted request do not overtake posted requests - Except for RO or IDO. (Rule C2b)

Read completions do not overtake posted requests - Except for RO or IDO. (Rule D2b)

Note that RO relaxation does not apply to “Read requests do not overtake posted requests”.

Entry D2b also permits a configuration write completion to pass a posted request in all cases.

Entry A5b relates only to PCI/PCI-X bridges. Therefore, this rule is not relevant and is not covered in this document.

Entry D5b requires that completions with the same Transaction ID must not pass. It does not add extra requirements between the completions of different transactions. Therefore, this rule is not discussed further in this document.

All other entries that are “Y/N” in the table need not be considered for functional correctness with respect to the PCIe ordering model. Hence, they do not impose any additional PCIe ordering related requirements on the PCIe interface and are not discussed further in this section.

The table below summarize the rules coverage in this document:

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

All entries within the green boxes are covered in this document.

5.2 Overtaking and Ordering

There are two ways to categorize the entries in the table. Firstly, whether they are ordering rules, where order must be maintained, or overtaking rules, where one transaction must be able to pass another to avoid deadlock if the earlier transaction is blocked. This categorization is shown in the table below. Note that the performance optimization related rules (A2b, B2b, C2b, D2b) are not shown in the table.

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

Ordering *Overtaking*

For a PCIe interface, the overtaking and ordering rules must be applied for the following:

- Requests and completions that are coming into the PCIe interface from the link.
- Requests and completions that the PCIe interface is sending out.

Outbound and Inbound Traffic on the PCIe link for the PCIe interface in a Root-SoC is shown in Figure 6 and that of an Endpoint-SoC is shown in Figure 7.

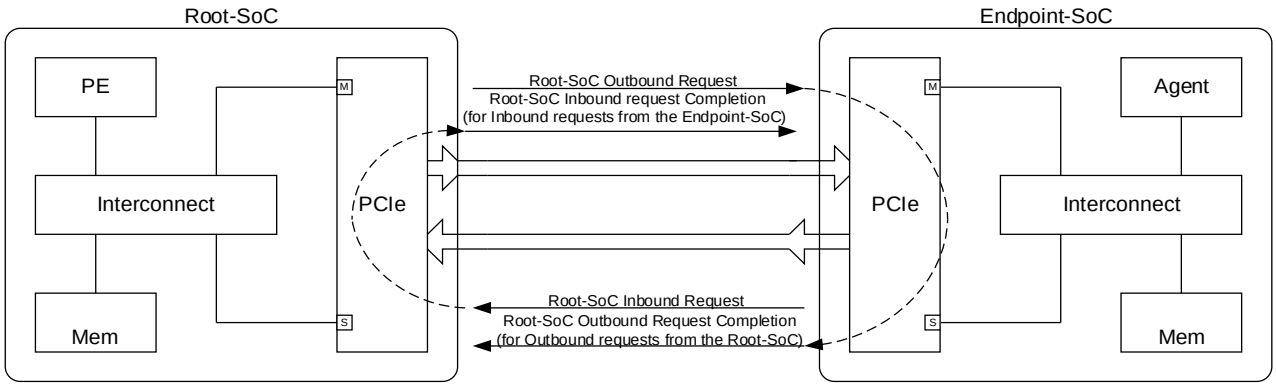


Figure 6 Inbound and Outbound traffic on the PCIe link for a Root-SoC

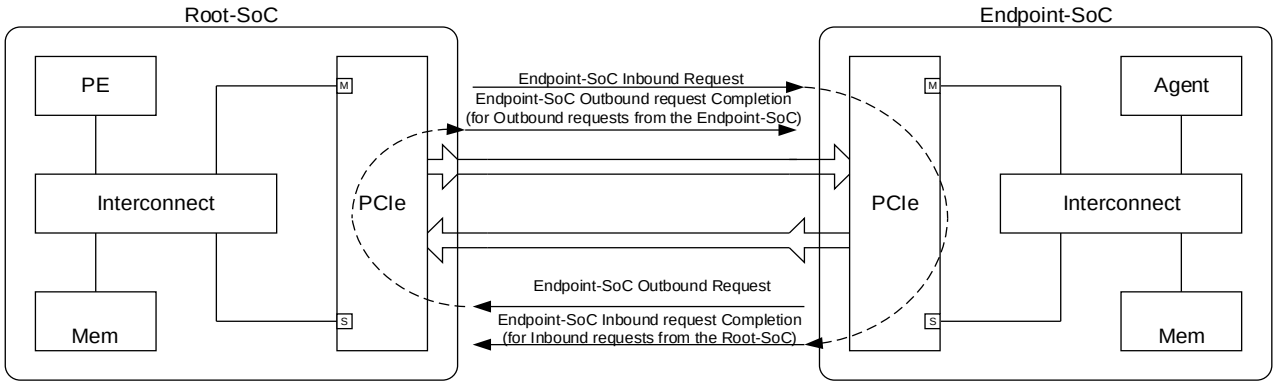


Figure 7 Inbound and Outbound traffic on the PCIe link for an Endpoint-SoC

Table below highlights the rules that apply between two requests and rules that apply between a completion and a request.

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

Rules applicable between requests

Rules applicable between requests and completions

From this table, it is evident that the overtaking and ordering requirements must be considered for the following cases:

- Both requests Outbound
- Both requests Inbound
- Outbound request / Inbound completion
- Outbound completion/ Inbound request

For Overtaking rules, “Both requests Outbound” and “Both requests Inbound” consider the overtaking rules of multiple requests, as described in table cells A3 and A4. While “Outbound request / Inbound completion” and

“Outbound completion/ Inbound request” consider the overtaking rules between requests and completions, as described in table cells D3 and D4.

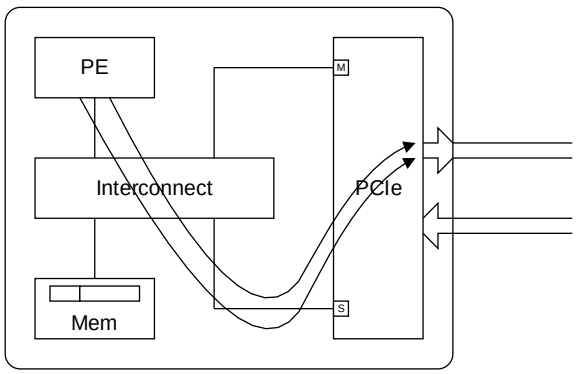
For Ordering rules, “Both requests Outbound” and “Both requests Inbound” consider the ordering rules of multiple requests, as described in table cells A2, B2 and C2. While “Outbound request / Inbound completion” and “Outbound completion/ Inbound request” consider the ordering rules between requests and completions, as described in table cell D2.

5.3 Overtaking Rules

This section considers the overtaking rules. These rules are all based on the principle that any packet which does not require a packet to flow in the opposite direction should make forward progress. So, read transaction completions (which are the last stage of a read transaction) are required to make forward progress and posted write transactions (which only require packets in one direction) are required to make forward progress.

5.3.1 Both requests Outbound

This describes the overtaking rules that apply for two Outbound requests being received by the PCIe Interface Subordinate port.



Posted Write must be able to overtake Read Request [A3]

A PCIe Interface must ensure that an Outbound posted write received from the AMBA interconnect is able to overtake blocked Outbound read requests even though the read requests were received earlier than the posted write.

For peer-to-peer traffic in a Root-SoC, it is required that there is a non-blocking path for Inbound posted writes coming into one PCIe interface to go out as Outbound posted writes in another PCIe interface through the AMBA interconnect. This non-blocking path must not use any shared read/write resource, to avoid read backpressure blocking the posted writes.

Posted Write must be able to overtake Configuration Write [A4]

A PCIe Interface must ensure that an Outbound posted write received from the AMBA interconnect is able to overtake blocked Outbound configuration write requests even though the configuration write requests were received earlier than the posted write.

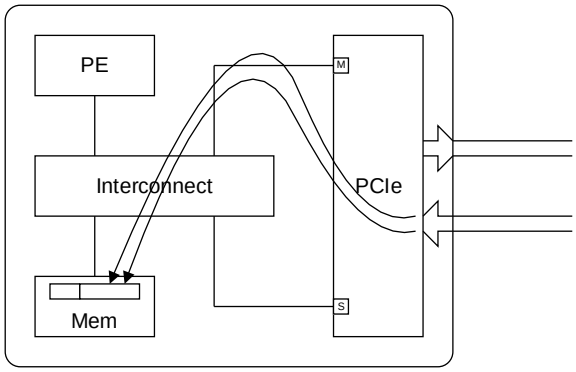
It is strongly recommended that a PCIe Interface can accept at least one configuration write, whilst still allowing subsequent posted writes to be accepted and be able to overtake the configuration write. To implement this requirement consideration needs to be given to configuration writes and posted writes that use the same AXI ID, as the AXI write responses must be given in the same order that the requests are received.

The AMBA system must ensure there is no back pressure on Inbound requests. It must ensure that any back pressure on configuration writes or posted writes does not block the progress of snoop transactions for Inbound requests. See section 6.2 for further details.

In a system where it is possible for multiple configuration writes to happen concurrently, the PCIe Interface must be capable of accepting all configuration writes that are presented to it, whilst still allowing subsequent posted writes to be accepted and to overtake the configuration writes.

5.3.2 Both requests Inbound

This describes the overtaking requirements that apply for two Inbound requests being issued by the PCIe Interface Manager port.



Posted Write must be able to overtake Read Request [A3]

A PCIe Interface must ensure that an Inbound posted write request is able to overtake Inbound read requests.

This requires that the PCIe Interface can issue a write request into the AMBA interconnect even if read requests that were received earlier are yet to be issued into the AMBA interconnect. This ensures that even if the read channel of the AMBA interconnect is blocked, Inbound posted writes can make forward progress.

Note that the PCIe interface must guarantee that it can always accept the data returned by the AMBA interconnect for a read request it has made. In practice, this means that the PCIe interface must not issue a read request into the AMBA interconnect unless it has buffer space reserved for the read data.

For peer-to-peer traffic in a Root-SoC, the same requirement for a non-blocking path from Inbound posted writes to Outbound posted writes in the AMBA system exists, as described above.

Posted Write must be able to overtake Configuration Write [A4]

This rule has different implications in a Root-SoC PCIe interface versus in an Endpoint-SoC PCIe interface.

For the Root-SoC PCIe interface

Inbound configuration writes are not supported for Root-SoC devices. This also means that there are no implications for peer-to-peer traffic.

For the Endpoint-SoC PCIe interface

An Inbound configuration write to an Endpoint-SoC is generally not expected to propagate past the PCIe Interface.

However, if the configuration write does propagate past the PCIe interface, then the PCIe Interface must ensure that an Inbound configuration write cannot block an Inbound posted write request.

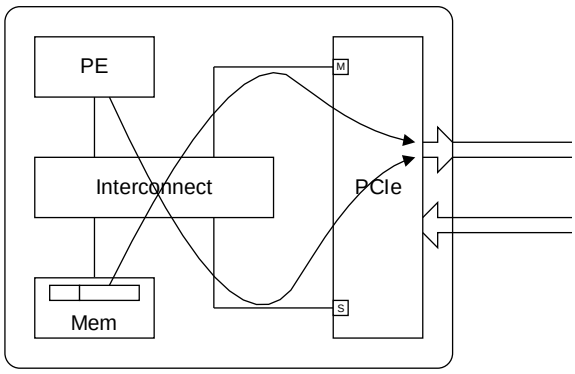
This requires that the PCIe Interface must be able to perform the following:

- The PCIe Interface can issue a write request into the AMBA interconnect even if Configuration write requests that were received earlier are yet to be issued into the AMBA interconnect.
- The PCIe interface can issue a later posted write request before the earlier configuration write request completion is returned on the PCIe link.

For the case where Inbound configuration write is sunk within the PCIe interface, the PCIe interface must ensure that Inbound posted write can be issued into the AMBA interconnect even if there are Inbound Configuration writes received earlier waiting to be sunk by the PCIe interface.

5.3.3 Outbound request / Inbound Completion

This describes the overtaking requirements of two packets, one is an Outbound request that is being received by the PCIe Interface Subordinate port and the other is an Inbound completion returning via the PCIe Interface Manager port.



Inbound completion must be able to overtake Outbound Read Request or Configuration Write. [D3, D4]

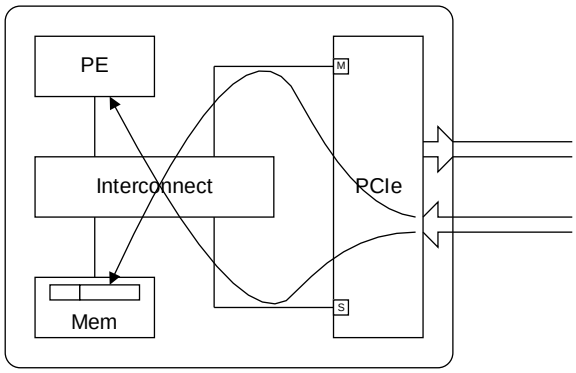
A PCIe Interface must ensure that completions for Inbound read requests (termed as “Inbound Completions”) are not blocked by Outbound read requests or configuration write requests. This means that the PCIe interface can schedule completions for Inbound read requests ahead of Outbound requests and Outbound configuration write requests that are blocked (e.g. due to lack of appropriate flow control credits).

PCIe Interface must also ensure that the need for Inbound completions to not overtake Outbound posted writes, does not prevent Inbound completions from overtaking Outbound read requests or configuration write requests.

There is no implication in the AXI or ACE interconnect domain, as read data (which eventually becomes the completion for the Inbound read request) is always returned without requiring forward progress on requests flowing in the same direction.

5.3.4 Outbound completion/ Inbound request

This describes the overtaking requirements of two packets, one is an Inbound request being issued by the PCIe Interface Manager port and the other is an Outbound completion (for an Outbound request transaction) returning via the PCIe Interface Subordinate port.



Outbound Completion must be able to overtake Inbound Read Request or Configuration Write. [D3, D4]

A PCIe Interface must ensure that completions for Outbound read requests and Outbound configuration write requests (termed as “Outbound completion”) are not blocked by Inbound read requests or Configuration write requests. This means that the PCIe interface is able to let Outbound completions reach the AMBA interconnect ahead of Inbound read requests and Inbound configuration write requests which are blocked from being sent into the AMBA interconnect (e.g. read requests being blocked due to the AMBA read channel being not ready).

PCIe Interface must also ensure that the need for Outbound completions to not overtake Inbound posted writes, does not prevent Outbound completions from overtaking Inbound read requests or configuration write requests.

There is no implication in the AXI or ACE interconnect domain, as read data or write response from an Outbound completion can flow through the interconnect without requiring forward progress of requests flowing in the same direction.

5.4 Ordering Rules

This section considers the ordering rules required to maintain the PCIe producer-consumer ordering model.

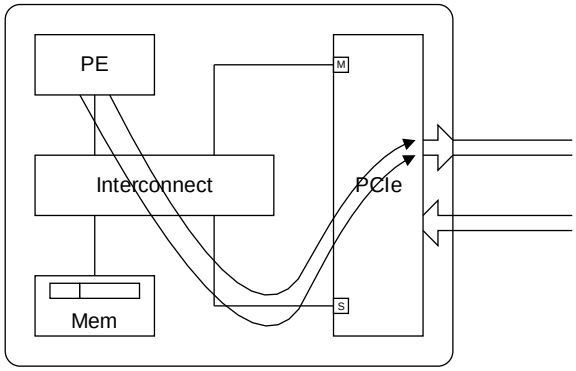
Each sub-section starts with a diagrammatic overview of the case being considered, followed by a summary of the requirements and then finally a use case example is given.

The use cases examples are intended to clarify the reasoning behind certain ordering requirements, but they do not form part of the requirements or attempt to describe every aspect that needs to be considered.

Note: In the many of the use case examples the Endpoint is considered as being a simple UART like device, with register access to features such as a write-only transmit buffer, a read-only receive buffer, a writeable register to force the transmit buffer to flush and a readable buffer status register to indicate the number of entries in both transmit/receive buffers. Whilst this simple example is not realistic, it is useful to illustrate why the ordering of two accesses to registers at different memory location may require to be ordered.

5.4.1 Both requests Outbound

This describes the ordering requirements of two packets that are both Outbound requests being received by the PCIe Interface Subordinate port.



5.4.2 Posted Write, Read Request, or Configuration Write must not overtake Posted Write [A2, B2, C2]

For complying with this ordering requirement, the PCIe interface requirements are as follows:

For Outbound posted writes with RO=0 and IDO=0, Outbound Read requests with IDO=0 and Outbound Configuration writes with IDO=0 [A2a, B2a, C2a in the PCIe ordering table]:

- Outbound read requests, posted write requests and configuration write requests must not be allowed to overtake an earlier Outbound posted write for which a completion response has been returned to the AMBA interconnect.

For posted writes with (RO=1 and IDO=0) or (RO=1 and IDO=1) [A2b in the PCIe ordering table]:

- Outbound posted writes with RO=1 can overtake earlier Outbound posted writes for which a completion response has been returned to the AMBA interconnect. Note that value of IDO need not be considered when RO=1.

For posted writes with RO=0 and IDO=1 [A2b in the PCIe ordering table]:

- Outbound posted writes with RO=0 and IDO=1 can overtake earlier Outbound posted writes for which a completion response has been returned to the AMBA interconnect provided either of the following are true:
    - Requester ID of the two writes are different
    - Both writes have PASID prefixes (relevant only for Endpoint-SoC), and the PASID values are different.

For Read requests with IDO=1 [B2b in the PCIe ordering table]:

- Outbound Read requests with IDO=1 can overtake earlier Outbound posted writes for which a completion response has been returned to the AMBA interconnect provided either of the following is true:
    - Requester IDs of write and the read are different.
    - Both write request and read request has PASID prefix (relevant only for Endpoint-SoC), and the PASID values are different.

For Configuration write requests with RO=1 or IDO=1 [C2b in the PCIe ordering table]:

- Outbound Configuration write requests with RO=0 and IDO=1 can overtake earlier Outbound posted writes for which a completion response has been returned to the AMBA interconnect provided the following is true:
    - Requester IDs of the posted write and configuration write are different.
- Outbound configuration write requests with (RO=1 and IDO=0) or (RO=1 and IDO=1) can overtake earlier Outbound posted writes for which a completion response has been returned to the AMBA interconnect. Note that value of IDO need not be considered when RO=1.

5.4.3 Example Use Cases

Usage model example for rule A2a

An example usage that requires multiple Outbound posted writes to be kept in the order in which the requester intended them to reach the completer would be a series of writes to data buffer followed by a write to valid flag location which indicates that the data is available. If ordering was not maintained, then it would be possible for the valid flag to be observed by the endpoint before all the data buffer writes had occurred.

Usage model example for rule B2a

An example usage model that requires Outbound reads to be prevented from overtaking Outbound posted writes is the following sequence:

1. Endpoint A sends a peer to peer write to Endpoint B via the Root-SoC.
2. Endpoint A writes a flag in Root-SoC memory indicating that the data is ready in Endpoint B's buffer.
3. Software reads the flag as set and sends a read to get the data from Endpoint B.
4. Both the software read from Endpoint B and Endpoint A write to Endpoint B goes out through the same PCIe interface.
5. This means that if software read of the buffer in Endpoint B overtakes Endpoint A's write to the same buffer, then software would get stale data. Therefore, it is necessary that Outbound reads do not overtake Outbound posted writes in this usage model.

Usage model example for rule C2a

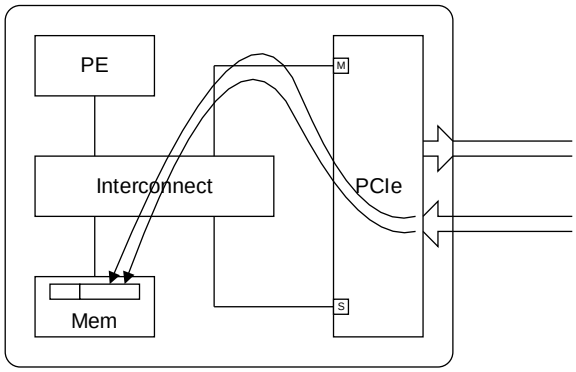
An example usage model that requires Outbound configuration writes to be prevented from overtaking Outbound posted write is the following sequence:

1. Software writes data to Endpoint A's memory space using posted writes.
2. Software writes to a configuration register in Endpoint A to trigger usage of the data written in (1).

3. This means that if configuration write overtakes posted writes, then stale data would be used. Therefore, it is necessary that Outbound configuration writes do not overtake Outbound posted writes in this usage model.

5.4.4 Both requests Inbound

This describes the ordering requirements of two packets that are both Inbound requests being issued by the PCIe Interface Manager port.



`Posted Write must not overtake Posted Write [A2]`

A PCIe Interface must maintain write ordering required for the PCIe ordering model by the appropriate use of AXI IDs for write requests and the serialization of write requests.

The exact ordering requirements of between two Inbound write requests will depend on the RO and IDO attributes of the requests as detailed below:

Inbound posted write with IDO=0 and RO=0 [Rule A2a in the PCIe ordering table]

In this case, all observers in the system must observe Inbound posted writes in the order in which they were received from the link by the PCIe interface. To guarantee the required ordering of an Inbound posted write with Inbound posted writes that arrived earlier, the PCIe interface can do one of the following:

- It issues the posted write request into the AMBA interconnect only after a completion response has been received for all earlier write requests.
- It issues posted writes into the AMBA interconnect in a pipelined manner by doing one of the following:
  - It uses the Ordered_Write_Observation property. The Ordered_Write_Observation property defines that the interface will ensure that all requests with the same AXI ID will be observed in the order in which they were issued into AMBA interconnect by the PCIe interface. Typically, this is supported by the interconnect scheduling the requests and using its knowledge of the system address map to ensure two requests are not issued in a pipelined manner if they are to different peripherals.
  - It takes advantage of the fact that all transactions to the same peripheral with the same AXI ID will be observed in the order in which they were issued into AMBA interconnect by the PCIe interface. This is done when the interface knows that all outstanding writes (i.e. previously issued writes for which completion response has not been received from the AMBA interconnect yet) are to the same peripheral. Note that when issuing a request to a different peripheral it is necessary to wait till all earlier write requests have received their completion response.

To take advantage of these two methods, a posted write with IDO=0 and RO=0 must be issued with the AXI ID of outstanding writes (i.e. previously issued writes for which completion response has not been received from the AMBA interconnect yet) that had IDO=0 and RO=0.

Note that posted write with a certain AXI ID can be issued into the *AMBA* interconnect only after completion response has been received for all previously issued writes with a different AXI ID.

Inbound posted write with IDO=1 and RO=0 [Rule A2b in the PCIe ordering table]

For Inbound posted writes with IDO=1 and RO=0, the PCIe interface can do one of the following:

- It issues posted writes into the AMBA interconnect in a pipelined manner by doing one of the following:
  - It uses the Ordered_Write_Observation property.
  - It takes advantage of the fact that all transactions to the same peripheral with the same AXI ID will be observed in the order in which they were issued into AMBA interconnect.

  To take advantage of these two methods, a posted write with a certain PCIe Requester ID or PASID must be issued with the AXI ID of outstanding writes that had the same PCIe Requester ID or PASID.

  Each PCIe Requester ID and each PASID is expected have its own unique AXI ID for posted writes. This ensures that all posted writes with same PCIe Requester ID or the same PASID remains in the order in which they were received from the link while allowing re-ordering between posted writes having different PCIe Requester IDs or different PASIDs.

  Note that a posted write having IDO=1 and RO=0 can be issued into the AMBA interconnect with a certain AXI ID only if the following conditions are true:
  - All previously issued writes that had the same PCIe Requester ID but a different AXI ID has received their completion response.
  - If the posted write has PASID prefix, then all previously issued writes that had the same PASID but a different AXI ID has received their completion response.
- It ignores the IDO=1 setting and uses one of the options listed in the section titled *Inbound posted write with IDO=0 and RO=0 [Rule A2a in the PCIe ordering table]* while issuing the posted write into the AMBA interconnect.

Inbound posted write with IDO=0 and RO=1 [Rule A2b in the PCIe ordering table]

For Inbound posted writes with IDO=0 and RO=1, the PCIe interface can do one of the following:

- Use different AXI ID for each posted write that has RO=1. This ensures that posted writes with RO=1 can overtake earlier Inbound posted writes.
- It ignores the RO=1 setting and uses one of the options listed in the section titled *Inbound posted write with IDO=0 and RO=0 [Rule A2a in the PCIe ordering table]* while issuing the posted write into the AMBA interconnect.

Inbound posted write with IDO=1 and RO=1 [Rule A2b in the PCIe ordering table]

For Inbound posted writes with IDO=1 and RO=1, the PCIe interface can do one of the following:

- Use different AXI ID for each posted write that has RO=1. This ensures that posted writes with RO=1 can overtake earlier Inbound posted writes.
- It ignores the RO=1 setting and does performance optimization based on IDO=1 as listed in the section titled *Inbound posted write with IDO=1 and RO=0 [Rule A2b in the PCIe ordering table]*.

- It ignores both RO=1 and IDO=1 settings and uses one of the options listed in the section titled *Inbound posted write with IDO=0 and RO=0 [Rule A2a in the PCIe ordering table]* while issuing the posted write into the AMBA interconnect.

Read Request must not overtake Posted Write [B2]

Inbound read with IDO=0 [Rule B2a in the PCIe ordering table]

A PCIe Interface must ensure an Inbound read request is issued into the AMBA interconnect only after all earlier Inbound posted writes have received their completion response.

Inbound read with IDO=1 [Rule B2b in the PCIe ordering table]

In this case, the Inbound read request can be issued into the AMBA interconnect without waiting for earlier writes to receive their completion responses if the Requester ID of the Inbound read is different from that of the writes.

Configuration Write must not overtake Posted Write [C2]

This rule has different implications in a Root-SoC PCIe interface versus in an Endpoint-SoC PCIe interface.

Inbound configuration write with IDO=0 and RO=0 [Rule C2a in the PCIe ordering table]

For the Root-SoC:

The ordering of Inbound configuration write request only applies to an Endpoint-SoC. Inbound configuration writes are not supported for a Root-SoC. Therefore, this rule is not relevant for the Root-SoC.

For the Endpoint-SoC:

An Inbound configuration write to an Endpoint-SoC is generally not expected to propagate past the PCIe Interface. However, if this is supported then the PCIe Interface in an Endpoint-SoC must ensure that the configuration write does not overtake the posted write. This can be achieved with the use of techniques detailed in the section titled *Inbound posted write with IDO=0 and RO=0 [Rule A2a in the PCIe ordering table].*

Inbound configuration write with IDO=1 or RO=1 or both IDO=1 and RO=1 [Rule C2b in the PCIe ordering table]

For the Root-SoC:

The ordering of Inbound configuration write requests only applies to an Endpoint-SoC. Inbound configuration writes are not supported for a Root-SoC. Therefore, this rule is not relevant for the Root-SoC.

For the Endpoint-SoC:

As mentioned previously, an Inbound configuration write to an Endpoint-SoC is generally not expected to propagate past the PCIe Interface. However, if the configuration can propagate past the PCIe interface into the AMBA interconnect, then the following rules apply:

If RO=1, the PCIe interface can allow the Inbound configuration write to overtake the Inbound posted write.

If IDO=1 the PCIe interface can allow the Inbound configuration write to overtake the Inbound posted write if one of the following is true:

- Requester IDs of posted write and the configuration write are different.
- Both posted write and the configuration write has PASID prefix and the PASID values are different.

Example Use Cases

For Inbound Requests to a Root-SoC, only posted writes and reads need to be considered.

Inbound Read after Inbound Read

No special consideration is required for read after read requests. There is no ordering guarantee provided by the PCIe link between read requests and therefore no benefit in establishing order when moving from the PCIe link to an AMBA system. If ordering was required between two Inbound reads then the requester (which is on the other side of the PCIe link) must not send the second read on the PCIe link till completion for the first read has been received.

Inbound Write after Inbound Write

This corresponds to rule A2a in the PCIe ordering rules table. PCIe Producer / Consumer ordering model requires that multiple posted writes are observed in the order in which the writes were received from the link by the PCIe interface.

For a Root-SoC, this would be required if an endpoint was writing a data buffer followed by a write to a valid flag. It is required that if an agent in the Root-SoC is able to read the valid flag as being set then if it immediately reads the location of the data buffer it is guaranteed to observe the writes to that buffer.

Inbound Read after Inbound Write

This corresponds to rule B2a in the PCIe ordering rules table. PCIe requires read after write ordering. To ensure this in an AMBA system it is necessary to ensure that the write transaction has completed before the read transaction is issued.

An example of where read after write ordering is required would be first writing to a peripheral device and following this with a read of a status register within that device. To ensure the order of the two transactions is correct it is necessary for the PCIe Interface in the device to wait for the earlier Inbound write transaction to compete before the later Inbound read is issued.

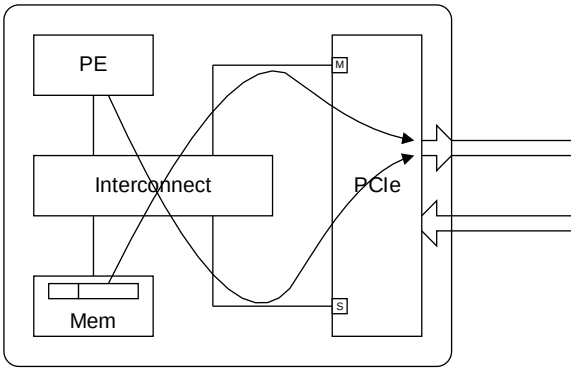
Inbound Write after Inbound Read

No special consideration is required for write after read requests. There is no ordering guarantee provided by the PCIe link for write after read requests and therefore no benefit in establishing order when moving from the PCIe link to an AMBA system.

In some usage models, the ordering of a write after read is naturally established by software or hardware by using the read return data to decide whether the write transaction needs to be issued. This approach naturally provides the serialization (i.e. the second issued only after the first is complete) that provides order.

5.4.5 Outbound request/Inbound completion

This describes the ordering requirements of two packets, one is an Outbound request that is being received by the PCIe Interface Subordinate port and the other is the completion for an Inbound read request (termed as “Inbound completion”) returning via the PCIe Interface Manager port.



Inbound Completion must not overtake Outbound Posted Write [D2]

For a Root-SoC only Inbound posted writes and reads need to be considered, therefore the only completion to be considered is that for an Inbound read request.

For an Endpoint-SoC completions for Inbound requests can also include configuration write responses. However, a Configuration Write Completion is permitted to pass a posted write request [rule D2b], so it is also the case that only completions for Inbound read requests needs to be considered when considering ordering.

For both Root-SoC and Endpoint-SoC the following apply:

For Inbound completion with IDO=0 and RO=0 [Rule D2a in the PCIe ordering table]:

- The PCIe Interface must not allow the completion for an Inbound read request to overtake an Outbound posted write for which completion response has been returned to the AMBA interconnect.

For Inbound completion with (IDO=0 and RO=1) or (IDO=1 and RO=1) [Rule D2b in the PCIe ordering table]:

- The PCIe interface can allow the completion for an Inbound read request to overtake an Outbound posted write.

For Inbound completion with IDO=1 and RO=0 [Rule D2b in the PCIe ordering table]:

- Completions for an Inbound read request with IDO=1 and RO=0 can overtake Outbound posted writes for which a completion response has been returned to the AMBA interconnect provided Requester ID of the posted write and completer ID of the completion are different.

Example Use Case

An example of when this is required is a PE in the Root-SoC writing to a data buffer in Endpoint memory using posted writes and this is followed by a write of the valid flag which resides in Root-SoC memory. The PE must only write the valid flag when posted writes to the data buffer are guaranteed to be observed by any agent which can read the valid flag as set.

The PE that is writing the data buffer using posted writes does not get a guarantee of ordering when the write transaction is accepted. It only gets a guarantee when a write response is given to the posted write. Once the write response has been given by the PCIe interface it is possible that, in a very short period of time, the PE can write the valid flag in Root-SoC memory and this can be read by an Endpoint Inbound read transaction. It is required that the read completion for the transaction from the Endpoint does not overtake the posted writes to the data buffer, so that when the sequence of transactions arrive at the Endpoint the valid flag will only be observed once the writes to the data buffer are guaranteed to be visible. This means that once the Endpoint has

received the data for the read of the valid flag it can immediately issue a sequence of reads to obtain the data buffer and it is guaranteed to observe the most recent writes to the buffer.

So, referring back to the original requirements that “Inbound completion must not overtake Outbound posted write request” it can be seen that this can be re-phrased as “A read completion arriving at the PCIe interface from the AMBA interconnect must not be allowed to overtake a write request for which a completion response has been returned.”

Note: Any read completion that arrives before a write response has been given for a posted write does not need to be ordered with respect to that posted write.

For completeness, the same requirements must be observed within an Endpoint-SoC.

The diagram below illustrates this example:

C1 – PE writes to the data buffer in Endpoint-SoC memory.

P1 – The PCIe interface accepts the write to the data buffer, as required by the writes from step C1 and returns a completion response to the PE.

L1 – The posted write to the data buffer is sent across the link first.

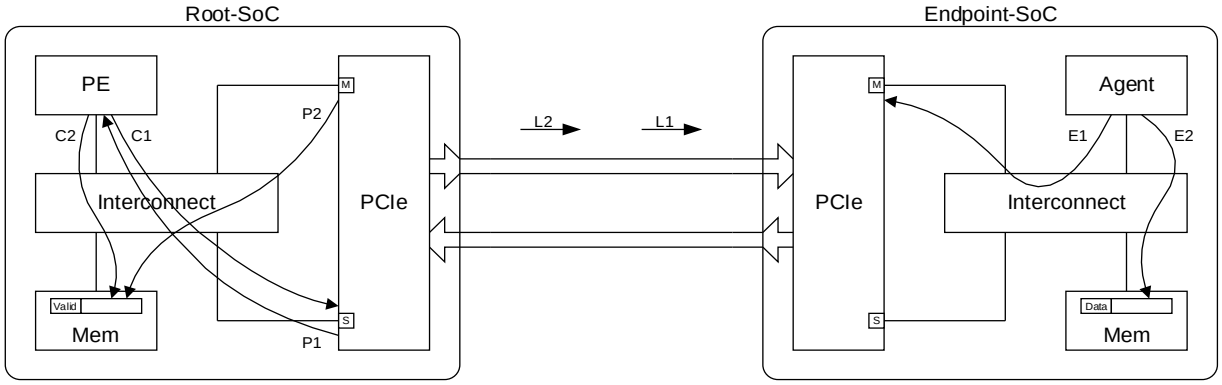
C2 – PE writes to set the valid flag in Root-SoC memory.

E1 – Endpoint reads from Root-SoC memory. For this example, it will read the valid flag as set.

P2 – The PCIe reads the valid flag, as required by step E1 and returns the read data, ordered after the earlier posted write.

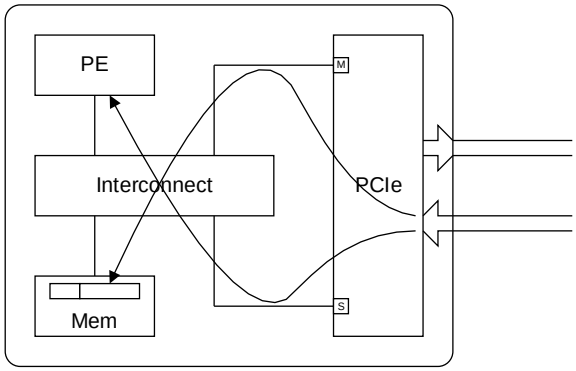
L2 – The read completion from the valid flag is sent across the link second.

E2 – After the Endpoint agent has completed step E1, it reads the data buffer from Endpoint-SoC memory. It is guaranteed to observe the updated data buffer.



5.4.6 Inbound request/Outbound completion

This describes the ordering requirements of two packets, one is an Inbound transaction request being issued by the PCIe Interface Manager port and the other is the completion for an Outbound request (termed as “Outbound completion”) returning via the PCIe Interface Subordinate port.



Outbound completion must not overtake Inbound Posted Write. [D2]

For an Outbound completion with IDO=0 and RO=0 [Rule D2a in the PCIe ordering table]

- The PCIe Interface must ensure that an Inbound posted write has received an AXI write completion response before the read data (which is supplied by the completion) for an Outbound read transaction is returned to the requesting agent. This is required when the completion is received after the write request from the PCIe link.

For an Outbound completion with (IDO=0 and RO=1) or (IDO=1 and RO=1) [Rule D2b in the PCIe ordering table]

- In this case, the completion can be returned to the requester without waiting for the earlier Inbound posted write to receive its completion response.

For an Outbound completion with IDO=1 and RO=0 [Rule D2b in the PCIe ordering table]

- In this case, if the Inbound posted write's Requester ID and the completion's completer ID are different, then the completion can be returned to the requester without waiting for the earlier Inbound posted write to receive its completion response.

Note that completions for Outbound Configuration writes can overtake Inbound posted write [D2b in the PCIe ordering table].

Example Use Case

When considering ordering of completions (which are coming Inbound as a result of Outbound requests) with respect to Inbound transaction requests it is only necessary to consider the relationship between completions for Outbound read requests (i.e. read data returning) and an Inbound posted write.

The requirement is that completions for Outbound read requests (i.e. read data returning) must not overtake an Inbound posted write. In other words, once a PE has received the returning read data it can immediately issue another read transaction and it must be guaranteed to observe the posted write.

An example to illustrate this case is the Endpoint writing a data buffer in Root-SoC memory, using posted writes across the PCIe link followed by a write to set a valid flag which is held in Endpoint memory. When the PE in the Root-SoC attempts to read the valid flag in Endpoint memory it will do so using a read transaction and the returning read data to the PE must force through the posted writes to the data buffer.

The PCIe Interface must ensure that the earlier posted writes are visible to the PE before the read data is returned. It can do this by ensuring that it has received a write completion response for the posted writes before the read data is returned.

The diagram below illustrates this example:

E1 – Endpoint agent writes to data buffer in Root-SoC memory.

E2 – Endpoint agent writes to set valid flag in Endpoint-SoC memory.

C1 – PE reads from Endpoint-SoC memory. For this example, it will read the valid flag as set.

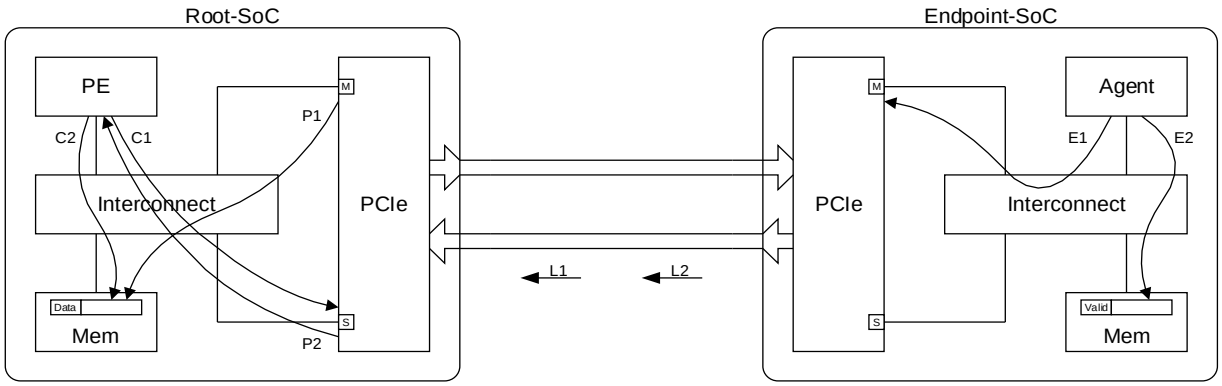
L1 – The posted write to the data buffer is sent across the link first.

L2 – The read completion from the valid flag is sent across the link second.

P1 – The PCIe interface writes to the data buffer, as required by the write from step E1.

P2 – The PCIe interface returns the read data, completing step C1, to the PE.

C2 – After PE has completed step C1, it reads the data buffer from Root-SoC memory.



5.5 Ordering and overtaking rules - quick reference

The table below provides a summary of the rules that must be met by a Root-SoC or an Endpoint-SoC with AMBA interconnect for complying with the PCIe ordering model. The “section” column provides a link to the detailed requirements for each rule.

Type	Packet Direction	Rule	Section
Overtaking	Both requests Outbound	Posted Write must be able to overtake Read Request [A3]	5.3.1
		Posted Write must be able to overtake Configuration Write [A4]	
	Both requests Inbound	Posted Write must be able to overtake Read Request [A3]	5.3.2
		Posted Write must be able to overtake Configuration Write [A4]	
	Outbound request / Inbound Completion	Completion for Inbound requests must be able to overtake Outbound Read Request or Configuration Write. [D3, D4]	5.3.3
	Outbound completion / Inbound request	Completion for Outbound requests must be able to overtake Inbound Read Request or Configuration Write. [D3, D4]	5.3.4
Ordering	Both requests Outbound	Posted Write, Read Request, or Configuration Write must not overtake Posted Write [A2, B2, C2]	5.4.1
	Both requests Inbound	Posted Write must not overtake Posted Write [A2]	5.4.4
		Read Request must not overtake Posted Write [B2]	
		Configuration Write must not overtake Posted Write [C2]	
	Outbound request/ Inbound completion	Completion for Inbound requests must not overtake Outbound Posted Write. [D2]	5.4.5
	Inbound request/ Outbound completion	Completion for Outbound requests must not overtake Inbound Posted Write. [D2]	5.4.6

5.6 IDO and RO for Outbound PCIe transactions

This section summarizes IDO and RO setting recommendations for transactions from PE and non-PE agents.

5.6.1 Root-SoC

- For PE generated requests, see section 4.7.
- For requests from non-PE agents (e.g. DMA engines), when to set RO to 1 is IMPLEMENTATION DEFINED.
- For Root-SoC, it is strongly recommended that IDO is not set to 1 for requests from all agents (PE and non-PE) within the Root-SoC. This is because setting IDO for Root-SoC generated transaction is discouraged by the PCIe specification itself [see section E.5.2 and E.4.2 of [3]].

- For PCIe peer to peer requests being forwarded by the PCIe interface in the Outbound direction, RO and IDO attributes are expected to be supplied by the original requester function.

5.6.2 Endpoint-SoC

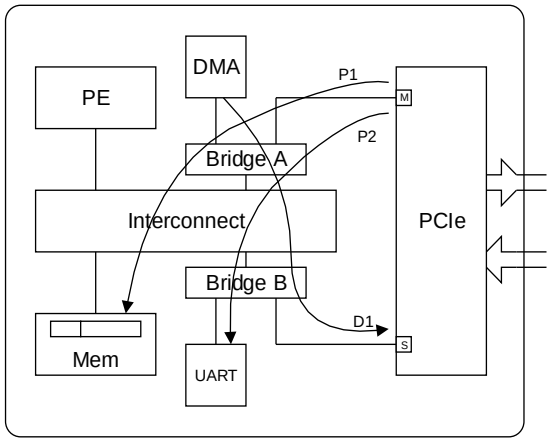
- For PE generated requests, see section 4.7.
- For requests from non-PE agents (e.g. DMA engines), when to set RO to 1 is IMPLEMENTATION DEFINED.
- For requests from non-PE agents in the Endpoint-SoC, when to set IDO to 1 is IMPLEMENTATION DEFINED.

6 Topology Considerations

This section considers some of the topology restrictions that must be observed when integrating PCIe and AMBA systems.

6.1 Bridge Topology Considerations

For the purposes of this discussion a system topology, as shown in the diagram below, is used. This topology has two bridges, Bridge A and Bridge B, on paths that are used for Inbound and Outbound PCI traffic. The bridges are used in the example to allow a discussion of interconnect components that exhibit behaviours. However, a discrete bridge is not required, and the discussion can be applied to any interconnect component.



The key features of the system topology are:

- The PCIe Interface Manager port and some other agent, in this case a DMA, share a bridge, Bridge A.
- The PCIe Interface Subordinate port and some other peripherals, in this case a UART, share a bridge, Bridge B.

To avoid deadlock it must be ensured that posted writes are always able to make forward progress.

6.1.1 Bridge A

It is necessary to ensure that posted writes from PCIe to main memory, path P1, and to peripherals, path P2, can always make forward progress. This must occur whatever traffic is sent by the DMA on path D1.

For a Root-SoC, it is known that all PCIe Inbound write traffic is posted writes, as there cannot be any Inbound configuration writes.

DMA reads from PCIe can be blocked. Therefore, it is necessary to ensure that Inbound posted writes from PCIe can always make forward progress, even if reads transactions issued by the DMA are blocked.

If the DMA were capable of issuing configuration writes to PCIe, it would also be necessary to consider the behaviour of the bridge if these writes were not making forward progress. Also, it would be necessary to consider the case where configuration writes were not making forward progress, and this then prevented posted writes which are using the same AXI write channel from making forward progress.

Ensuring that the bridge has some level of dedicated resource to ensure progress of posted writes from PCIe (or writes in general, if all writes through the bridge can be guaranteed to make forward progress) is recommended.

6.1.2 Bridge B

Bridge B must also ensure that posted writes to PCIe can always make forward progress, even if read transactions to PCIe or configuration writes to PCIe are blocked. In a Root-SoC, support for PCIe per-to-peer traffic also requires posted writes from the PCIe Interface Manager port to make forward progress to the PCIe Interface Subordinate port.

It is recommended that Bridge B contains dedicated resources for write transactions, to ensure that blocked read transactions do not prevent forward progress of write transactions.

In an AXI system, which uses a shared write channel for PCIe posted write transactions and configuration writes, it is necessary to ensure that sufficient resources exist downstream of Bridge B to accept all outstanding configuration writes and ensure that the write channel is not blocked.

6.2 IO Coherent PCIe Traffic

In a system where the Inbound PCIe traffic is of memory type Outer Shareable WB, it is necessary for snoop transactions to be performed for Inbound posted write transactions. The ACE protocol permits a fully coherent Manager port to stall snoop transactions until earlier write transactions are complete, which could include PCIe posted write transactions and PCIe configuration write transactions.

Therefore, it is necessary for an ACE system to ensure PCIe write transactions can make forward progress. This may require sufficient resources, such as a write buffer, to ensure that it is possible to accept all outstanding configuration writes and ensure that the write channel is not blocked.

For a Root-SoC it is permitted to stall Inbound posted write transactions, while Outbound posted write transactions complete. This is required to support peer-to-peer traffic.

However, an Endpoint-SoC is not permitted to stall Inbound posted write transactions while Outbound posted write transactions complete. Therefore, an Endpoint-SoC that requires IO Coherent Inbound traffic must ensure that snoop transactions can always complete without requiring Outbound posted write transactions to complete. This may require sufficient resources to accept all write transactions that could block the progress of snoop transactions for Inbound IO Coherent traffic.

6.3 Intermediate Components

The traffic from a PCIe Interface may have to pass through intermediate system components before reaching its destination. Two examples of this are:

- Transactions to memory may pass through a System Memory Management Unit (SMMU), for the purposes of address translation.
- Interrupt messages may pass through an Interrupt Translation Service (ITS) and Generic Interrupt Controller (GIC), for the purposes of interrupt translation and interrupt delivery.

In both of these cases, and potentially other similar cases, it can be required that the intermediate component must perform read and write accesses to main memory before it can make forward progress on the traffic it is servicing. For example, a read from memory can be required to obtain translation information or a write to memory can be required to update state that is held in a memory-based table.

To ensure the correct operation of such intermediate components, it is necessary to ensure that these components can access memory without being blocked. This can be achieved using either a dedicated physical channel to memory or a virtual network that provides access to memory that is guaranteed not to be blocked.

To ensure that the channel to memory remains free-flowing and is not blocked, it is important to ensure that the channel (either physical or virtual) is not shared with any traffic that may itself need the intermediate component to make forward progress, either directly or indirectly.

The free-flowing memory traffic must not share a path with:

Traffic that could itself require the use of the intermediate component.

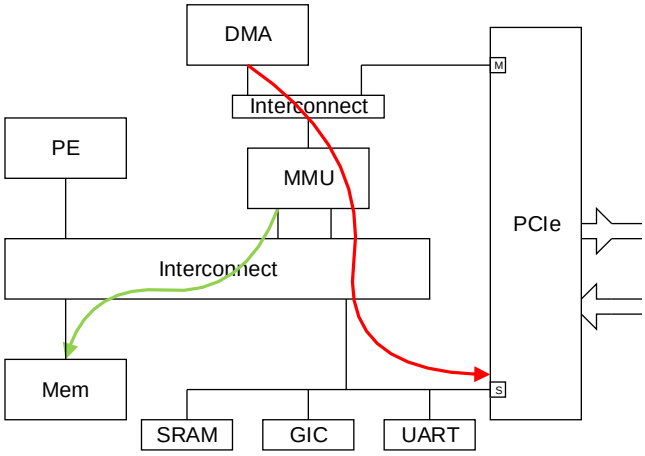
Traffic that could be blocked by other traffic that requires the use of the intermediate component.

The following two sections provide examples to illustrate the requirements.

6.3.1 System MMU and PCIe Example

The first example, as shown in the diagram below, illustrates the use of a System MMU which is used to provide address translation. The green arrow shows the requirement for a non-blocking path to memory which is used to obtain address translation information. To avoid deadlock this path must not be shared with other traffic that itself has a requirement for the SMMU to make forward progress. In this example, the red arrow shows DMA traffic directed towards PCIe. In order to guarantee forward progress for PCIe Outbound traffic, it is necessary to ensure that Inbound PCIe traffic can make forward progress, and this may require use of the SMMU.

Therefore, it is necessary that the SMMU has separate paths for the two traffic types. This can be implemented as separate physical channels or using multiple virtual channels on the same physical wires.



6.3.2 GIC and PCIe Example

The second example, shown below, illustrates a case that needs to be considered when integrating a GIC (Interrupt Controller) with a PCIe Interface. For the purposes of this example, the GIC maintains some state in main memory and therefore must access memory to make forward progress.

The diagram shows traffic from a DMA to the PCIe Interface (red arrow). This traffic, which could be blocked dependent on PCIe Inbound traffic, must not block either of the following:

- GIC traffic (blue arrow) to main memory, which is used to load and store GIC state in which is held in memory-based tables.
- PCIe Interface Inbound traffic, which may be to either the GIC or it could also be to main memory.

The use of separate paths, either separate physical paths or the use of multiple virtual channels on the same physical wires, guarantees forward progress can be made.