Address Translation Services
Revision 1.1

January 26, 2009

REVISION	REVISION HISTORY	DATE
1.0	Initial release	3/8/2007
1.1	PCI Express Capability Structure changed to PCI Express Extended Capability Structure. Page Request Interface and minor clarifications added.	1/26/2009

PCI-SIG disclaims all warranties and liability for the use of this document and the information contained herein and assumes no responsibility for any errors that may appear in this document, nor does PCI-SIG make a commitment to update the information contained herein.

Contact the PCI-SIG office to obtain the latest revision of this specification.

Questions regarding the ATS Specification or membership in PCI-SIG may be forwarded to:

Membership Services
www.pcisig.com
E-mail: administration@pcisig.com
Phone: 503-619-0569
Fax: 503-644-6708

Technical Support
techsupp@pcisig.com

DISCLAIMER

Contents

Figures

Tables

Preface

This specification describes the extensions required to allow PCI Express Devices to interact with an address translation agent (TA) in or above a Root Complex (RC) to enable translations of DMA addresses to be cached in the Device. The purpose of having an Address Translation Cache (ATC) in a Device is to minimize latency and to provide a scalable distributed caching solution that will improve I/O performance while alleviating TA resource pressure.

This specification must be used in conjunction with the *PCI Express Base Specification, Revision 1.1*, and associated ECNs.

Document Organization

The specification is organized into the following five sections:

1. Introduction and Architectural Overview of the Address Translation Services (ATS) – This section covers the problem space and associated approach to solving this space including ATS operations.
2. ATS TLP Messages and Associated Semantics – This section provides a detailed discussion of the ATS TLP messages and their operational semantics.
3. ATS Invalidation Protocol – This section provides a detailed discussion of the ATS Invalidation protocol with a number of implementation notes to enable developers implementing to this specification.
4. Page Request Services – This section provides a detailed discussion of the ATS Page Request interface.
5. Configuration – This section provides a detailed discussion of ATS configuration, how to enable an ATC, etc.

Documentation Conventions

Capitalization

Some terms are capitalized to distinguish their definition in the context of this document from their common English meaning. Words not capitalized have their common English meaning. When terms such as "memory write" or "memory read" appear completely in lower case, they include all transactions of that type.

Register names and the names of fields and bits in registers and headers are presented with the first letter capitalized and the remainder in lower case.

Numbers and Number Bases

Hexadecimal numbers are written with a lower case "h" suffix, e.g., FFFh and 80h. Hexadecimal numbers larger than four digits are represented with a space dividing each group of four digits, as in 1E FFFF FFFFh. Binary numbers are written with a lower case "b" suffix, e.g., 1001b and 10b. Binary numbers larger than four digits are written with a space dividing each group of four digits, as in 1000 0101 0010b.

All other numbers are decimal.

Reference Information

Reference information is provided in various places to assist the reader and does not represent a requirement of this document. Such references are indicated by the abbreviation "(ref)." For example, in some places, a clock that is specified to have a minimum period of 400 ps also includes the reference information maximum clock frequency of "2.5 GHz (ref)."

Requirements of other specifications also appear in various places throughout this document and are marked as reference information. Every effort has been made to insure that this information accurately reflects the referenced document; however, in case of a discrepancy, this document takes precedence.

Implementation Notes

Implementation Notes should not be considered to be part of this specification. They are included for clarification and illustration only.

Terms and Acronyms

Address Translation and Protection Table (ATPT)	The data structure(s) accessed by a Translation Agent to determine the mapping of untranslated DMA addresses into translated addresses. The data structure may contain fields that indicate the protection attributes of the translation entry.
Address Translation Cache (ATC)	A hardware entity that stores recently used address translations. This term is used instead of Translation Look-aside Buffer (TLB) to differentiate the TLB used for I/O from the TLB used by the CPU. Each TA is expected to have an ATC co-located with it, but an ATC need not be co-located with a TA.
Address Translation Services (ATS)	The set of configuration, wire protocol, ATC, etc., required to deliver an address translation solution.
Clear	A bit with the value of 0b or the act of causing a bit to have the value of 0b.
PCIe	PCI Express
RP	PCIe Root Port. See the *PCI Express Base Specification* for additional details.
Set	A bit with the value of 1b or the act of causing a bit to have the value of 1b.
Smallest Translation Unit (STU)	The minimum increment for translation and invalidation. This value is expressed in terms of 4096-byte units and is programmed into a configuration register on the Function.
TC	PCIe Traffic Class. See the *PCI Express Base Specification* for additional details.
Translated Address	An address formed by using the results of an address Translation Request.

Translation Agent (TA)	A logical entity that converts addresses expressed in terms of one address space into an address in a different address space. A Translation Agent (TA) is associated with memory that contains translation tables that the TA will reference for address translation (see ATPT). A TA may contain an Address Translation Cache. A TA may be implemented either in hardware, software, or a combination of both. A TA is treated as implementation-specific and, therefore, is outside the scope of this specification.
Untranslated Address	An address that is formed using the existing programming mechanisms of PCIe.

1

1. Architectural Overview

Most contemporary system architectures make provisions for translating addresses from DMA (bus mastering) I/O Functions. In many implementations, it has been common practice to assume that the physical address space seen by the CPU and by an I/O Function is equivalent. While in others, this is not the case. The address programmed into an I/O Function is a "handle" that is processed by the Root Complex (RC). The result of this processing is often a translation to a physical memory address within the central complex. Typically, the processing includes access rights checking to insure that the DMA Function is allowed to access the referenced memory location(s).

The purposes for having DMA address translation vary and include:

- Limiting the destructiveness of a "broken" or miss-programmed DMA I/O Function
- Providing for scatter/gather
- Ability to redirect message-signaled interrupts (e.g., MSI or MSI-X) to different address ranges without requiring coordination with the underlying I/O Function
- Address space conversion (32-bit I/O Function to larger system address space)
- Virtualization support

Irrespective of the motivation, the presence of DMA address translation in the host system has certain performance implications for DMA accesses.

Depending on the implementation, DMA access time can be significantly lengthened due to the time required to resolve the actual physical address. If an implementation requires access to a main-memory-resident translation table, the access time can be significantly longer than the time for an untranslated access. Additionally, if each transaction requires multiple memory accesses (e.g., for a table walk), then the memory transaction rate (i.e., overhead) associated with DMA can be high.

To mitigate these impacts, designs often include address translation caches in the entity that performs the address translation. In a CPU, the address translation cache is most commonly referred to as a translation look-aside buffer (TLB). For an I/O TA, the term address translation cache or ATC is used to differentiate it from the translation cache used by the CPU.

While there are some similarities between TLB and ATC, there are important differences. A TLB serves the needs of a CPU that is nominally running one thread at a time. The ATC, however, is generally processing requests from multiple I/O Functions, each of which can be considered a separate thread. This difference makes sizing an ATC difficult depending upon cost models and expected technology reuse across a wide range of system configurations.

The mechanisms described in this specification allow an I/O Device to participate in the translation process and provide an ATC for its own memory accesses. The benefits of having an ATC within a Device include:

- Ability to alleviate TA resource pressure by distributing address translation caching responsibility (reduced probability of "thrashing" within the TA)
- Enable ATC Devices to have less performance dependency on a system's ATC size
- Potential to ensure optimal access latency by sending pretranslated requests to central complex

This specification will provide the interoperability that allows PCIe Devices to be used in conjunction with a TA, but the TA and its Address Translation and Protection Table (ATPT) are treated as implementation-specific and are outside the scope of this specification. While it may be possible to implement ATS within other PCIe Components, this specification is confined to PCIe Devices and PCIe Root Complex Integrated Endpoints.

Figure 1-1 illustrates an example platform with a TA and ATPT, along with a set of PCIe Devices and RC Integrated Endpoints with integrated ATC. A TA and an ATPT are implementation-specific and can be distinct or integrated components within a given system design.

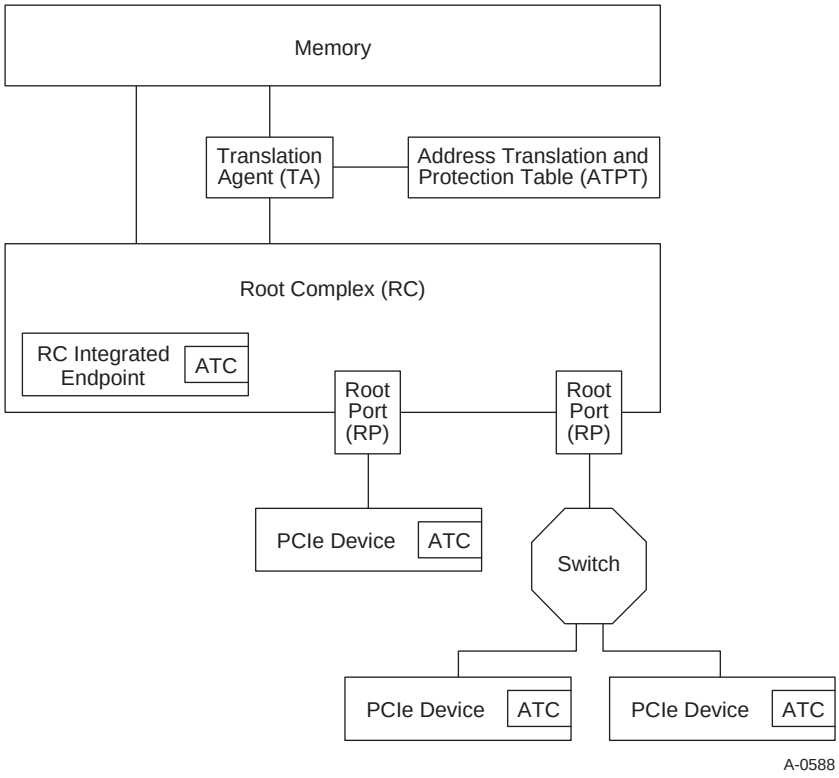

Figure 1-1: Example Illustrating a Platform with TA, ATPT, and ATC Elements

1.1. Address Translation Services (ATS) Overview

ATS builds upon the *PCI Express Base Specification* to provide a new set of TLP and associated semantics. ATS uses a request-completion protocol between a Device[1] and a Root Complex (RC) to provide translation services. In addition, a new AT field is defined within the Memory Read and Memory Write TLP as defined within the *PCI Express Base Specification*. The new AT field enables an RC to determine whether a given request has been translated or not via the ATS protocol.

Figure 1-2 illustrates the basic flow of an ATS Translation Request operation.

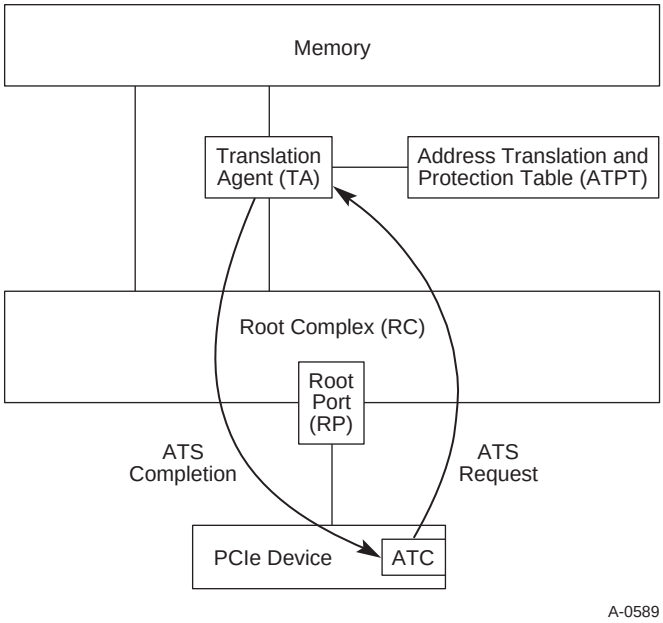

Figure 1-2: Example ATS Translation Request/Completion Exchange

In this example, a Function-specific work request is received by a single-Function PCIe Device. The Function determines through an implementation-specific method that caching a translation within its ATC would be beneficial. There are a number of considerations a Function or software can use in making such a determination; for example:

- Memory address ranges that will be frequently accessed over an extended period of time or whose associated buffer content is subject to a significant update rate
- Memory address ranges, such as work and completion queue structures, data buffers for low-latency communications, graphics frame buffers, host memory that is used to cache Function-specific content, and so forth

Given the variability in designs and access patterns, there is no single criteria that can be applied.

---

[1] All references within this specification to a Device apply equally to a PCIe Device or a Root Complex Integrated Endpoint. ATS does not delineate between these two types in terms of requirements, semantics, configuration, error handling, etc. From a software perspective, an ATS-capable Root Complex Integrated Endpoint must behave the same as an ATS-capable non-integrated Device.

The Function generates an ATS Translation Request which is sent upstream through the PCIe hierarchy to the RC which then forwards it to the TA. An ATS Translation Request uses the same routing and ordering rules as defined within the *PCI Express Base Specification*. Further, multiple ATS Translation Requests can be outstanding at any given time; i.e., one may pipeline multiple requests on one or more TC. Each TC represents a unique ordering domain and defines the domain that must be used by the associated ATS Translation Completion.

Upon receipt of an ATS Translation Request, the TA performs the following basic steps:

1. Validates that the Function has been configured to issue ATS Translation Requests.
2. Determines whether the Function may access the memory indicated by the ATS Translation Request and has the associated access rights.
3. Determines whether a translation can be provided to the Function. If yes, the TA issues a translation to the Function.
    a. ATS is required to support a variety of page sizes to accommodate a range of ATPT and processor implementations.
        i. Page sizes are required to be a power of two and naturally aligned.
        ii. The minimum supported page size is 4096 bytes. ATS capable components are required to support this minimum page size.
    b. A Function must be informed of the minimum translation or invalidate size it will be required to support to provide the Function an opportunity to optimize its resource utilization. The smallest minimum translation size must be 4096 bytes.
4. The TA communicates the success or failure of the request to the RC which generates an ATS Translation Completion and transmits via a Response TLP through a RP to the Function.
    a. An RC is required to generate at least one ATS Translation Completion per ATS Translation Request; i.e., there is minimally a 1:1 correspondence independent of the success or failure of the request.
        i. A successful translation can result in one or two ATS Translation Completion TLPs per request. The Translation Completion indicates the range of translation covered.
        ii. An RC may pipeline multiple ATS Translation Completions; i.e., an RC may return multiple ATS Translation Completions and these ATS Translation Completions may be in any order relative to ATS Translation Requests.
        iii. The RC is required to transmit the ATS Translation Completion using the same TC (Traffic Class) as the corresponding ATS Translation Request.
    b. The requested address may not be valid. The RC is required to issue a Translation Completion indicating that the requested address is not accessible.

When the Function receives the ATS Translation Completion and either updates its ATC to reflect the translation or notes that a translation does not exist. The Function proceeds with processing its work request and generates subsequent requests using either a translated address or an untranslated address based on the results of the Completion.

a. Similar to Read Completions, a Function is required to allocate resource space for each completion(s) without causing backpressure on the PCIe Link.

b. A Function is required to discard Translation Completions that might be "stale." Stale Translation Completions can occur for a variety of reasons.

As one can surmise, ATS Translation Request and Translation Completion processing is conceptually similar and, in many respects, identical to PCIe Read Request and Read Completion processing. This is intentional to reduce design complexity and to simplify integration of ATS into existing and new PCIe-based solutions. Keeping this in mind, ATS requires the following:

- ATS capable components must interoperate with *PCI Express Base Specification, Revision 1.1* compliant components.
- ATS is enabled through a new Capability and associated configuration structure. To enable ATS, software must detect this Capability and enable the Function to issue ATS TLP. If a Function is not enabled, the Function is required not to issue ATS Translation Requests and is required to issue all DMA Read and Write Requests with the TLP AT field set to "untranslated."
- ATS TLPs are routed using either address-based or Requester ID (RID) routing.
- ATS TLPs are required to use the same ordering rules as specified within the *PCI Express Base Specification*.
- ATS TLPs are required to flow unmodified through PCIe 1.1-compliant Switches.
- A Function is permitted to intermix translated and untranslated requests.
- ATS transactions are required not to rely upon the address field of a memory request to communicate additional information beyond its current use as defined by the PCI-SIG.

IMPLEMENTATION NOTE

Address Range Overlap

It is likely that the untranslated and translated address range will overlap, perhaps in their entirety. This is not a requirement of ATS but may be an implementation constraint on the TA so that memory requests will be properly routed.

In contrast to the prior example, Figure 1-3 illustrates an example multi-Function Device. In this example Device, there are three Functions. Key points to note in Figure 1-3 are:

- Each ATC is associated with a single Function. Each ATS-capable Function must be able to source and sink at least one of each ATS Translation Request or Translation Completion type.
- Each ATC is configured and accessed on a per Function basis. A multi-Function Device is not required to implement ATS on every Function.
- If the ATC implementation shares resources among a set of Functions, then the logical behavior is required to be consistent with fully independent ATC implementations.

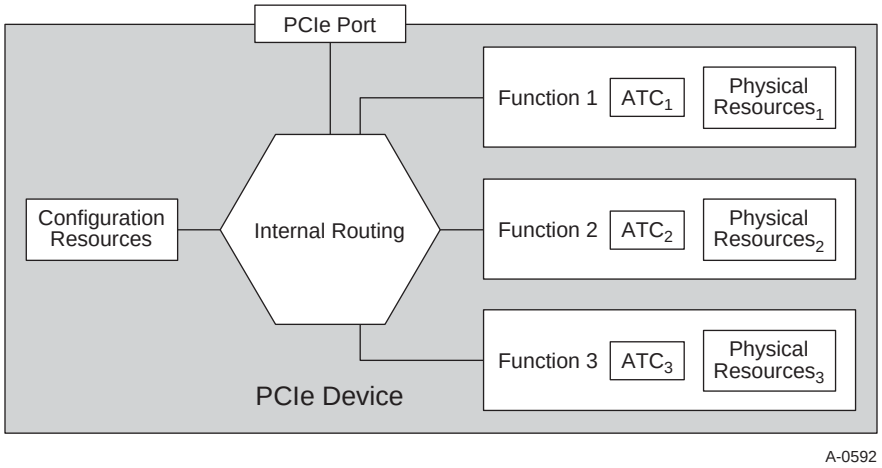

Figure 1-3: Example Multi-Function Device with ATC per Function

Independent of the number of Functions within a Device, the following are required:

- A Function is required not to issue any TLP with the AT field set unless the address within the TLP was obtained through the ATS Translation Request and Translation Completion protocol.
- Each ATC is required to only be populated using the ATS protocol; i.e., each entry within the ATC must be filled via an ATS Translation Completion in response to the Function issuing an ATS Translation Request for a given address.
- Each ATC cannot be modified except through the ATS protocol. That is:
  - Host system software cannot modify the ATC other than through the protocols defined in this specification except to invalidate one or more translations in an ATC. A Device or Function reset would be an example of an operation performed by software to change the contents of the ATC, but a reset is only allowed to invalidate entries not modify their contents.
  - It must not be possible for host system software to use software executing on the Device to modify the ATC.

When a TA determines that a Function should no longer maintain a translation within its ATC, the TA initiates the ATS invalidation protocol. The invalidation protocol consists of a single Invalidation Request and one or more Invalidate Completions.

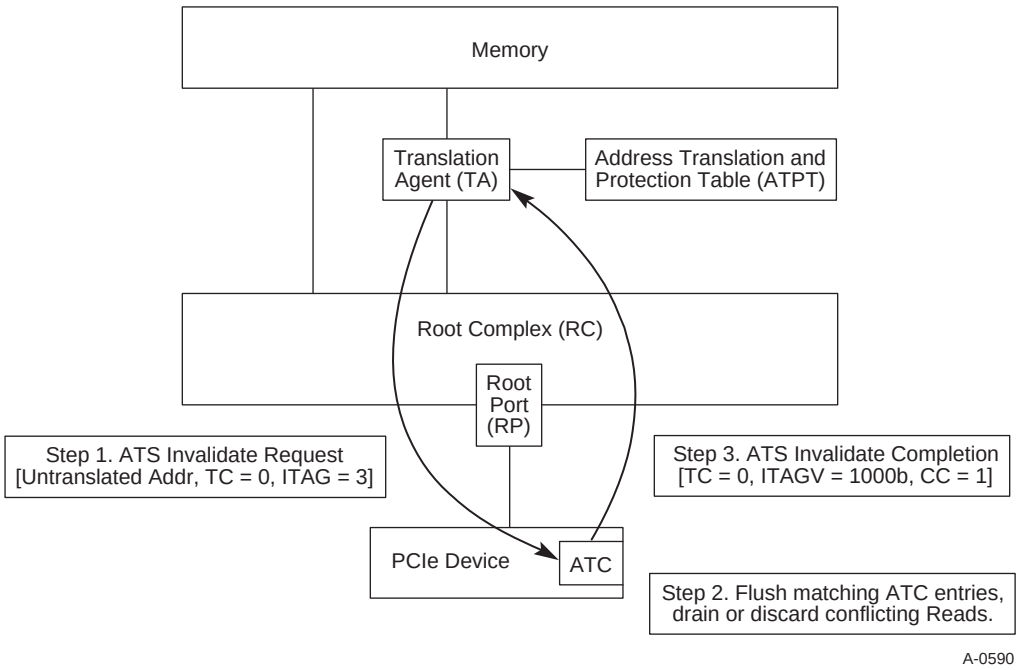

Figure 1-4: Invalidation Protocol with a Single Invalidation Request and Completion

As Figure 1-4 illustrates, there are essentially three steps in the ATS Invalidation protocol:

1. The system software updates an entry in the tables used by the TA. After the table is changed, the TA determines that a translation should be invalidated in an ATC and initiates an Invalidation Request TLP which is transmitted from the RP to the example single-Function Device. The Invalidate Request communicates an untranslated address range, the TC, and an RP unique tag which is used to correlate Invalidate Completions with the Invalidation Request.
2. The Function receives the Invalidate Request and invalidates all matching ATC entries. A Function is not required to immediately flush all pending requests upon receipt of an Invalidate Request. If transactions are in a queue waiting to be sent, it is not necessary for the Function to expunge requests from the queue even if those transactions use an address that is being invalidated.
    a. A Function is required not to indicate the invalidation has completed until all outstanding Read Requests or Translation Requests that reference the associated translated address have been retired or nullified.
    b. A Function is required to ensure that the Invalidate Completion indication to the RC will arrive at the RC after any previously posted writes that use the "stale" address.

3. When a Function has ascertained that all uses of the translated address are complete, it issues one or more ATS Invalidate Completions.
   a. An Invalidate Completion is issued for each TC that may have referenced the range invalidated. These completions act as a flush mechanism to ensure the hierarchy is cleansed of any in-flight transactions which may contain references to the translated address.
      i. The number of Completions required is communicated within each Invalidate Completion. A TA or RC implementation can maintain a counter to ensure that all Invalidate Completions are received before considering the translation to no longer be in use.
      ii. If more than one Invalidation Complete is sent, the Invalidate Completion sent in each TC must be identical in the fields detailed in Section 3.2.
   b. An Invalidate Completion contains the ITAG from Invalidate Request to enable the RC to correlate Invalidate Requests and Completions.

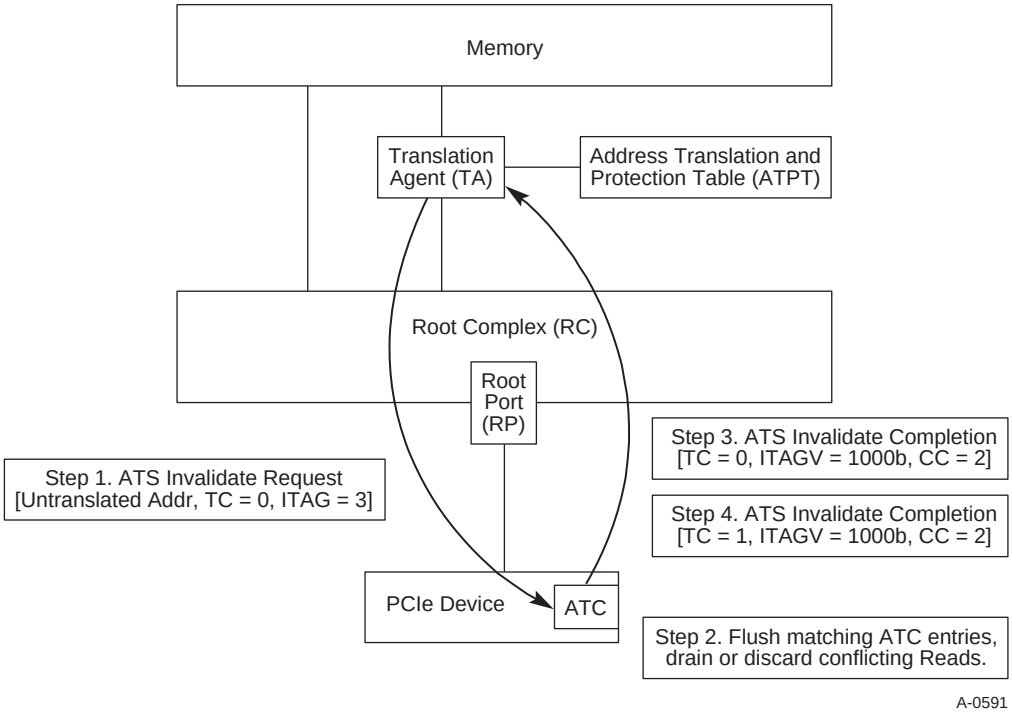

Figure 1-5: Single Invalidate Request with Multiple Invalidate Completions

1.2. Page Request Interface Extension

ATS improves the behavior of DMA based data movement. An associated Page Request Interface (PRI) provides additional advantages by allowing DMA operations to be initiated without requiring that all the data to be moved into or out of system memory be pinned.[2] The overhead associated with pinning memory may be modest, but the negative impact on system performance of removing large portions of memory from the pageable pool can be significant.

PRI is functionally independent of the other aspects of ATS. That is, a device that supports ATS need not support PRI, but PRI is dependent on ATS's capabilities.

Intelligent I/O devices can be constructed to make good use of a more dynamic memory interface. Pinning will always have the best performance characteristics from a device's perspective–all the memory it wants to touch is guaranteed to be present. However, guaranteeing the residence of all the memory a device might touch can be problematic and force a sub-optimal level of device awareness on a host. Allowing a device to operate more independently (to page fault when it requires memory resources that are not present) provides a superior level of coupling between device and host.[3]

The mechanisms used to take advantage of a Page Request Interface are very device specific. As an example of a model in which such an interface could improve overall system performance, let us examine a high-speed LAN device. Such a device knows its burst rate and need only have as much physical buffer space available for inbound data as it can receive within some quantum. A vector of unpinned virtual memory pages could be made available to the device, that the device then requests as needed to maintain its burst window. This minimizes the required memory footprint of the device and simplifies the interface with the host, both without negatively impacting performance.

The ability to page, begs the question of page table status flag management. Without any additional hints about how to manage pages mapped to an I/O device, an RC would need to conservatively assume that if an I/O device can write to a page, it has written to the page. I/O writable pages would need to be marked as dirty before their translated addresses are made available to a device.

This conservative dirty-on-write-permission-grant behavior is generally not a significant issue for non-pageable devices, where I/O pages are pinned and the cost of saving a clean page to memory will seldom be paid. However, devices that support the Page Request Interface could pay a significant penalty if all writable pages are treated as dirty, since such devices operate without pinning their accessible memory footprints and may issue speculative page requests for performance. The cost of saving clean pages (instead of just discarding them) in such systems can diminish the value of otherwise attractive paging techniques.

The No Write (NW) flag in Translation Requests indicates whether a page should be marked as dirty if it is writable, or whether a device is willing to restrict its usage to only reading the page, independent of the access rights that would otherwise have been granted. If a device chooses to

---

[2] Locked in place so that it cannot be swapped out by the system's dynamic paging mechanism.

[3] The alternative is a private interface between a device and its driver that is used to communicate device state so that the driver can ensure the availability of pinned memory resources.

request only read access and then determines that it does wish to write to the page, then the device would need to issue a new Translation Request.

2

2. ATS Translation Services

A TA does translations. An ATC can cache those translations. If an ATC is separated from the TA by PCIe, the memory request from an ATC will need to be able to indicate if the address in the transaction is translated or not. The modifications to the memory transactions are described in this section, as are the transactions that are used to communicate translations between a remote ATC and a central TA.

2.1. Memory Requests with Address Type

A Function with an ATC can send memory read/write Requests that contain either translated or untranslated addresses. As shown in Figure 2-1 and Figure 2-2, the Address Type (AT) field is used to indicate the type of address that is present in the request header.

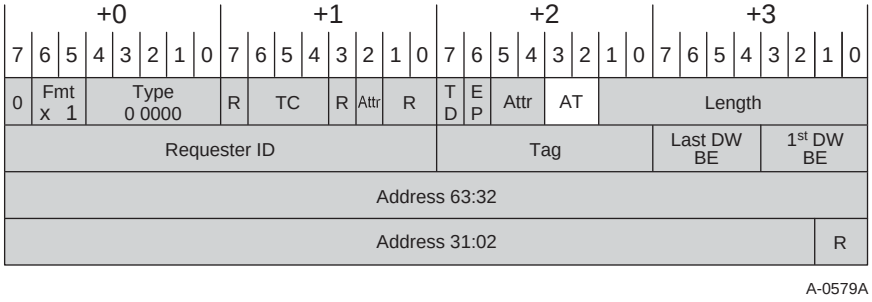

Figure 2-1: Memory Request Header with 64-bit Address

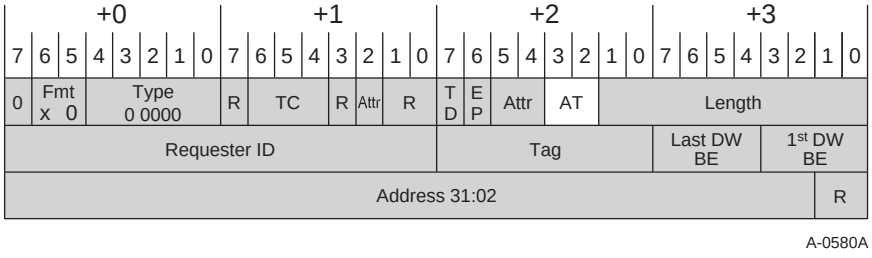

Figure 2-2: Memory Request Header with 32-bit Address

The AT field in the requests is a redefinition of a reserved field in the *PCI Express Base Specification*. Functions that do not implement an ATC will continue to set the AT field to its defined reserved value (00b). Functions that implement an ATC will set the AT field as listed in Table 2-1.

Table 2-1: Address Type (AT) Field Encodings

AT Coding	Mnemonic	Meaning
00b	Untranslated	A TA may treat the address as either virtual or physical.
01b	Translation Request	The TA will return the translation of the address contained in the address field of the request as a read completion. This value only has meaning for an explicit Translation Request (see Section 2.2). The TA will signal an Unrecognized Request (UR) if it receives a TLP with the AT field set to 01b in a Memory Request other than Memory Read.
10b	Translated	The address in the transaction has been translated by an ATC. If the Function associated with the SourceID is allowed to present physical addresses to the system memory, then the TA might not translate this address. If the Function is not allowed to present physical addresses, then the TA may treat this as an UR.
11b	reserved	The TA will signal an Unrecognized Request (UR) if it receives a Memory Request TLP with the AT field set to 11b.

The AT field is only defined for Memory Requests. The field remains reserved for other TLPs.

2.2. Translation Requests

A Translation Request has a format that is similar to that of a memory read. The AT field is used to differentiate a Translation Request from a normal memory read.

The request header for a Translation Request has the formats illustrated in Figure 2-3 and Figure 2-4.

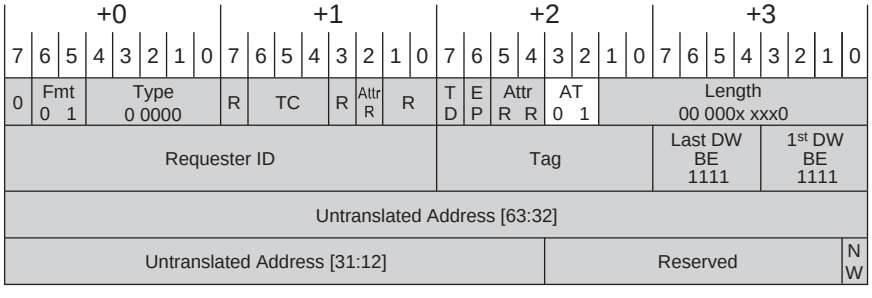

Figure 2-3: 64-bit Translation Request Header

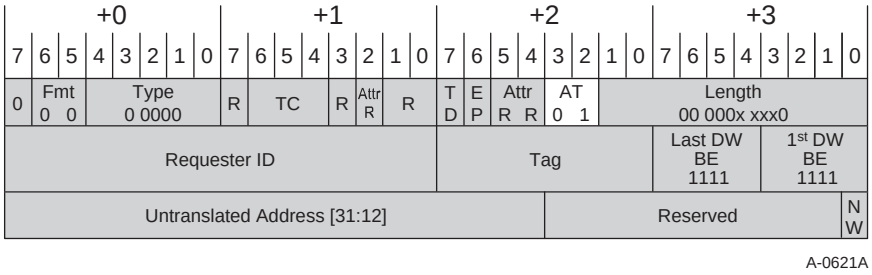

Figure 2-4: 32-bit Translation Request Header

Translation Requests have the same completion timeout intervals as Read Requests.

2.2.1. Attribute Field

For a Translation Request, the Attr field is reserved for future use. There are no ordering requirements for a Translation Request. A TA may reorder a Translation Request with respect to any other request.

IMPLEMENTATION NOTE

Translation Request Ordering

Because no ordering can be assumed between Translation Requests and other types of Requests, a Translation Request does not make an effective flushing/ordering primitive.

2.2.2. Length Field

The Length field is set to indicate how many translations may be returned in response to this request. Each translation is 8 bytes in length and represents one or more STUs (Smallest Translation Unit). The maximum setting for the Length field is the RCB. The Length field in a Translation Request must always indicate an even number of DWORDs. If Length is set to indicate a value greater than allowed, or if the least-significant bit of the Length field is non-zero, then the TA will treat the request as a Malformed Packet.

If the Length field has a value greater than two, then the Function is requesting translations for a range of memory greater than a single STU. The additional translations, if provided, are assumed to be for sequentially-increasing, equal-sized, STU-aligned regions, starting at the requested address.

2.2.3. Tag Field

The Tag field has the same meaning as in a Memory Read Request.

2.2.4. Untranslated Address Field

A Translation Request includes either a 32-bit or a 64-bit Untranslated Address field. This field indicates the address to be translated. The TA will make decisions about the validity of the request, based on the address in the translation request. The TA is permitted to return fewer translations than requested, but it will not return more.

When multiple translations are requested, the TA will not return a translation if the range of that translation does not overlap the implied range of the Translation Request (this would only apply to translations after the initial value). The implied range of the Translation Request is $[2^{STU+12} * (Length/2)]$ bytes.

The Untranslated Address field in the Translation Request is any address in the range of the first STU. Address bits 11:0 are not present in the Translation Request and are implied to be zero. If a requester has Page Aligned Request Set (see Section 5.1.2), it must ensure that bits 11:2 are zero. If a requester has Page Aligned Request Clear, it is permitted to supply any value for bits 11:2.[4] The TA must ignore bits 11:2 as well as any low-order bits not required to determine the translation.

For example, if using 64-bit addressing for a Function with the Page Aligned Request bit Set that is programmed with an STU of 1 (i.e., 8192-byte pages), bits 63:13 are significant, bit 12 is ignored by the TA and bits 11:0 are implied to be zero.

2.2.5. No Write (NW) Flag

The No Write flag, when Set, indicates that the Function is requesting read-only access for this translation.

The TA may ignore the No Write Flag, however, if the TA responds with a translation marked as read-only then the Function must not issue MemWrite transactions using that translation. In this case, the Function may issue another translation request with the No Write flag Clear, which may result in a new translation completion with or without the W (Write) bit Set.

2.3. Translation Completion

A Translation Completion (either a Cpl or a CplD) is sent by a TA for each Translation Request. This specification describes the meaning of fields in Translation Completions. Fields not defined in this specification have the same meanings proscribed for Read Completions in the *PCI Express Base Specification*. The Attr field is reserved for future use.

If the TA was not able to perform the requested translation, a completion with the format shown in Figure 2-5 is used.

[4] Note: An earlier version of this specification did not support the Page Aligned Request bit.

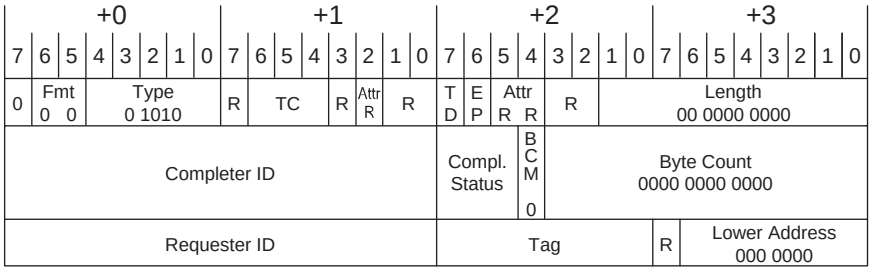

Figure 2-5: Translation Completion with No Data

The values and meaning for the Completion Status field are listed in Table 2-2.

Table 2-2: Translation Completion with No Data Status Codes

Value	Status	Meaning
000b	Success	This completion status has a nominal meaning of "success." The TA will not return this value in a Cpl.
001b	Unsupported Request (UR)	Translation Requests from this Function are not supported by the TA. If a Function receives this Completion code, it must disable its ATC and not send requests using translated addresses until the ATC is re-enabled. For transactions the Function may internally have in flight, the Function may either terminate or complete them. The mechanism a Function receiving this code uses to report this condition is outside the scope of this specification. The TA detecting this error is a "Completer Sending a Completion with UR/CA Status" and shall behave as defined in the *PCI Express Base Specification.*
010b	CRS	This value is not allowed in any Completion to a request initiated by a PCI Express Function. If received by a Function, it shall be treated as a Malformed TLP.
100b	Completer Abort (CA)	The TA was not able to translate the address because of an error in the TA. This nominally causes an error to be reported to the device driver associated with the ATC. See AER within the *PCI Express Base Specification.*
All others	Reserved	A Translation Completion with a Reserved Completion Status value is treated as if the Completion Status was Unsupported Request (001b).

Note: Return values other than *Success* indicate an error.

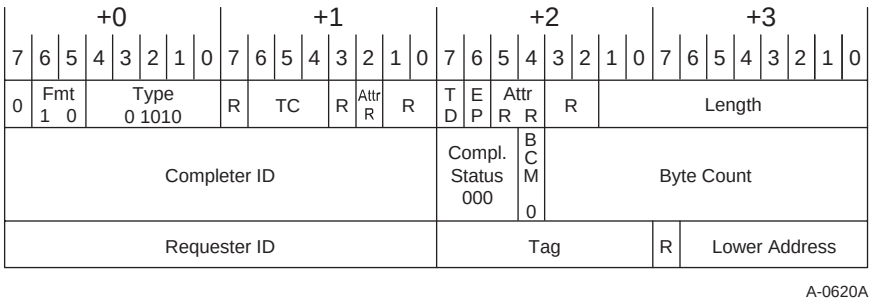

Figure 2-6: Successful Translation Completion

Fields are set in accordance with Sections 2.2.9 and 2.3.1 of the *PCI Express Base Specification*.

Translation Completions must be sent using the same TC as the Translation Request. The Function is not required to verify that the same TC was used.

The Lower Address field will contain a value that will make the packet consistent with RCB semantics. If the result is returned in a single packet, Lower Address is set to RCB minus Byte Count. If the results are returned in multiple packets, the first packet will have a Lower Address field of RCB minus (Length * 4) and subsequent packets will have a Lower Address field of 000 0000b.

If the Completion Status field is 000b, then the translation was successful and a data payload will follow the header. The contents of the data payload are shown in Figure 2-7.

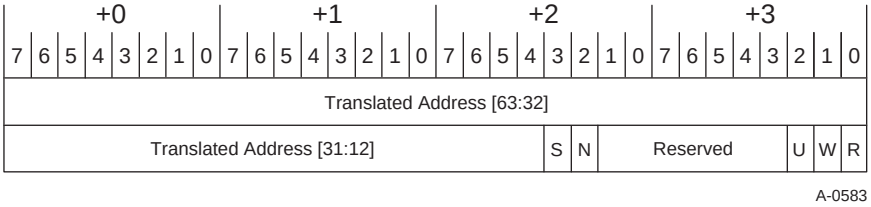

Figure 2-7: Translation Completion Data Entry

Table 2-3: Translation Completion Data Fields

Field	Meaning
S	**Size of translation** – This field is 0b if the translation applies to a 4096-byte range of memory. If this field is 1b, then the translation applies to a range of memory that is larger than 4096 bytes (see Section 2.3.1).
N	**Non-snooped accesses** – If this field is 1b, then the read and write requests that use this translation must Clear the No Snoop bit in the Attribute field. If it is 0b, then the Function may use other means to determine if No Snoop should be Set.
Reserved	These bits shall be ignored by the ATC.
U	**Untranslated access only** – When this field is Set to 1b in a Translation Completion entry, the indicated range may only be accessed using untranslated addresses, and the Translated Address field of this Translation Completion entry may not be used in a subsequent Read/Write Request with AT set to Translated. This value may be cached if R or W is Set.
R,W	**Read, Write** – These two fields indicate the transaction types that are allowed for the requests using the translation. The encodings are: 00b Neither read nor write transactions are allowed. This translation is considered not to be valid. The contents of the Translated Address, N, and U fields are undefined. A translation with this value may not be cached in the ATC. 01b Write Requests that target this range are allowed, but Read Requests are not unless they are zero-length reads. 10b Read Requests that target this range are allowed, but Write Requests are not. 11b Read and Write Requests that target this range are allowed.

2.3.1. Translated Address Field

If the R and W fields are both Clear, or if U is Set, then the Translated Address field may not be used by the Function for any purpose.

If either the R or W field is Set, and the U field is Clear, then the Translated Address field contains an address that can be used by the Function in a Memory Request with the AT field set to Translated and the Function may cache the Translated Address. When cached, the R and W fields must be stored with the same value as the Translation Completion entry. The address that is cached must be a subset of the address range indicated in the Translation Completion (the subset may include the entire range).

While the Translated Address is cached in the Function's ATC, it shall not be possible for the Function to modify the entry other than to delete it. The entry must be deleted from the ATC when an Invalidation Request is received that has an indicated range that overlaps any portion of the cached address.

A Function is not allowed to make an entry into its ATC unless the entry is in a Translation Completion and the E (Enable) field within the ATS Capability is Set. Entries in an ATC cache that are written before the E field is Set must not be used in Memory Request. They must either be invalidated when the E field is Set or ignored and not used.

2.3.2. Translation Range Size (S) Field

If S is Set, then the translation applies to a range that is larger than 4096 bytes. If S = 1b, then bit 12 of the Translated Address is used to indicate whether or not the range is larger than 8192 bytes. If bit 12 is 0b, then the range size is 8192 bytes, but it is larger than 8192 bytes if set to 1b. If S = 1b and bit 12 = 1b, then bit 13 is used to determine if the range is larger than 16384 bytes or not. If bit 13 is 0b, then the range size is 16384 bytes, but it is larger than 16384 bytes if set to 1b.

Low-order address bits are consumed in sequence to indicate the size of the range associated with the translation.

> Note: This encoding method is also used to indicate the size of the memory range being invalidated.

Examples for different translation sizes are shown in Table 2-4.

Table 2-4: Examples of Translation Size Using S Field

Address Bits																						Translation Range Size in Bytes
63:32*	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	S	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	4 K
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	8 K
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	1	16 K
x	x	x	x	x	x	x	x	x	x	x	x	0	1	1	1	1	1	1	1	1	1	2 M
x	x	x	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1 G
x	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4 G

*Note: Upper address bits are used to indicate the size for ranges larger than 4 GB.

The size field is set to indicate the range size in multiples of 4096 bytes regardless of the setting of STU. For example, if STU is set to indicate that the minimum translation is 8192 bytes, then S should be Set on all translation returned in a Translation Completion and in all Invalidate Requests. If STU is set to indicate a 16384-byte minimum, then S and bit 12 would both be Set in all translation and invalidate ranges.

If S is Set and bits 63:12 are all 1b, then the behavior is undefined. If S is Set and bit 63 is 0b, and bits 62:12 are all 1b, then the request is to invalidate all translations.

If a Function receives a Translation Completion with a Translation Size field smaller than the Function's programmed STU value, it shall treat the Translation Completion as if it had completion status UR.

2.3.3. Non-snooped (N) Field

This field is Set to indicate that Read and Write Requests that target memory in the range of this translation must Clear the No Snoop Attribute bit in the Request header. When this field is 0b, the Function is allowed to Set the No Snoop Attribute bit in a Function-specific manner.

Note: When this field is Cleared, the Function is not allowed to Set No Snoop in a Memory Request if the Enable No Snoop field in the Device Control register is Cleared.

The N bit may be cached by the ATC if either R or W is Set.

When U is Set, the meaning of this field is undefined, and the TA may set this field to any value.

2.3.4. Untranslated Access Only (U) Field

This field is Set when the Function is not allowed to access the implied range of memory using a translated address (the range is implied by the untranslated address in the Translation Request and the offset of the translation in the Translation Completion). The Function may use untranslated addresses to access the range as long as the accesses are allowed by the R and W fields. The Function may cache this translation value if either R or W is Set. If the U field is Set, the Translated Address field in the translation is not necessarily a valid memory address and the Function may not use the value in a Read or Write Request with AT set to Translated.

Note: One of the possible uses of this field is to avoid unnecessary invalidations. If a Function uses translated requests for some portions of memory, but not others, then the U field can be used on the portions for which translated requests are not used. When a translation changes if the U field is Set, then it will not necessarily be required that an Invalidate Request be sent to the Function. An example of this use is a Function with a ring buffer that is used for commands. The ring buffer may be allocated for a long period of time and have very high re-use (locality). For this reason, it is useful for the Function to use translated addresses in its memory request that target the command buffer. The same Function might access data buffers that have poor locality and low reuse. Accesses to the data buffers might best be handled by using untranslated Requests. Setting the U field for the data buffer translations ensures that the Function will not attempt to use a translated value to access the data buffer so, when the data buffer mappings are changed, no Invalidation Request is required.

2.3.5. Read (R) and Write (W) Fields

These fields indicate if the returned translation value may be used in a read or write memory request. The ATC may not issue a non-zero read request using the translation value if the R field is Cleared. The ATC may not issue a write request using the translation value if the W field is Cleared. The ATC may not issue any type of request using the translation value if neither the R nor W fields are Set. If both R and W fields are Cleared, the range of the translation is still indicated, but the meaning of the other values in the translation is undefined.

Note: The range of a Translation entry is indicated even if R = W = 0b in order to allow a "hole" in the Translation Completion. For example, if the Translation Request has a Length of six DWORDs, then up to three translations could be included in the Translation Completion. The first and third translations may have Set R or W but the second could have R = W = 0b. To avoid ambiguity about the size of the indicated gap, the range of the gap is indicated in the Translation Completion even if R = W = 0b.

The R = 0b, W = 0b state is used to indicate that the address field in the translation may not be used to form a translated address value for a subsequent request.

When the host changes the translation in the TA, to make the translation present, the host is not required to send an invalidation indication to the ATC so that it will know of the change in state of the translation. Since the ATC may not be notified of changes of the translation, a translation value of R = W = 0b may not be cached.

If no table entry is found for the requested address, the TA will return a CplD with a single translation value with R = W = 0b.

Note: Implementations should not assume that receiving a translation response with the R or W bits Set (independent of the value of the U bit) implies that a subsequent read or write request with the same **untranslated** address will succeed. Although it may be possible for a device and its controlling software to ensure this property, the method for doing so is outside the scope of this specification.

2.4. Completions with Multiple Translations

An ATC is allowed to request that the TA provide translations for a virtually contiguous range of addresses. It does this by setting the Length field in the Translation Request to a value that is two times the number of requested translations as long as the request size (Length * 4) is not larger than either Max_Read_Request_Size or RCB.

If multiple translations are requested, the TA may return one or more translations as long as the number of translations does not exceed the number of requested translations. It is not an error for the TA to return fewer translations than requested and no error indication is sent unless there is an error in accessing the data.

If the Translation Completion contains multiple translations, all translations must have the same indicated size. Also, successive translations must apply to the virtual address range that abuts the previous translation in the same completion.

If a translation has both R = 0b and W = 0b, the TA must still set the Size field and the lower bits of the Translated Address field used to encode the completion size to appropriate values.

Each translation in a Translation Completion will have some overlap with the implied memory range of the Translation Request (see Section 2.2).

A Translation Completion may require one or two CplDs.

If a Translation Completion CplD has a Byte Count that is greater than four times the Length field, then additional CplDs are required to complete the transaction.

If a Translation Completion CplD has a Byte Count that is equal to four times the Length field, then the packet completes the request. For such a CplD, if the sum of Byte Count and Lower Address is not a multiple of RCB, then the CplD is the last of a sequence and it is an error if no previous CplD has been received, and all translation values should be discarded.

Note: There are multiple reasons that the TA may truncate the results of the completion. For example, the request might ask for a range of addresses, not all of which are defined. This could occur if the first translation is valid but located at the end of a page of translations. The TA, in looking up the next page of translations, may find that the page is not valid so the addresses are not valid. The range of addresses that are valid would be returned and no error indicated. When truncating a Translation Completion the TA is not allowed to pad the response with invalid entries (R = 0b, W = 0b).

Note: There are multiple reasons that the TA may break a Translation Completion into multiple TLPs. As an example, if the virtual address of the Translation Completion resolves to a table access that crosses an RCB boundary of the memory system, the completion to the TA may be broken into multiple completions by the memory. Rather than require that the TA accumulate the results, it is allowed to send each portion of the Translation Completion to a Function when it is received from the system memory.

3

3. Invalidation

ATS uses the messages shown in this section to maintain consistency between the TA and the ATC. This specification assumes there is a single TA associated with each ATC. The TA (in conjunction with its associated software) must ensure that the address translations cached in the ATC are not stale by issuing Invalidate Requests.

3.1. Invalidate Request

When a translation is changed in the TA and that translation might be contained within an ATC in a Function, the TA (in conjunction with its associated software) must send an Invalidate Request to the ATC to maintain proper synchronization between the ATPT and the ATC. An Invalidate Request is used to clear a specific subset of the address range from the ATC. Invalidate Requests are constrained to cover power of 2 multiple of 4096-byte pages.

The format of an Invalidate Request is shown in Figure 3-1.

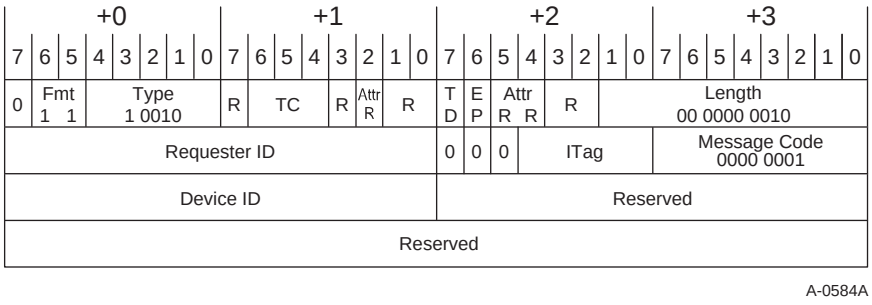

Figure 3-1: Invalidate Request Message

The Invalidate Request is a MsgD transaction with 64 bits of data. Invalidate Request messages may be sent in any TC. The ITag field is constrained to the values 0 to 31 and is used by the TA to uniquely identify requests it issues. A TA must ensure that once an ITag is used, it is not reused until either released by the corresponding Invalidate Completions or by a vendor specific timeout mechanism (see below).

The TA may have a single pool of ITag values for all invalidates that it issues or it may have a pool for each Device ID or any other combination. A Device with multiple ATCs on different Functions must manage the ITags separately for each Requester ID.

The address range specified in an Invalidate Request may span one or more STU 4096-byte pages. Invalidation ranges are required to be naturally aligned and may not be smaller than STU 4096-byte pages. Upon receiving an Invalidate Request with a range less than STU an ATC may either

(1) signal an Unrecognized Request or (2) round the range of the request up to a value greater than or equal to the STU.

IMPLEMENTATION NOTE

Invalidate Completion Timeout

Devices should respond to Invalidate Requests within 1 minute (+50% -0%).Having a bounded time permits an ATPT to implement Invalidate Completion Timeouts and reuse the associated ITag values. ATPT designs are implementation-specific. As such, Invalidate Completion Timeouts and their associated error handling are outside the scope of this specification.

The content of the payload is the untranslated address range to be invalidated. The payload format is shown in Figure 3-2.

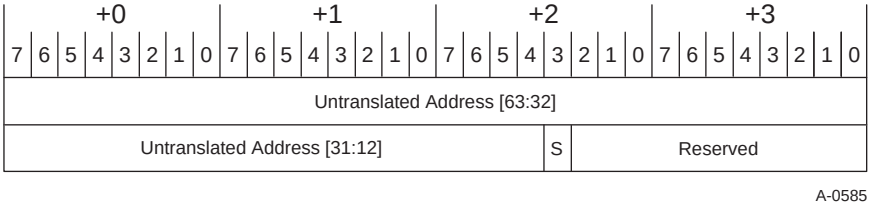

Figure 3-2: Invalidate Request Message Body

The S field is used to indicate if the range being invalidated is greater than 4096 bytes. Its meaning is the same as for the Translation Completion (see Section 2.3.1 and Section 2.3.2).

3.2. Invalidate Completion

When a Function completes an Invalidate operation, it will send one or more Invalidate Completion messages to the TA. These messages must be tagged with information extracted from the Invalidate Request to enable the TA to associate the Invalidate Completions with the Invalidate Request.

The format of the Invalidate Completion message is shown in Figure 3-3.

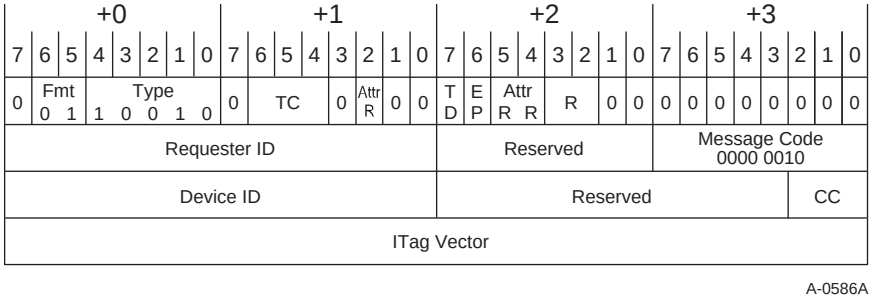

Figure 3-3: Invalidate Completion Message Format

The Invalidate Completion message is a Msg transaction routed by ID. The Requester ID field of the Invalidate Completion message is set to the Requester ID of the Function containing the ATC. The Device ID field of the Invalidate Completion is set to the Requester ID of the TA. The ATC may derive the Requester ID of the TA from the Requester ID field of the corresponding Invalidate Request. Alternatively, since the ATC is only associated with a single TA, the ATC may sample and store the Requester ID from the first Invalidate Request following a Fundamental Reset or FLR. Subsequent Invalidate Completion messages may use this value to set the Device ID field of Invalidate Completion messages.

The Completion Count (CC) field indicates the number of individual Invalidate Completion messages that must be sent for the associated Invalidate Request. Setting the CC field to 0 indicates that eight responses must be sent. The TA is responsible for collecting all the responses associated with a given Tag before considering the corresponding Invalidate Request to be complete.

Invalidate Completion messages may be sent on any TC, independent of the TC the originating Invalidate Request was received. This enables implementations to utilize the Invalidate Completion to push outstanding transactions to the TA to guarantee the required invalidation semantics are met. Implementations that utilize a single upstream TC are required to send a single Invalidate Completion in the utilized TC.

The ITag Vector field is used to indicate which Invalidate Request has been completed. Each of the 32 possible ITag field values from the Invalidation Request is represented by a single bit in the ITag Vector field. The least significant bit (bit 0; i.e., the right-most bit in the schematic representation of the Invalidate Completion message shown in Figure 3-3) of the ITag Vector field corresponds to the ITag field value of 0. The most significant bit (bit 31) of the ITag Vector field corresponds to the ITag field value of 31. Implementations are allowed to coalesce multiple Invalidate Completions by setting multiple ITag Vector bits in a single message provided the following conditions are met:

- The Invalidate Completions flow in the same TC.
- The Invalidate Completions have the same CC value.
- All fragments of an Invalidate Completion must have identical Request ID, CC, and ITag Vector fields.

A TA that receives an Invalidation Completion for an ITag that has no outstanding Invalidation Request shall report this error using implementation specific mechanisms. One possible such mechanism is to report the Invalidation Completion as an Unexpected Completion (UC).

Functions that do not support ATS will treat an Invalidate Request as UR. See the *PCI Express Base Specification* for further details.

Functions supporting ATS are required to send an Invalidate Completion in response to a Invalidate Request independent of whether the Bus Master Enable bit is Set or not. Note that the above conditions must be satisfied even when Bus Master Enable is Cleared. The method for a device to achieve this is implementation dependent.

IMPLEMENTATION NOTE

Bus Master Enable Change

When Bus Master Enable changes from Set to Clear, no further memory requests should be queued. It is possible that queued write requests are present when BME is Cleared. These requests could block an Invalidate Completion. These requests must be either sent or dropped. This will ensure that all outstanding write transactions that are potentially dependent upon the outstanding invalidation are complete.

3.3. Invalidate Completion Semantics

Before an ATC can return an Invalidate Completion for a given Invalidate Request, it must ensure the following conditions are satisfied:

- All new requests initiated by the Function will not utilize stale address translations.
- All outstanding read requests utilizing translated address matching the invalidated range have either completed or been tagged to be discarded (method to discard is implementation specific).
- All outstanding posted writes utilizing a translated address matching the invalidated range have been pushed to the TA. The ATC is required to send a copy of the Invalidate Completion message in each TC in which a posted write has been issued but not known to have been pushed to the TA. The CC field must be set to the same value in each copy of the Invalidate Completion message indicating number of copies sent. The TA is responsible for collecting all sent responses before considering the invalidation to be complete.

IMPLEMENTATION NOTE

Implied TC Flushing

When making the decision as to which TC to send Invalidate Completions, an ATC may infer, in an implementation specific manner, that an issued posted write has been pushed to the TA. For example, a Function that has sent a read transaction to a destination above the TA and received its corresponding response may infer that any preceding posted writes issued in the same TC have been pushed to the TA.

3.4. Request Acceptance Rules

In accord with the request acceptance rules enumerated in the *PCI Express Base Specification*, a Function is not allowed to create a dependency in which the acceptance of a posted transaction is dependent upon the transmission of a posted transaction. Given Invalidate Requests and Invalidate Completions both are posted transactions, Functions must not make the acceptance of an Invalidate Request dependent upon the transmission of an Invalidate Completion. The method for achieving this is implementation specific.

A Function with an ATS capability in its configuration space must be able to accept Invalidate Requests and send Invalidate Completions even if ATS is not enabled.

IMPLEMENTATION NOTE

Invalidate Queue Depth

An ATC is only associated with a single TA. Each TA is limited to a total of 32 outstanding invalidations to any given ATC. This limits the number of outstanding Invalidation Requests active to a single ATC to 32. To avoid a post-to-post dependency, an ATC is required to accept up to 32 Invalidation Requests.

An ATC may choose to implement a maximally sized input queue holding Invalidate Requests. Alternatively, an ATC may choose to implement a maximally sized output queue holding Invalidate Completions. Note that queuing Invalidate Completions requires significantly less state per entry resulting in a potentially more efficient implementation than input queue buffering.

Note that the choice of whether to implement input queuing or output queuing (or a hybrid of both) has no impact on ensuring deadlock free behavior. But implementation choices with regard to queuing may have a significant impact on performance (see Section 3.5).

3.5. Invalidate Flow Control

Due to the variety of caching architectures and queuing strategies, implementations may vary greatly with respect to invalidation latency and throughput. It is possible that a TA may generate Invalidate Requests at a rate that exceeds the average ATC service rate. When this happens, the credit based flow control mechanisms will throttle the TA issue rate. A side effect of this is congestion spreading to other channels and Links through the credit based flow control mechanism. Depending on the frequency and duration of this congestion, performance may suffer. It is highly recommended that TA and its associated software implement higher level flow control mechanisms.

To assist with the implementation of Invalidate Flow Control, an ATC must publish the number of Invalidate Requests it can buffer before back pressuring the Link. This field applies to all invalidations serviced by the Function, independent of the size of the invalidation. This value is communicated in the Invalidate Queue Depth field in the ATS capability structure (see Section 5.1). A value of 0 0000b indicates that invalidate flow control is not necessary to this Function.

IMPLEMENTATION NOTE

Invalidate Flow Control

A Function may indicate that invalidate flow control is not required when one or more of the following is true:

1. The Function can handle invalidations at the maximum arrival rate of Invalidate Requests.
2. The Function will not or very rarely cause Link backpressure (performance loss is negligible).
3. The Function can fully buffer the maximum number of incoming invalidations without back pressuring the Link.

3.6. Invalidate Ordering Semantics

Invalidate Requests and Translation Completions may be sent using different TC and are, therefore, unordered with respect to each other (from the Link's perspective). An ATC must ensure that the proper invalidation behavior is maintained when an Invalidate Request bypasses a Translation Completion to an overlapping region.

An ATC must "snoop" its outstanding translation request queue against all arriving Invalidate Requests. When snooping a request for a N*STU sized translation (N is a power of 2), the ATC must snoop the range of addresses starting at the STU aligned region containing the specified address and ending (N-1) STU size pages later.

If an Invalidate Request overlaps the address range in an outstanding Translation Request, the Translation Request must be tagged as invalid and the results of its corresponding Translation Response must be discarded prior to transmission of the Invalidate Completion. If the Translation Response is received before the Invalidate Completion is sent, an implementation is free to issue requests utilizing the translation result provided the Invalidate Completion Semantics (see Section 3.3) are satisfied.

IMPLEMENTATION NOTE

Request Range Overlap in Invalidations

In the description above, N is the number of STU sized translations that were requested in the Translation Request. This is equal to (Length field in Translation Request)/2.

As an example:

STU is 00 0010b indicating 16384-byte pages.

An outstanding Translation Request has a Length field of 00 0000 0100b indicating two translations covering a range of 32768 bytes.

The high-order 48 bits of the Translation Request are 0000 0FFF FFFFh.

The low-order 16 bits of the address in the request are 11xx xxxx xxxx xxxxb indicating that the translation request covers a range that overlaps a 32768-byte boundary (in fact, the request crosses a 16-TB boundary).

If two translations are returned, they would cover the two STU sized regions at 0000 0FFF FFFF C000h and 0000 1000 0000 0000h.

An Invalidate Request is received with the high-order 48 bits of 0000 1000 0000h and the low-order 16 bits of 0001 1xxx xxxx xxxxb.

The ATC must detect that a translation associated with a portion of the Translation Request is now invalidated and the Translation Completion associated with the invalidated region must be discarded (for simplification, the ATC is allowed to discard all of the Translation Completion).

It should be noted that, processing of the Invalidate Requests is simplified if Translation Requests do not cross alignment boundaries of the request. The Translation Request from the above example is not aligned to a 32768-byte boundary. If it were broken into two requests, it would be simpler to associate the range of the Invalidate Request with the address in the Translation Request. Breaking the Translation Requests into aligned requests is not a requirement.

3.7. Implicit Invalidation Events

The following events will cause the invalidation of all ATC entries:

- Conventional Reset (all forms)
- Function Level Reset
- E field in ATS Capability changes from Clear to Set

No explicit Invalidate Completion message is sent when these implied invalidate events occur.

4

4. Page Request Services

The general model for a page request is as follows:

1. A Function determines that it requires access to a page for which an ATS translation is not available.
2. The Function causes the associated Page Request Interface to send a Page Request Message to its RC. A Page Request Message contains a page address and a Page Request Group (PRG) index. The PRG index is used to identify the transaction and is used to match requests with responses.
3. When the RC determines its response to the request (which will typically be to make the requested page resident), it sends a PRG Response Message back to the requesting Function.
4. The Function can then employ ATS to request a translation for the requested page(s).

A Page Request Message is a PCIe Message Request that is Routed to the Root Complex (see the *PCI Express Base Specification*) with a Message Code of 4 (0000 0100b). The mechanism employed at the RC to buffer requests is implementation specific. The only requirement is that an RC not silently discard requests.

All Page Request Messages and PRG Response Messages travel in PCIe Traffic Class 0. A Page Request Message or PRG Response Message with a Traffic Class other than 0 shall be treated as Malformed TLPs by the RC or endpoint that receives the same. Intermediate routing elements (e.g., Switches) shall not detect this error.

The Relaxed Ordering and ID Base Ordering bits in the Attr field of Page Request Messages and PRG Response messages may be used as defined in the PCI Express Base Specification. The No Snoop bit in the Attr field is reserved.

The page request service allows grouping of page requests into Page Request Groups (PRGs). A PRG can contain one or more page requests. All pages in a PRG are responded to en mass by the host. Individual pages within a PRG are requested with independent Page Request Messages and are recognized as belonging to a common PRG by sharing the same PRG index. The last request of a PRG is marked as such within its Page Request Message. One request credit is consumed per page request (not per PRG).

A PRG Response Message is a PCIe Message Request that is Routed by ID back to the requesting Function. It is used by system software to alert a Function that the page request(s) associated with the corresponding PRG has (have) been satisfied. The page request mechanism does not guarantee any request completion order and all requests are inherently independent of all other concurrently outstanding requests. If a Function requires that a particular request be completed before another request, the initial request will need to complete before the subsequent request is issued. It is valid for a Function to speculatively request a page without ascertaining its residence state and/or to issue multiple concurrently outstanding requests for the same page.

A Page Request Interface is allocated a specific number of page request message credits. An RC (system software) can divide the available credits in any manner deemed appropriate. Any measures the host chooses to employ to ensure that credits are correctly metered by Page Request Interfaces (a Page Request Interface is not using more than its allocation) is an implementation choice. A Page Request Interface is not allowed to oversubscribe the available number of requests (doing so can result in the page request mechanism being disabled if the buffer limit is exceeded at the root). A Page Request Interface's page request allocation is static. It is determined when the Page Request Interface is enabled and can only be changed by disabling and then re-enabling the interface.

4.1. Page Request Message

A Function uses a Page Request Message to send page requests to its associated host. A page request indicates a page needed by the Function. The Page Request Interface associated with a Function is given a specific Page Request allocation. A Page Request Interface shall not issue page requests that exceed its page request allocation.

A page request contains the untranslated address of the page that is needed, the access permissions needed for that page, and a PRG index. A PRG Index is a 9-bit scalar that is assigned by the Function to identify the associated page request. Multiple pages may be requested using a single PRG index. When more than a single page is to be associated with a given PRG, the Last flag in the Page Request Record is cleared in all the requests except the last request associated with a given PRG (the flag is set in the last request). Page requests are responded to en mass. No response is possible (except for a Response Failure error) until the last request of a PRG has been received by the root. The number of PRGs that a Function can have outstanding at any given time is less than or equal to the associated Page Request Interface's Outstanding Page Request Allocation. It is valid for a request group to contain multiple requests for the same page and for multiple outstanding PRGs to request the same page.

The first two DWORDs of a Page Request Message contain a standard PCIe message header. The second two DWORDs of the message contain page request specific data fields.

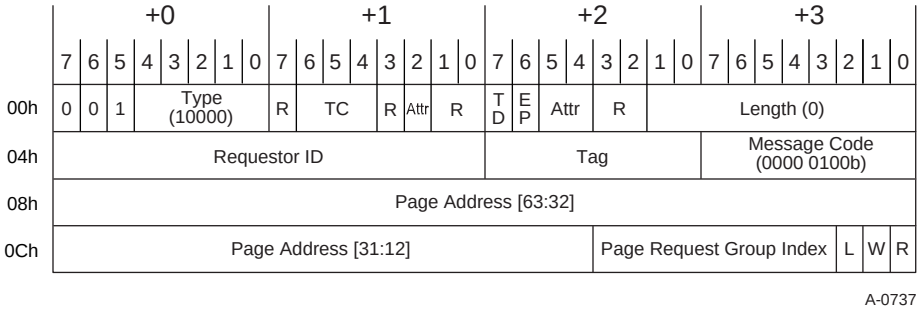

Figure 4-1: Page Request Message

Table 4-1: Page Request Message Data Fields

Field	Meaning
R	**Read Access Requested** – This field, when Set, indicates that the requesting Function seeks read access to the associated page. When Clear, this field indicates that the requesting Function will not read the associated page.
W	**Write Access Requested** – This field, when Set, indicates that the requesting Function seeks write access and/or zero-length read access to the associated page. When Clear, this field indicates that the requesting Function will not write to the associated page.
L	**Last Request in PRG** – This field, when Set, indicates that the associated page request is the last request of the associated PRG. A PRG can have a single entry, in which case the PRG consists of a single request in which this field is Set. When Clear, this field indicates that additional page requests will be posted using this record's PRG Index.
Page Request Group Index	**Page Request Group Index** – This field contains a Function supplied identifier for the associated page request. A Function need not employ the entire available range of PRG index values. A host shall never respond with a PRG Index that has not been previously issued by the Function and that is not currently an outstanding request PRG Index (except when issuing a Response Failure, in which case the host need not preserve the associated request's PRG Index value in the error response).
Page Address	**Page Address** – This field contains the untranslated address of the page to be loaded. For pages larger than 4096 bytes, the least significant bits of this field are ignored. For example, the least significant bit of this field is ignored when an 8096-byte page is being requested.

IMPLEMENTATION NOTE

Last Bit and Relaxed Ordering

If multiple page requests are associated with a single PRG index, the last page request of a PRG should have the Relaxed Ordering attribute bit Clear in addition to having the Last flag Set. All other page request messages may have the Relaxed Ordering attribute bit set to any value.

4.2. Page Request Group Response Message

System hardware and/or software communicate with a Function's page request interface via PRG Response Messages. A PRG Response Message is used by a host to signal the completion of a PRG, or the catastrophic failure of the interface. A single PRG Response Message is issued in response to a PRG, independent of the number of page requests associated with the PRG. There is no mechanism for indicating a partial request completion or partial request failure. If any of the pages associated with a given PRG cannot be satisfied, then the request is considered to have failed and the reason for the failure is supplied in the PRG Response Message. The host has no obligation to partially satisfy a multi-page request. If one of the requested pages cannot be made resident, then

the entire request can, but need not, be discarded. That is, the residence of pages that share a PRG with a failed page request, but that are not associated with the failure, is indeterminate from the Function's perspective.

There are three possible page request failures:

1. The requested page is not a valid untranslated address or both R and W fields are clear in the page request.
2. The requested page does not have the requested access attributes.
3. The system is, for an unspecified reason, unable to respond to the request. This response is terminal (the host may no longer respond to any page requests and may not supply any further replies to the Function until the Function's page request interface has been reset). For example, a request that violates a Function's assigned request limit or overflows the RC's buffering capability may cause this type of failure.

A Function's response to page request failure cases 1 and 2 above is implementation dependent. A failure is not necessarily persistent, that is, a failed request may, in some instances succeed if re-issued. The range of possibilities precludes the precise specification of a generalized failure behavior, though on a per Function basis, the response to a failure will be an implementation dependent behavior.

All responses are sent to their associated Functions via PRG Response Messages. A Function must be capable of sinking multiple consecutive messages without losing any information. To avoid deadlock, a Function must able to process PRG Response Messages for all of the Function's outstanding Page Request Messages without depending on the Function sending or receiving any other TLP.[5] A PRG Response Message is an ID routed PCIe message. The only Page Request Interface specific fields in this message are the Response Code and PRG. All other fields are standard PCIe message fields. (Note: these messages are routed based on the ID in bytes 8 and 9; with bytes 4 and 5 containing the host's RID.)

Receipt of a PRG Response Message that contains a PRG Index that is not currently outstanding at a Function shall result in the UPRGI flag in the PRI Extended Capability being Set and in the issuance of an Unexpected Response (UR) by the Function containing the PRI Extended Capability. With the exception of setting the UPRGI flag, a Function treats receipt of an unexpected PRG Index in exactly the same manner that it treats receipt of a standard PCIe read completion for which there is no outstanding request.

[5] For example, processing a PRG Response Message that causes the Function to send a TLP upstream must not block processing of subsequent downstream TLPs even if the upstream TLP is delayed by flow control.

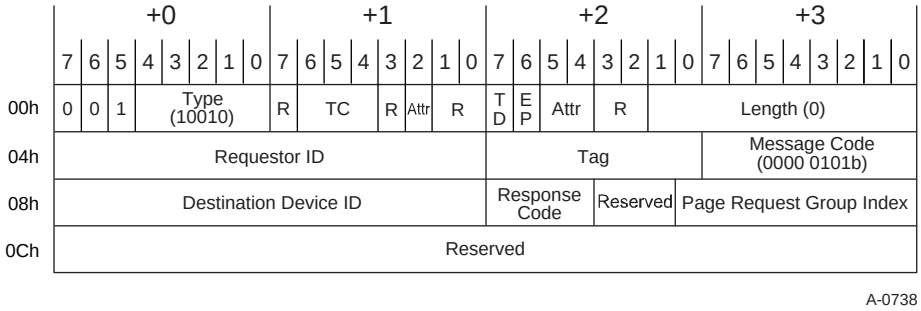

Figure 4-2: PRG Response Message

Table 4-2: PRG Response Message Data Fields

Field	Meaning
Page Request Group Index	**Page Request Group Index** – This field contains a Function supplied index to which the RC is responding. A given PRG Index will receive exactly one response per instance of PRG (with the possible exception of a Response Failure).
Response Code	**Response Code** – This field contains the response type of the associated PRG. The encodings are presented in Section 4.2.1.

4.2.1. Response Code Field

The values and meaning for the Response Code field are listed in Table 4-3.

Table 4-3: Response Codes

Value	Status	Meaning
0000b	Success	All pages within the associated PRG were successfully made resident.
0001b	Invalid Request	One or more pages within the associated PRG do not exist or requests access privilege(s) that cannot be granted. Unless the page mapping associated with the Function is altered, re-issuance of the associated request will never result in success.
1110b:0010b	Unused	Unused Response Code values. A Function receiving such a message shall process it as if the message contained a Response Code of Response Failure.
1111b	Response Failure	One or more pages within the associated request group have encountered/caused a catastrophic error. This response disables the Page Request Interface at the Function. Any pending page requests for other PRGs will be satisfied at the convenience of the host. The Function shall ignore any subsequent PRG Response Messages, pending re-enablement of the Page Request Interface.

5

5. Configuration

5.1. ATS Extended Capability Structure

Each Function that supports ATS must have the ATS Extended Capability structure in its extended configuration space.

Figure 5-1 details allocation of the register fields in the ATS Extended Capability structure.

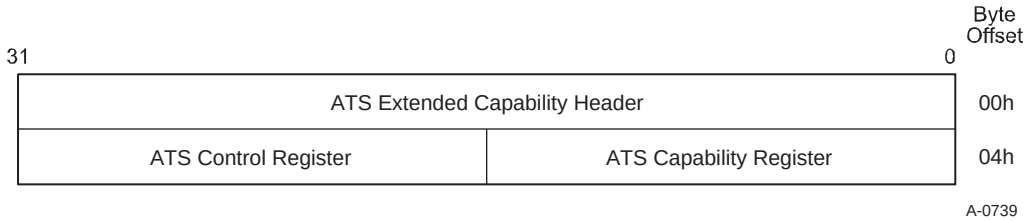

Figure 5-1: ATS Extended Capability Structure

5.1.1. ATS Extended Capability Header

Figure 5-2 details allocation of the register fields in the ATS Extended Capability header; Table 5-1 provides the respective field definitions.

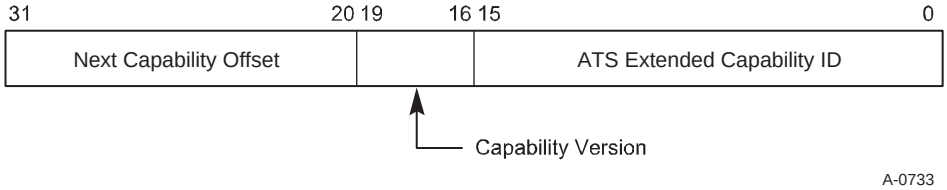

Figure 5-2: ATS Extended Capability Header

Table 5-1: ATS Extended Capability Header

Bit Location	Register Description	Attributes
15:0	**ATS Extended Capability ID** – Indicates the ATS Extended Capability structure. This field must return a Capability ID of 000Fh indicating that this is an ATS Extended Capability structure.	RO
19:16	**Capability Version** – This field is a PCI-SIG defined version number that indicates the version of the Capability structure present. Must be 1h for this version of the specification.	RO
31:20	**Next Capability Offset** – The offset to the next PCI Extended Capability structure or 000h if no other items exist in the linked list of capabilities.	RO

5.1.2. ATS Capability Register

Figure 5-3 details the allocation of register fields of an ATS Capability register; Table 5-2 provides the respective bit definitions.

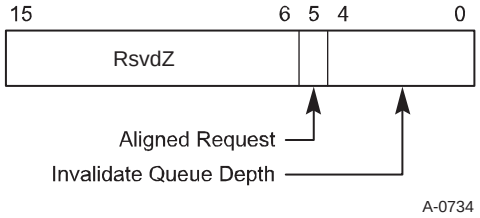

Figure 5-3: ATS Capability Register

Table 5-2: ATS Capability Register

Bit Location	Register Description	Attributes
4:0	**Invalidate Queue Depth** – The number of Invalidate Requests that the Function can accept before putting backpressure on the upstream connection. If 0 0000b, the Function can accept 32 Invalidate Requests.	RO
5	**Page Aligned Request** – If Set, indicates the Untranslated Address is always aligned to a 4096 byte boundary. Setting this bit is recommended. This bit permits software to distinguish between implementations compatible with earlier version of this specification that permitted a requester to supply anything in bits [11:2].	RO

5.1.3. ATS Control Register

Figure 5-4 details the allocation of register fields of an ATS Control register; Table 5-3 provides the respective bit definitions.

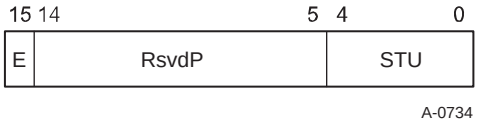

Figure 5-4: ATS Control Register

Table 5-3: ATS Control Register

Bit Location	Register Description	Attributes
4:0	**Smallest Translation Unit (STU)** – This value indicates to the Function the minimum number of 4096-byte blocks that is indicated in a Translation Completions or Invalidate Requests. This is a power of 2 multiplier and the number of blocks is 2^{STU}. A value of 0 0000b indicates one block and a value of 1 1111b indicates 2^{31} blocks (or 8 TB total) Default value is 0 0000b.	RW
15	**Enable (E)** – When Set, the Function is enabled to cache translations. Default value is 0b.	RW

5.2. Page Request Extended Capability Structure

A Page Request Extended Capability Structure is used to configure the Page Request Interface mechanism. A Multi-Function Device may implement a Page Request Interface and the associated capability on any Function. For SR-IOV, a single Page Request Interface is permitted for the PF and is shared between the PF and the associated VFs. The PF implements this capability and the VFs do not. Every Page Request Interface mechanism operates independently.

Note: For SR-IOV, even though the Page Request Interface is shared between PFs and VFs, it sends the requesting Function's ID (PF or VF) in the Requester ID field of the Page Request Message and expects the requesting Function's ID in the Destination Device ID field of the resulting PRG Response Message.

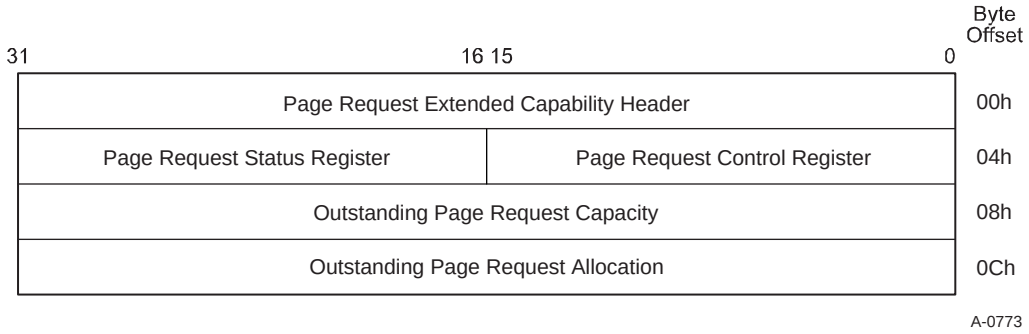

Figure 5-5: Page Request Extended Capability Structure

5.2.1. Page Request Extended Capability Structure

Figure 5-6 details allocation of the register fields in the Page Request Extended Capability header; Table 2-1 provides the respective field definitions.

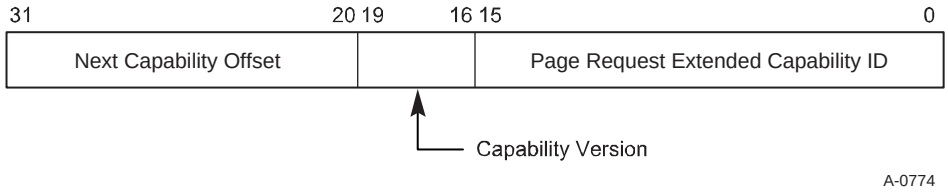

Figure 5-6: Page Request Extended Capability Header

Table 5-4: Page Request Extended Capability Header

Bit Location	Register Description	Attributes
15:0	**Page Request Extended Capability ID** – Indicates that the associated extended capability structure is a Page Request Extended Capability. This field must return a Capability ID of 0013h.	RO
19:16	**Capability Version** – This field is a PCI-SIG defined version number that indicates the version of the Capability structure present. Must be 1h for this version of the specification.	RO
31:20	**Next Capability Offset** – The offset to the next PCI Extended Capability structure or 000h if no other items exist in the linked list of capabilities.	RO

5.2.2. Page Request Control Register (04h)

Figure 5-7 details allocation of the register fields in the Page Request Control Register; Table 5-5 provides the respective field definitions.

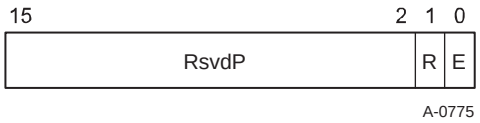

Figure 5-7: Page Request Control Register

Table 5-5: Page Request Control Register

Bit Location	Register Description	Attributes
0	**Enable (E)** – This field, when set, indicates that the Page Request Interface is allowed to make page requests. If this field is Clear, the Page Request Interface is not allowed to issue page requests. If both this field and the Stopped field are Clear, then the Page Request Interface will not issue new page requests, but has outstanding page requests that have been transmitted or are queued for transmission. When the Page Request Interface is transitioned from not-Enabled to Enabled, its status flags (Stopped, Response Failure, and Unexpected Response flags) are cleared. Enabling a Page Request Interface that has not successfully Stopped has indeterminate results. Default value is 0b.	RW
1	**Reset (R)** – When the Enable field is clear, or is being cleared in the same register update that sets this field, writing a 1b to this field, clears the associated implementation dependent page request credit counter and pending request state for the associated Page Request Interface. No action is initiated if this field is written to 0b or if this field is written with any value while the Enable field is Set. Reads of this field return 0b	RW

5.2.3. Page Request Status Register (06h)

Figure 5-8 details allocation of the register fields in the Page Request Error Register; Table 5-6 provides the respective field definitions.

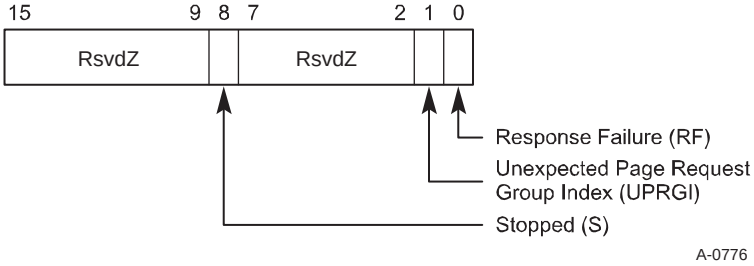

Figure 5-8: Page Request Status Register

Table 5-6: Page Request Error Register

Bit Location	Register Description	Attributes
0	**Response Failure (RF)** – This field, when Set, indicates that the Function has received a PRG Response Message indicating a Response Failure. The Function expects no further responses from the host (any received are ignored). This field is Set by the Function and Cleared when a one is written to the field. For SR-IOV, this field is Set in the PF if any associated Function (PF or VF) receives a PRG Response Message indicating Response Failure. Default value is 0b.	RW1C
1	**Unexpected Page Request Group Index (UPRGI)** – This field, when Set, indicates that the Function has received a PRG Response Message containing a PRG index that has no matching request. This field is Set by the Function and cleared when a one is written to the field. For SR-IOV, this field is Set in the PF if any associated Function (PF or VF) receives a PRG Response Message that does has no matching request. Default value is 0b.	RW1C

Bit Location	Register Description	Attributes
8	**Stopped (S)** – When this field is Set, the associated page request interface has stopped issuing additional page requests and that all previously issued Page Requests have completed. When this field is Clear the associated page request interface either has not stopped or has stopped issuing new Page Requests but has outstanding Page Requests. This field is only meaningful if Enable is Clear. If Enable is Set, this field is undefined. When the Enable field is Cleared, after having been previously Set, the interface transitions to the stopping state and Clears this field. After all page requests currently outstanding at the Function(s) have completed, this field is Set and the interface enters the disabled state. If there were no outstanding page requests, this field may be Set immediately when Enable is Cleared. Resetting the interface will cause an immediate transition to the disabled state. While in the stopping state, receipt of a Response Failure message will result in the immediate transition to the disabled state (Setting this field). For SR-IOV, this field is Set only when all associated Functions (PF and VFs) have stopped issuing page requests. Default value is 1b.	RO

5.2.4. Outstanding Page Request Capacity (08h)

This register contains the number of outstanding page request messages the associated Page Request Interface physically supports. This is the upper limit on the number of pages that can be usefully allocated to the Page Request Interface.

This register is Read Only.

5.2.5. Outstanding Page Request Allocation (0Ch)

This register contains the number of outstanding page request messages the associated Page Request Interface is allowed to issue (have outstanding at any given instance).

The number of PRGs a Page Request Interface has outstanding is less than or equal to the number of request messages it has issued. For example, if system software allocates 1000 messages to a Page Request Interface then a single PRG could use all 1000 of the possible requests. Conversely, at one request per PRG the Page Request Interface would run out of PRG indices (of which there are only 512) before it consumes all its page request credits. A Page Request Interface must pre-allocate its request availability for any given PRG, that is, all the requests required by a given PRG must be available before any of the requests may be issued.

This register is Read/Write. Behavior is undefined if this register is changed while the Enable flag is set. Behavior is undefined if this register is written with a value larger than Outstanding Page Request Capacity. Default value is 0.

Acknowledgements

The following persons were instrumental in the development of the ATS Specification:[6]

Antonio Asaro, Advanced Micro Devices, Inc.

Steve Glaser, Advanced Micro Devices, Inc.

Lucian Gozu, Neterion Corporation

Andrew Gruber, Advanced Micro Devices, Inc.

Mark Hummel, Advanced Micro Devices, Inc.

David Kahn, Sun Microsystems

Michael Krause, Hewlett-Packard Corporation

Brian Langendorf, Nvidia Corporation

David Mayhew, Advanced Micro Devices, Inc.

Renato Recio, IBM Corporation

Rajesh Sankaran, Intel Corporation

David Wooten, Microsoft Corporation

William Wu, Broadcom Corporation

[6] Company affiliation listed is at the time of specification contributions.