Using Non-transparent Bridging in PCI Express Systems

By Jack Regula

Using Non-transparent Bridging in PCI Express Systems

Introduction

Distributed systems are gaining popularity as they fill the need of next generation systems. Multihost systems provide not only the ability to increase processing bandwidth, but also allow greater system reliability through host failover. These features are becoming increasingly important, especially in next generation storage and communications devices.

The PCI Express specification has been silent with regards to implementing multi processor systems. Because of this, many have assumed that distributed processing cannot be implemented using PCI Express. This, of course, is incorrect; given that PCI Express is software compatible with PCI, and PCI systems have long implemented distributed processing.

PCI was originally designed as an interconnect for personal computers; because of the nature of PCs at that time, the protocol architects did not anticipate the need for multiprocessors. Therefore, they designed the system assuming that the host processor would enumerate the entire memory space. Obviously, if another processor is added, the system operation would fail as both processors would attempt to service the system requests.

This paper outlines how to implement multiprocessor systems using industry standard practices established in the PCI paradigm..

Introduction to Non-transparent Bridging

The use of non-transparent bridges in PCI systems to support intelligent adapters in enterprise systems and multiple processors in embedded systems is well established. The Intel (ne' DEC) DrawBridge established the paradigm of the embedded bridge and became a defacto standard in such environs as Compact PCI and intelligent adapters for enterprise systems. In these systems, the non-transparent bridge functions as a gateway between the local subsystem and the backplane.

Such applications can be ported to PCI Express by the use of non-transparent bridges, either between a local PCI or PCI-X bus and the PCI Express interconnect or with the non-transparent bridge integrated into a PCI Express switch in place of one of the transparent bridges that are part of a PCI Express switch's software model. As discussed herein, a PCI Express Switch with a single port configurable to operate in either transparent or non-transparent bridge mode supports both intelligent adapter, dual-host, and dual-star fabric usage models.

Non-transparent bridges isolate intelligent subsystems from each other by masquerading as endpoints to discovery software and translating the addresses of transactions that cross the bridge. A non-transparent PCI – PCI Bridge, or PCI Express to PCI Express Bridge, exposes a Type 0 CSR header on both sides to terminate discovery and forwards

transactions from one side to the other with address translation, using the BARs of those CSR headers.

A non-transparent bridge is functionally similar to a transparent bridge in that both provide a path between two independent PCI buses (or PCI or PCI Express busses). The key difference is that when a non-transparent bridge is used, devices on the downstream side (relative to the system host) of the bridge are not visible from the upstream side. This allows an intelligent controller on the downstream side to manage devices there, making them appear as a single controller to the system host. The path between the two buses allows the devices on the downstream side to transfer data directly to the upstream side of the bus without directly involving the intelligent controller in the data move. Thus transactions are forwarded across the bus unfettered just as in a P2P Bridge, but the resources responsible are hidden from the host, which sees a single device.

A non-transparent bridge can also be used to link a secondary host with the hierarchy of a primary host. It provides isolation while allowing communications between the two systems. A non-transparent bridge typically includes doorbell registers to send interrupts from each side of the bridge to the other and scratchpad registers accessible from both sides for inter-processor communications. Upon failure of the primary host, the non-transparent bridge resources allow access by the secondary host to reconfigure the system so that it can take over as host.

The following subsections contrast the two types of bridges with the point of view suggested by the subsection headings.

Device Identification and Transparent/Non-transparent Mode Control

Devices identify themselves via the Class Code register of the standard CSR header. A transparent PCI-PCI bridge uses a Class Code 060400h. A non-transparent bridge identifies itself as a RAM controller via a Class Code of 050000h. This identification reflects the fact that its typical use is to map memory space, containing memory rather than memory mapped I/O, from one address domain into another.

PCI Express Capabilities registers include a Device Port Type field. In these registers, a transparent bridge/switch port would identify itself as either an upstream or a downstream port while a non-transparent bridge/switch port would identify itself as an endpoint. The transparent/non-transparent mode of a switch as well as its upstream/downstream property is controlled via writes to this register via the memory-mapped view of the switch CSRs (see CSR BAR). This writes are sticky so that any configuration thus established survives reset. In standard CSR space, this register field behaves as RO per the PCI Express Base specification.

CSR Header

The transparent bridge uses a Type 1 CSR header. The non-transparent bridge uses a Type 0 CSR header and exposes a separate such header on each side of the bridge.

Discovery software running on each side of the bridge sees an endpoint with BARs defining tunnels into the other's memory space.

Reset Propagation

The Secondary Bus Reset bit in the Bridge Control register of the transparent bridge provides a mechanism for propagating reset in the downstream direction to devices on the other side of the bridge.

In a non-transparent bridge, such a reset would not be propagated across the bridge. A sideband signal mechanism can be provided to allow propagation of a hot reset from the system host to the entire local subsystem for the intelligent adapter usage model. When a hot reset is received at a non-transparent bridge, an external pin can be asserted. This can be connected to the local root complex and used there to drive reset down into the entire local hierarchy.

The detailed effects of a local host reset on the non-transparent bridge/switch port are discussed in subsequent sections.

Scratchpad Registers

Scratchpad registers are both readable and writeable from both sides of the non-transparent bridge, providing a generic means for interprocessor communications. A block of such registers, typically eight, is provided. They can be accessed in either memory or I/O space from both the primary and secondary interfaces of the bridge. They can pass control and status information between primary and secondary bus devices or they can be generic R/W registers. Writing or reading a scratchpad register does not cause an interrupt to be asserted. Doorbell interrupts can be used for this purpose.

Doorbell Registers

Doorbell registers are used to send interrupts from one side of the non-transparent bridge to the other. The following text describes a typical set of doorbell control registers.

A 16-bit software controlled interrupt request register and an associated 16-bit mask register is implemented for each interface (primary and secondary). These registers can be accessed from the primary or the secondary interface of the Bridge in either memory or IO space. The doorbell mechanisms consist of the following register set:

- Primary IRQ Status Register
- Primary IRQ Request Register
- Primary IRQ Set Mask Register
- Primary IRQ Clear Mask Register
- Secondary IRQ Status Register
- Secondary IRQ Request Register
- Secondary IRQ Set Mask Register
- Secondary IRQ Clear Mask Register

An Interrupt is asserted on the Primary interface whenever one or more of the bits in the IRQ Request Register are set and their corresponding mask bits are zero. The interrupt is asserted as long as this condition exists. The secondary works identically. The interrupt is deasserted when each the asserted bit is either masked or cleared. In a PCI Express switch, the interrupt state transitions from asserting to de-asserting or vice-versa result in packets being sent upstream on the appropriate side of the bridge. Standard PCI Express capability structures allow these interrupts to be configured as INTx or MSI.

The Status/Request registers are internally the same register, they just have two interfaces, one location is used to set bits and the other location is used to clear bits. The status can be read from either register. The Set/Clear Mask registers are also internally the same register. One interface is used to set a mask bit, the other is used to clear a mask bit.

Transaction Forwarding with Address Translation

The transparent bridge uses base and limit registers in each of I/O space, non-prefetchable memory space, and prefetchable memory space to map transactions in the downstream direction across the bridge. All downstream devices are required to be mapped in contiguous address regions such that a single aperture in each space is sufficient. Upstream mapping is done via inverse decode relative to the same registers. A transparent bridge does not translate the addresses of forwarded transactions/packets.

The non-transparent bridge uses the standard set of BARs in its Type 0 CSR header to define resource apertures that allow the forwarding of transactions to the opposite (other side) interface. There are two sets of BARs, one on the Primary side, one on the Secondary. For each BAR on each side of the bridge there exists a set of associated control and setup registers writable usually from the other side of the bridge. Each BAR has a “setup” register, which defines the size and type of its aperture, and an address translation register. Some bars also have a limit register that can be used to restrict its aperture’s size to less than a power of two. These registers need to be programmed prior to allowing access from outside the local subsystem. This is typically done by software running on a local processor.

In PCI Express, the Device ID fields of packets passing through these apertures are also translated to support Device ID routing.

The transparent bridge forwards CSR transactions in the downstream direction according to the secondary and subordinate bus number registers, converting Type 1 CSRs to Type 0 CSRs as required. The non-transparent bridge accepts only those CSR transactions addressed to it.

BAR Setup Registers

All upstream and downstream BARS have programmable window sizes, with the exception of BAR0 & BAR1 (on both interfaces), which provide memory and/or IO mapped access to the CSRs. The BAR Setup registers are used to program the window

size of each BAR (Note BAR2/3 (primary only) and BAR4/5 can be programmed to be 64bit BARs).

Direct Address Translation

The addresses of all upstream and downstream transactions are translated (except BARs accessing CSRs). With the exception of the cases in the following two sections, addresses that are forwarded from one interface to the other are translated by adding a Base Address to their offset within the BAR that they landed in. The BAR Base Translation Registers are used to setup these base translations for the individual BARs.

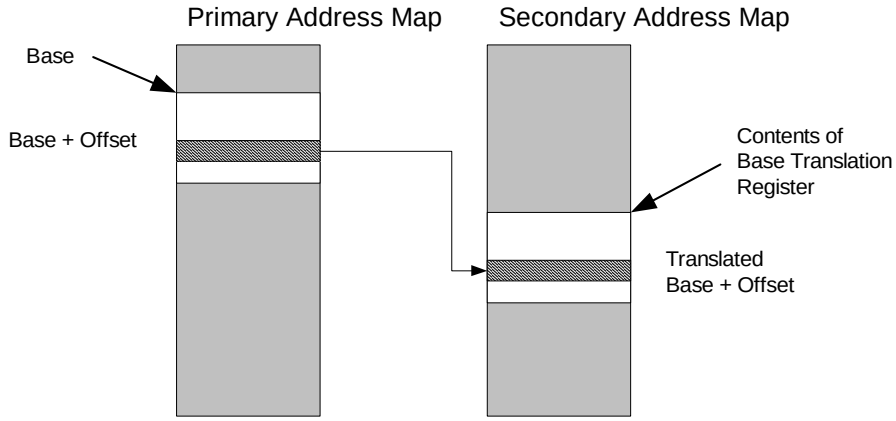

Figure 1 Direct Address Translation

Lookup Table Based Address Translation

On the secondary side, BAR3 uses a special lookup table based address translation for transactions that fall inside its window. The lookup table provides more flexibility in secondary bus local addresses to primary bus addresses. The location of the index field with the address bus is programmable to adjust aperture size.

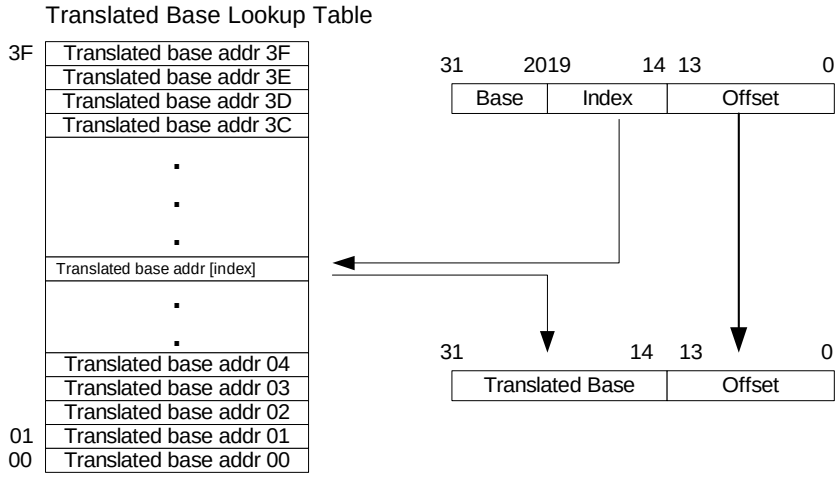

Figure 2 Lookup Table Based Translation

Downstream BAR limit registers

The two downstream BARs on primary side (BAR2/3 and BAR4/5) also have Limit registers, programmable from the local side, to further restrict the size of the window they expose. BARs can only be assigned Memory resources in "power of two" granularity. The limit registers provide a means to obtain better granularity by "capping" the size of the BAR within the "power of two" granularity. Only transactions below the Limit registers are forwarded to the secondary bus. Transactions above the limit are discarded or return 0xFFFFFFFF, or a master abort equivalent packet, on reads.

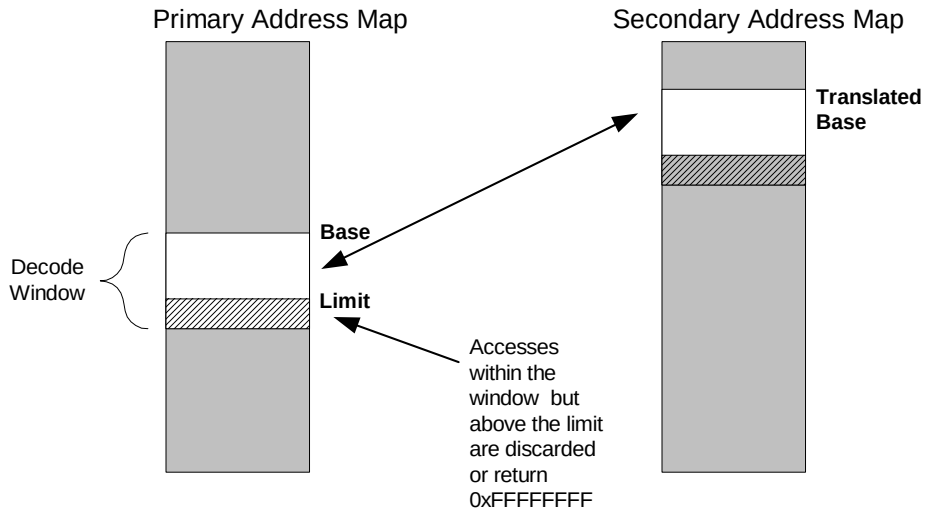

Figure 3 Use of Limit Register

Forwarding 64bit Address Memory Transactions

Certain BARs can be configured to work in pairs to provide the base address and translation for transactions containing 64-bit addresses. Transactions that hit within these 64-bit BARs are forwarded using direct address translation. The Base Register and Base Register Upper pair determine the lower limit or base address of the window into system space. The Setup and Setup Upper register pair determine its size. The contents of the Base Translation and Base Translation Upper Registers replace bits of like weight in the address being translated under the mask provided by the setup register pair.

Replacement under mask address translation constrains the base address to an integer multiple of its size. Suppose the system's physical memory space extends from 0 to 512 GBytes. A 512 GByte window is created in local space and translated to zero and up in system space. Translating the base address down to zero in the target address space allows the window, if appropriately sized, to span the entire target physical memory space. Clearly, a 64-bit address space is large enough to cover any even remotely practical size of physical memory in this way. The address translation process is shown in Figure 4.

A 64-bit BAR pair on the system side of the bridge is used to translate a window of 64-bit addresses in packets originated on the system side of the bridge down below 2^{32} in local space.

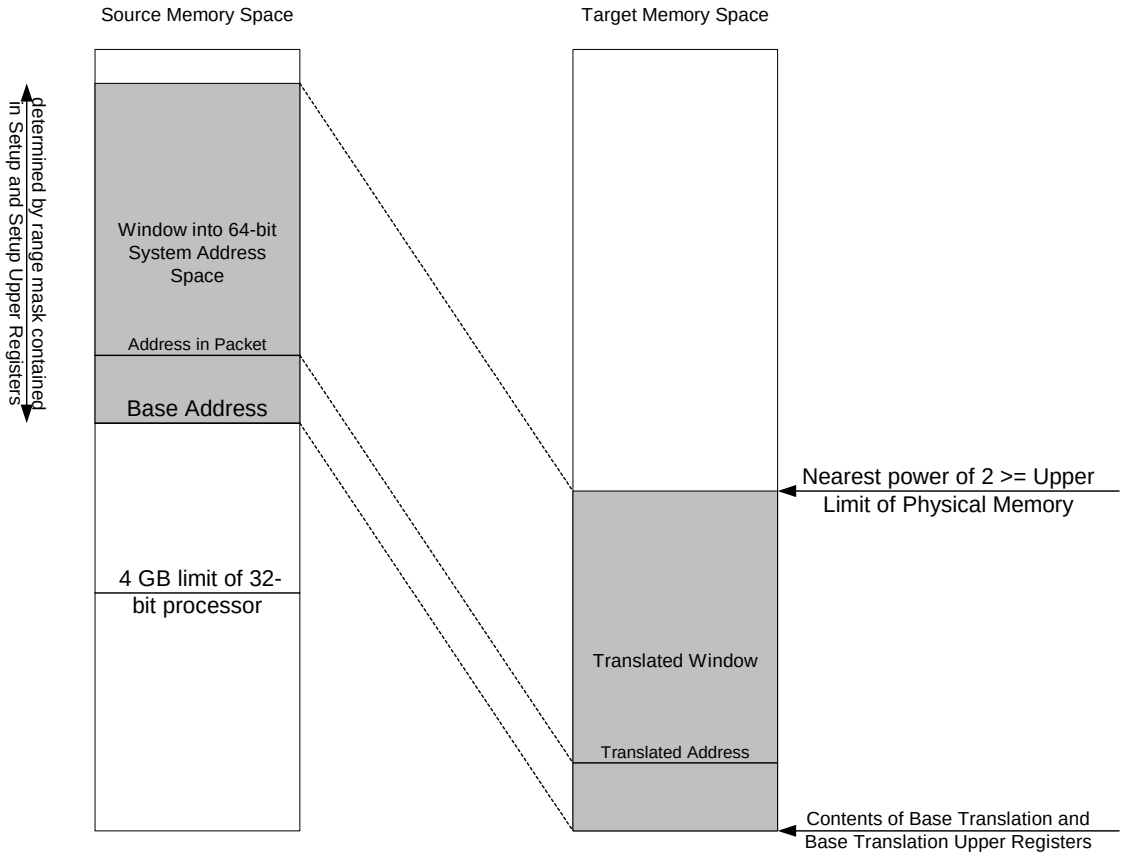

Figure 4 Translation of 64-bit Addresses

CSR Access Enable Control

A transparent bridge is configurable only from its upstream side. Its CSRs are always accessible, except during EEPROM load, if any, when they receive a configuration retry response.

The non-transparent bridge has CSRs accessible from both sides with special control register bits used to determine which sides are permitted to access the CSRs. Like the transparent bridge, CSR accesses from either side received configuration retry response during EEPROM load. The control bits are:

- Secondary Bus Access Enable
- Primary Bus Access Enable

In Intelligent Adapter applications, Secondary Bus Access Enable is asserted after EEPROM load to allow the local processor to complete its configuration of the local subsystem before the system host is permitted to complete discovery and configuration.

In dual-host applications, both Secondary Bus Access Enable and Primary Bus Access Enable are asserted after EEPROM load, or by default, if no EEPROM. This is necessary to avoid a vulnerability to a dead or missing primary host at power-up. The dual-host software typically implements a handshake protocol in software to ensure that configuration from each side occurs in the proper order.

One of the BARs on each side can be used to access the CSRs via memory mapped I/O transactions. Certain registers in the bridge used for failover are writable only via the memory mapped view. Protection can be implemented via the BAR's enable bits.

Requester ID Translation

The non-transparent bridge port must either take ownership of non-posted requests passing through it or translate their Requester IDs in such a way as to ensure that their completions can be routed back to their requesters.

The Transaction ID consists of the Requester ID plus a tag, which is similar to a sequence number. The Requester ID consists of the device's bus number, device number, and function number. Were the non-transparent bridge port to take ownership of a transaction it would substitute the bus number and device number associated with the Type 0 CSR header on the PCI Express side of the bridge for those in the bridged packet and replace the tag with one assigned by itself. It would then store the original Requester ID and tag in an array indexed by the assigned tag, or associated with it. When the completion returned, the non-transparent port would retrieve the original transaction ID from the array and use it to restore the Requester ID field in the completion to that in the original request.

The operation described above is moderately complex and requires use of a finite resource, which might become a performance limitation. We have found that it is sufficient to translate only the Requester ID, avoiding the complexity and potential performance constraint associated with taking ownership. When translating just the Requester ID, it is not required to save state for each outstanding request.

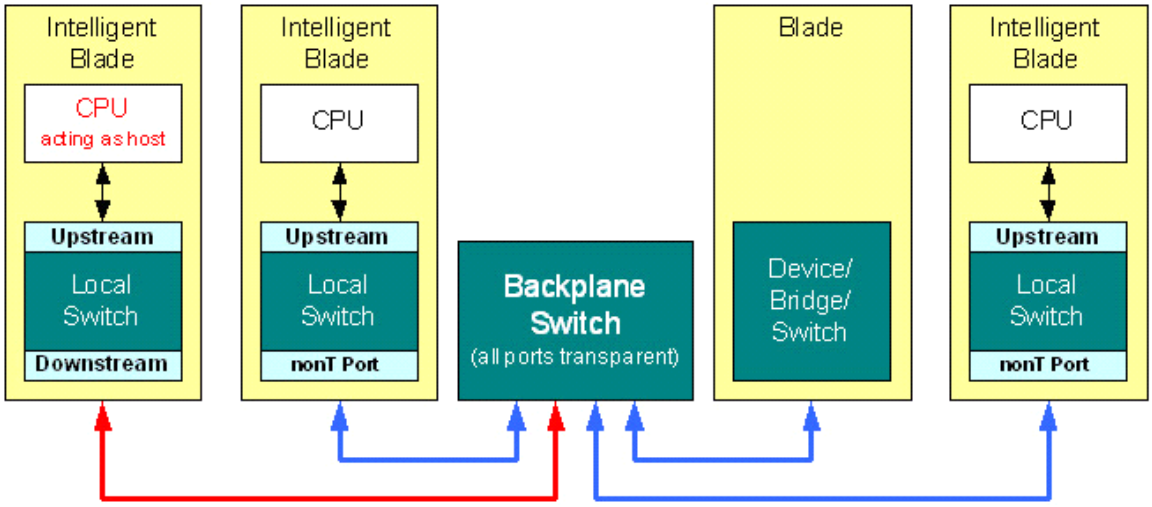

Figure 5 System Model for ID Translation

Figure 5 shows the system model from which the Requester ID translation algorithm requirements have been derived – that devices or software agents on any module in the figure must be able to communicate with their counterparts on any other module. Up to four translation steps occur in the routing of a request from a device behind a non-transparent switch port, across a PCI Express fabric, and to a completer (e.g. memory) behind a second non-transparent switch port and the subsequent routing of a completion in the reverse direction. If the requester or the completer is not behind a non-transparent switch port, as would be the case for a device in the Root Complex, two of those steps are avoided. Each of the following cases is described and illustrated in the following subsections. The first two cases are a subset of the third case.

- Requester only behind non-transparent port
- Completer only behind non-transparent port
- Requester and completer each behind different non-transparent ports

Requester Only Behind Non-transparent Port

Figure 6 shows the Requester ID translation that occurs when a non-posted request is sent from a device behind a non-transparent switch port to a location in what might be called system or host space. An example is the reading of host memory by a device on an intelligent adapter.

Translation of an Outgoing Request

The translation of outgoing requests uses an 8-entry CAM. Each CAM entry supports all the outgoing requests and any number of outstanding requests made by a single {Device, Function}. If a device uses phantom function numbers to increase the maximum number of outstanding transactions, then each phantom function consumes a CAM entry. The CAM must be configured, either by EEPROM or local firmware, before it is possible to send requests to the system domain. This provides a measure of security/protection. CAM entries are never retired but may be changed by software, although it is unlikely that software would need to make a change, except during the development process.

When an outgoing request arrives at a non-transparent port from the switch's virtual bus side, the packet's Requester ID is associated into this CAM. If it hits, the corresponding TxIndex is inserted into the Function Number field of the packet's Requester ID. If it misses, an Unsupported Request completion is returned. At the same time, the contents of the non-transparent port's link side Bus Number and Device Number Capture Registers, the values last used during the last CSR write to the port, are copied into the bus number and device number fields of the packet's Requester ID.

The Requester ID of the packet as emitted onto the link into the system backplane is shown in the middle row of the figure.

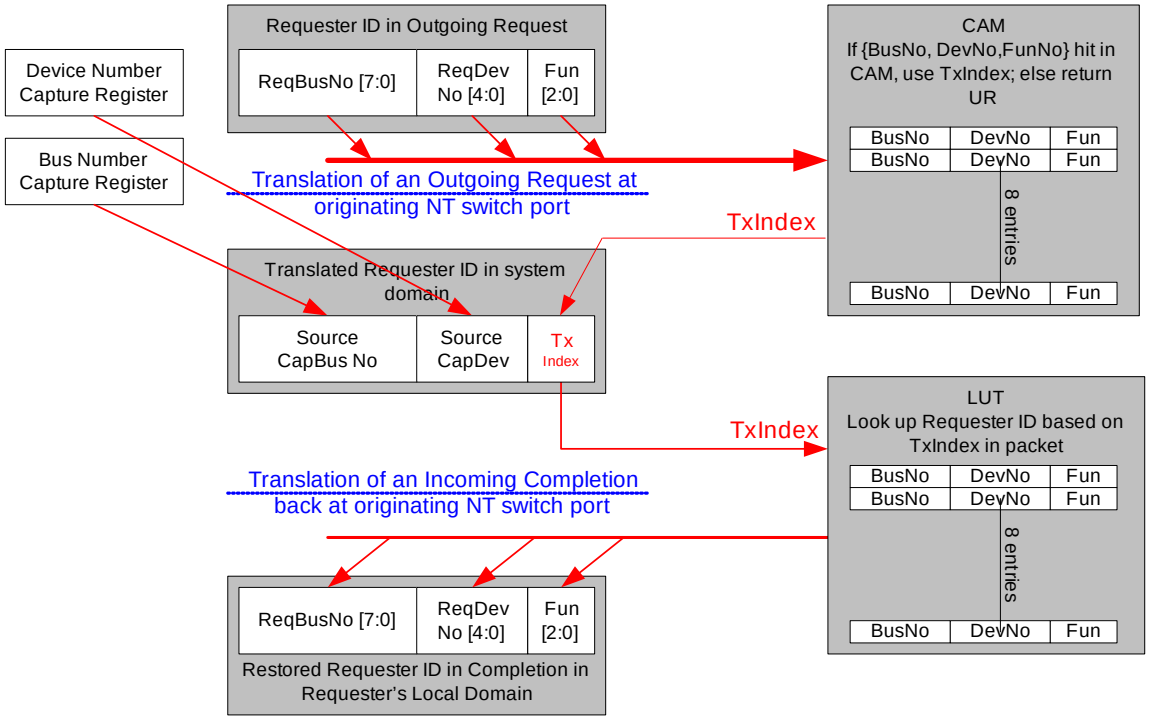

Figure 6 Requester ID Translation During Read of Host Memory

Translation of an Incoming Completion

When a completion returned in response to previous non-poster request enters the non-transparent port from the link/system side, the TxIndex is retrieved from the function number field of the completion's Requester ID and used to look up the original Requester ID. This is inserted into the packet and packet is forwarded to the original requester in the local domain. The Requester ID of the completion as forwarded into the local domain of the requester is shown on the bottom row of Figure 6.

Note that the LUT and the CAM can be a shared use of the same data structure. At 8 entries times 16-bits, only 128 bits of register and equality comparator are required.

Completer Only Behind Non-transparent Port

Figure 7 illustrates the Requester ID translation steps when only the completer is behind a non-transparent switch port, as would be the case when the host reads memory on an intelligent adapter.

This translation uses a second CAM, in this case with 32-entries. This data structure supports 32 devices elsewhere in the system sending requests through the associated non-transparent switch port. Because the function number is not used in the CAM association, a separate CAM entry is not required for each requesting or phantom function of a device. The requirement again exists to configure the CAM before sending requests through the non-transparent switch port. This requester registration process, which can't be done by a peer, is an effective security and protection mechanism.

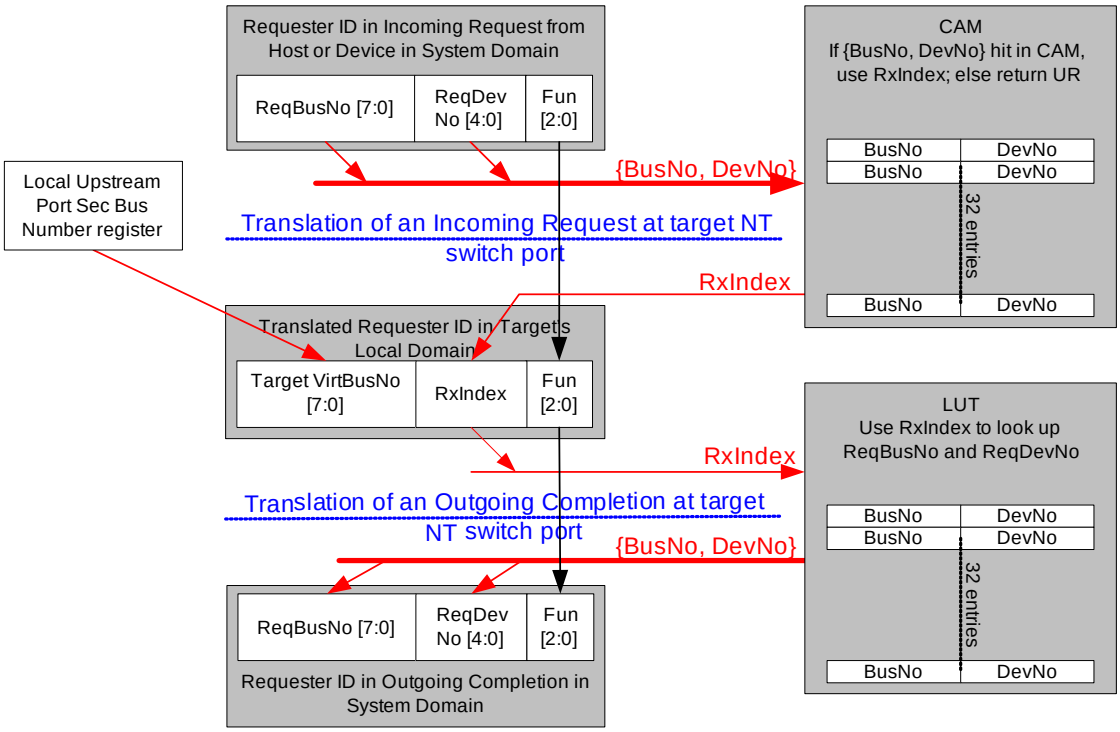

Figure 7 Requester ID Translation During Read of Intelligent Adapter Memory

Translation of an Incoming Request

When a request is received from its PCI Express link at a non-transparent switch port, a second translation of the request's ID occurs. The bus number and the device number, but not the function number, from the request's Requester ID is associated into the CAM and the RxIndex corresponding to the matching entry is substituted into the device number field of the packet's Requester ID. At the same time, the target switch's internal virtual PCI bus number is copied from its upstream port's Secondary Bus Number Register into the bus number field of the packet's Requester ID. The packet is then forwarded into the completer's local domain with its Requester ID as shown in the middle row of Figure 7.

The switch's internal virtual PCI bus number suffices to route the completion from the completer back to the non-transparent switch port in the completer's domain because, based on the limitation of a single non-transparent switch port per switch, the non-transparent switch port is the only possible requester on the switch's internal virtual bus. Anywhere else in the local hierarchy, the bus number suffices to route the completion back into the switch containing the non-transparent port.

Translation of an Outgoing Completion

The inverse translation occurs when a completion passes through the non-transparent switch port on its way into the system domain. The RxIndex is retrieved from the Requester ID field and used to look up the bus number and device number of the Requester ID as received in the request from the system domain. Note that if the request originated in the system/host domain, these would be from the ID of the original requester whereas if the request originated behind another non-transparent switch port, these would be the bus and device number of the virtual endpoint associated with the requester's non-transparent switch port. The restored Requester ID in the completion as sent into the system domain is shown on the bottom row of Figure 7.

Both Requester and Completer Behind Non-transparent Switch Ports

Figure 8 illustrates the Requester ID translation process when both requester and completer are behind non-transparent switch ports. It first shows the translation of an outgoing request at a non-transparent switch port, as originally illustrated in Figure 6, followed by the translation of an incoming request at a non-transparent switch port, as originally illustrated in Figure 7. These are then followed by the inverse translations of the completions, again as originally illustrated in the same two figures. Showing all of this on a single drawing manifests the non-interference of the two translations and that neither is dependent upon the other having been done.

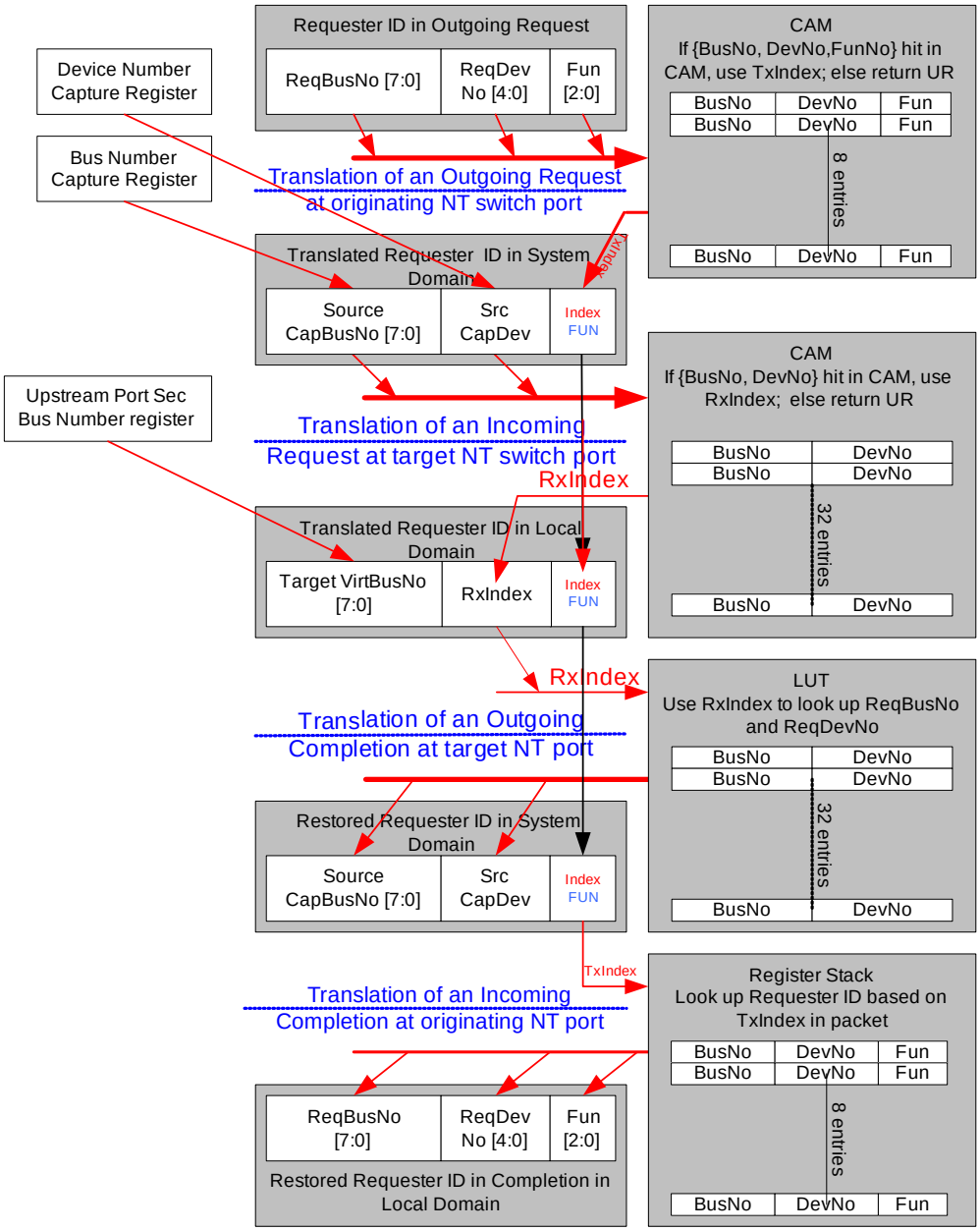

Figure 8 Requester ID Translation When Both Requester and Completer are Behind Non-transparent Switch Ports

Completer ID Translation

The Completer ID is part of a completion and requires translation when a completion passes through a non-transparent switch port. Figure 8 shows the translation of the Completer IDs in both outgoing and incoming completions.

This translation of the Completer ID in an outgoing completion identifies the virtual endpoint associated with the non-transparent switch port as the source of the completion.

This would only be important to software processing a packet error logged using advanced error reporting capability.

The translation of the Completer ID of an incoming completion identifies the virtual endpoint associated with the virtual side of the non-transparent switch port as the source of the completion. Again, this is of potential importance only to error processing software examining an advanced error reporting header log. Bus number and device number capture registers are not needed on the virtual PCI bus inside the switch and so their use is not shown in the right half of Figure 8.

Without Completer ID translation, bus and device numbers aliasing between local and system domains or between requester and completer local domains may occur.

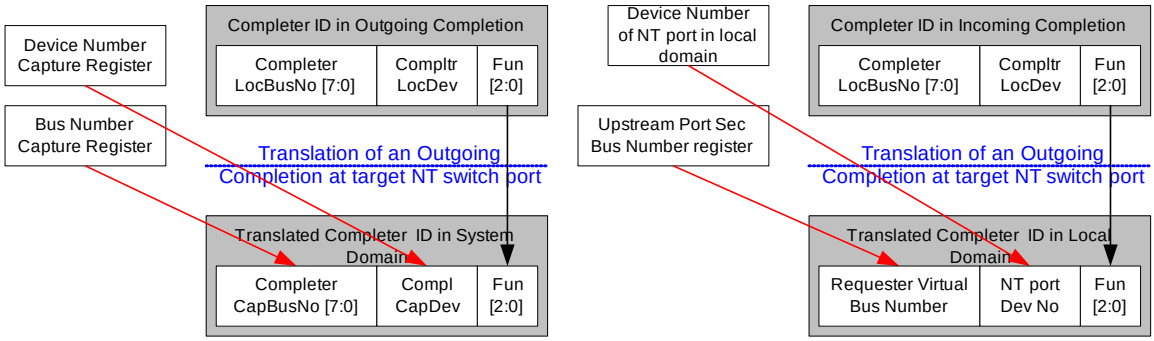

Figure 9 Completer ID Translation

Specification Compliance

Switch port or endpoint?

The non-transparent bridge/switch port has two personalities because of the virtual PCI Express endpoint associated with each side of the switch port. It behaves as an endpoint to configuration transactions and as a switch port to other kinds of transactions. In the PCI Express specification, different, sometimes conflicting, requirements are placed upon endpoints and switch ports. In specifying the behavior of the non-transparent switch port, one must choose which rule to apply in those cases.

For example, the PCI Express specification contains a rule that endpoints may not be located on the virtual PCI bus of a switch. We judge that this does not apply to the non-transparent switch port because it is not an endpoint; it merely uses the same CSR header format as an endpoint.

An endpoint is required to advertise infinite completion credits. A switch port is allowed to flow control completions. The non-transparent switch port flows completions again because it is not an endpoint; it merely uses the same CSR header format as an endpoint.

An endpoint is by default allowed only 32 outstanding transactions. A switch port is not required to limit the number of outstanding transactions it forwards. The non-transparent

switch port does not take ownership of non-posted requests in order to limit the number of outstanding transactions because it is not an endpoint; it merely uses the same CSR header format as an endpoint.

The Device Status Register of PCI Express devices includes a Transactions Pending bit. For an endpoint, this bit when set indicates that the device has issued Non-Posted Requests which have not been completed. The non-transparent switch port does not issue non-posted requests on its own behalf and so hardwires this bit to zero, as permitted by the PCI Express specification. The TP bit therefore does not indicate that all non-posted requests forwarded through the non-transparent port have been complete.

Use of captured device number

The PCI Express specification requires all devices to capture both the bus number and the device number on every configuration write request and use the captured values in the Requester and Completer IDs of initiated transactions containing a Requester ID. However, there is some ambiguity if one differentiates between *initiated* and *forwarded* transactions. Interpretation in the event of ambiguity should be guided, as we have been, by the requirement for correct functionality.

The non-transparent switch port as described herein makes a single exception to the above rule. For both requests and completions forwarded onto the link/system side of the non-transparent port, and for completions forwarded into the local domain, it is in complete conformance. However, the Requester ID of requests forwarded into the local domain contains the mandated bus number but a CAM index instead of the mandated device number. Earlier we described how the architecture is limited to a single non-transparent port in a switch and therefore on a virtual PCI bus segment. Consequently, the non-transparent port is the only possible requester on such a bus segment. Therefore, only the bus number is required for routing the completion back to the non-transparent port. Thus correct functionality is assured.

Intelligent Adapter Usage Model

When an intelligent adapter is implemented with PCI Express native devices, a switch with a non-transparent port for connection to the system host is required. Such a switch may be a separate component, as shown in Figure 10, or may be integrated into the local root complex.

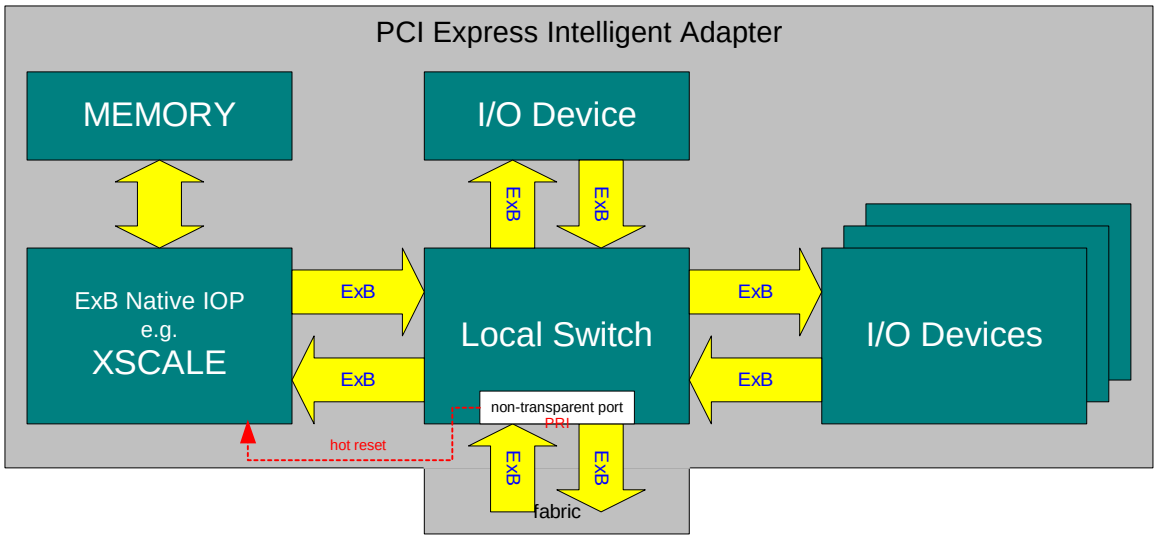

Figure 10 Intelligent Adapter with PCI Express Native Devices

The software model for the intelligent adapter usage model is shown in Figure 11 and the associated address translation model in Figure 12. The system host is connected ostensibly via a fabric to the PRI side of the non-transparent bridge port. There it sees a CSR BAR for memory-mapped access to the port's registers and a number of downstream ports for creating address translated apertures into the IOP's memory space. The local host and local devices see a number of upstream BARs on the SEC side of the non-transparent bridge port that creates address-translated apertures into the host's memory space. The local host and any of the local I/O devices may exchange data with the host via these apertures. Even if the local processor supports only 32 bits of address, local devices may employ 64-bit addressing in communications with the system host.

In this model, all configuration of the BAR setup, translation, and limit registers is done by the local processor. The system host, connected to the Primary side of the non-transparent bridge is responsible only for establishing the base addresses of its windows into the local processor's space. The CSRs of the non-transparent bridge port include a Primary Bus Access enable bit and a Secondary Bus Access enable bit which prevent access by either processor before appropriate configuration of the local subsystem has been accomplished. These are under control of the local processor. Software initialization is discussed in a later section.

A hot reset coming from the system host via the PLL of the PCI Express Link is coupled to the local root complex as a sideband signal. There, it can be used to cause a reset that is driven down into the entire local hierarchy or can be used simply as an interrupt.

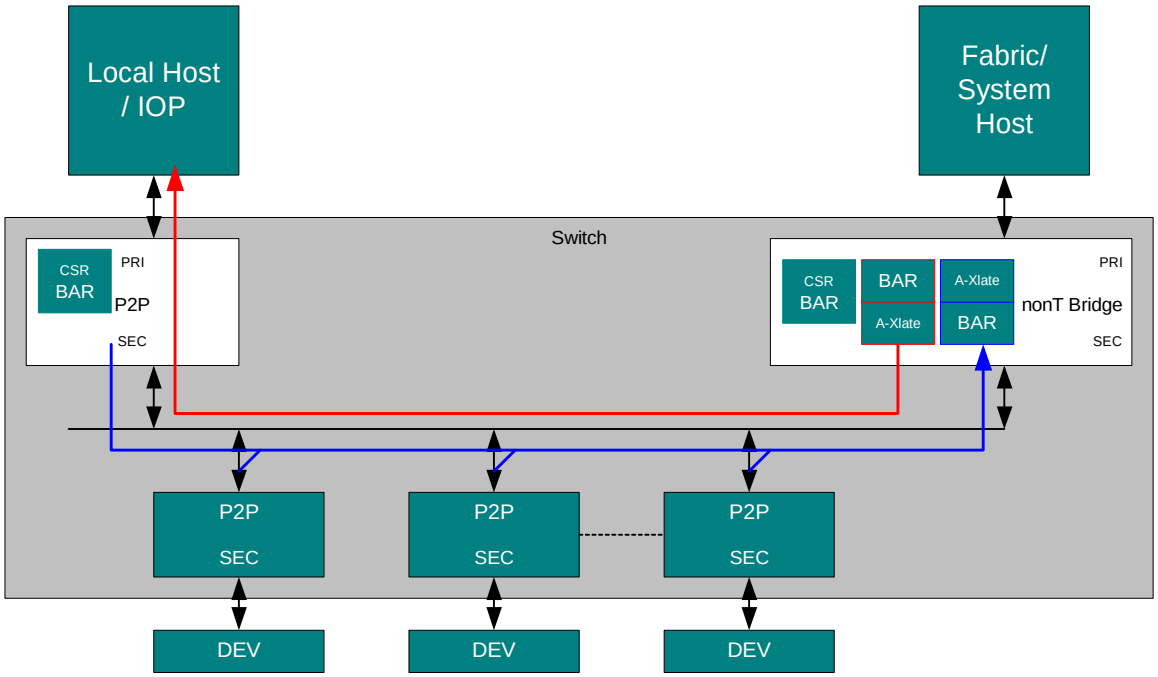

Figure 11 Software Model for the Intelligent Adapter Usage Model

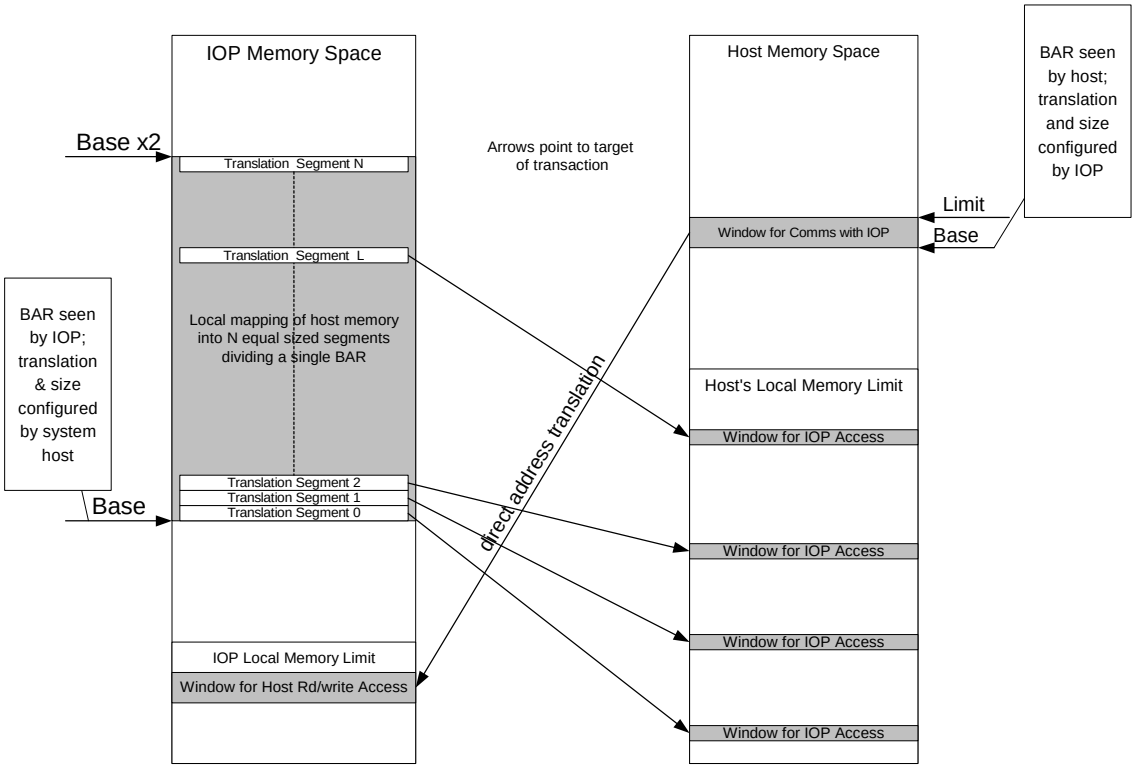

Figure 12 Address Translations for the Intelligent Adapter Usage Model

Dual-host/Fail Usage Model

In a dual-host application, provision is made for both a primary or active host and a secondary or backup host. During normal operation, heartbeat messages are sent from primary to secondary to indicate that it is still alive. Checkpoint or journal message containing the current state and transaction history are also sent periodically from primary to secondary. The job of the secondary host is to monitor the state of the primary and, upon detection of its failure, to take over as primary host continuing system operation from the last valid checkpoint.

In our usage model, the secondary or backup host is connected into the system via a non-transparent bridge while the primary or active host is connected via a transparent bridge. This is shown in Figure 13 below. The secondary host connection could be directly to its Root Complex or through a fabric connection. In the latter case, both hosts may be active simultaneously. In this case, heartbeat and checkpoint messages would flow in both directions.

The BARs on both sides of the non-transparent bridge are used to create tunnels through which each host may send messages to the other host. The doorbell registers available in the non-transparent bridge may be used for heartbeat messages. The memory access tunnels are used for checkpoint and other data transfer.

Failure of the primary host is declared when the secondary host fails to receive a certain number of the regularly scheduled heartbeat messages. As part of the fail over process, the secondary hosts' port is reconfigured to be transparent as shown in Figure 14 and as the upstream port of the PCI hierarchy. Failure of the primary host likely leaves switch buffers backlogged and device endpoints with incomplete transactions. During the fail over process, detailed later, the secondary host causes the buffers to be flushed and terminates incomplete transactions at endpoints. It then reconfigures the system with itself as host and restarts the devices and applications in some application specific way, using checkpoint or journal data.

Figure 15 shows a system with two active hosts, in which each host serves as the secondary host to the other.

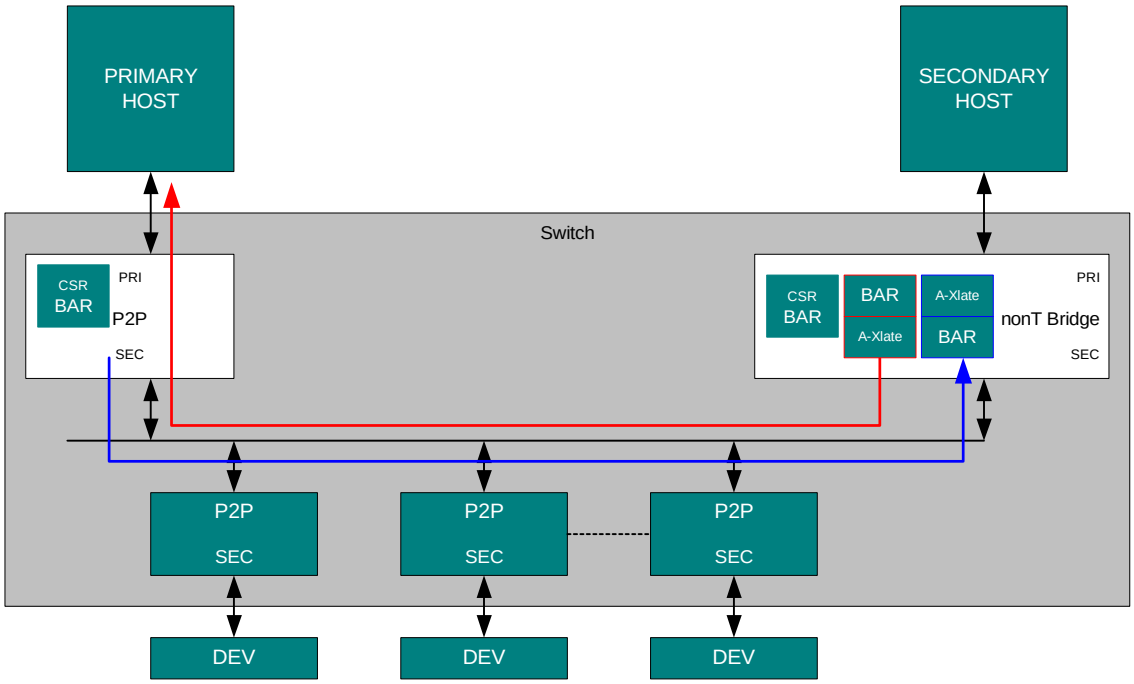

Figure 13 Dual-host System Pre Fail Over

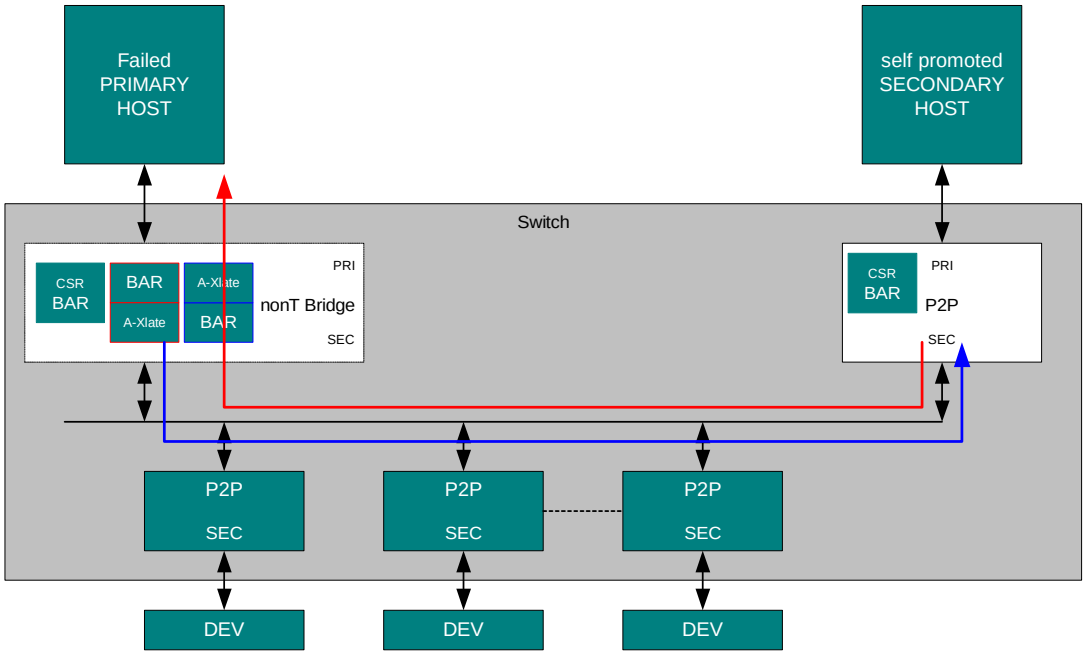

Figure 14 Dual-host System Post Fail Over

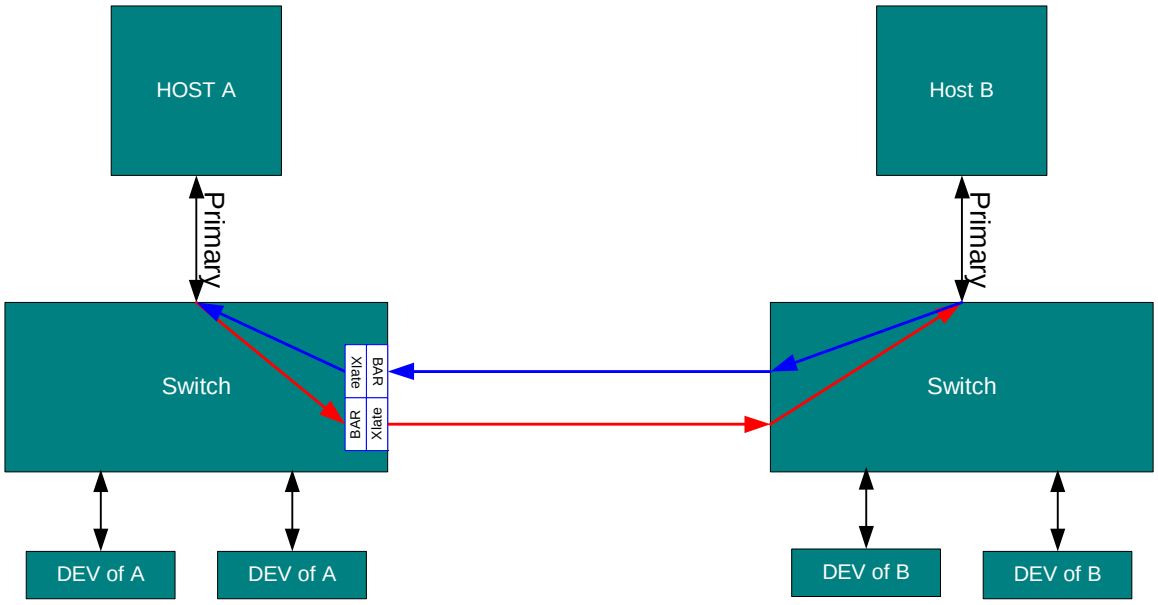

Figure 15 Dual-host System With Both Hosts Active

Dual-star Topology Usage Model Extension

The dual-host system model may be extended to a fully redundant dual star system by using additional switches to dual-port the hosts and line cards into a redundant fabric as shown in Figure 16. Two host cards are shown. The Host A is the primary host of Fabric A and the secondary host of Fabric B. Similarly, Host B is the primary host of Fabric B and the secondary host of Fabric A.

Each host is connected to the fabric it serves via a transparent bridge/switch port and to the fabric for which it provides only backup via a non-transparent bridge/switch port. These non-transparent ports are used for host-to-host communications. They also support cross-domain peer-to-peer transfers where address maps do not allow a more direct connection.

Line cards are shown on the left of the system. Each includes a switch for providing connections into both fabrics. One of the fabric links is configured to be the upstream port of this local switch, thus defining its affinity to one of the hosts. If the local subsystem is intelligent, then the local switch port closest to it is configured to be non-transparent per the intelligent adapter usage model.

Peer-to-peer transfers within a host domain are switched directly between endpoints by the fabric associated with the domain. Cross-domain peer-to-peer transfers may be routed first to the domain's host, through its non-transparent switch port, then back down the other hierarchy to the endpoint. This has the advantage of better supporting the PCI ordering model by pushing more writes ahead of any such reads but the disadvantage of potentially creating a bottleneck at the host. If the host links are fat compared to the line card links this may not be problematic. In many cases, system memory maps can be

created that allow direct cross domain transfer through the alternate fabric connection at the source line card, avoiding this potential bottleneck.

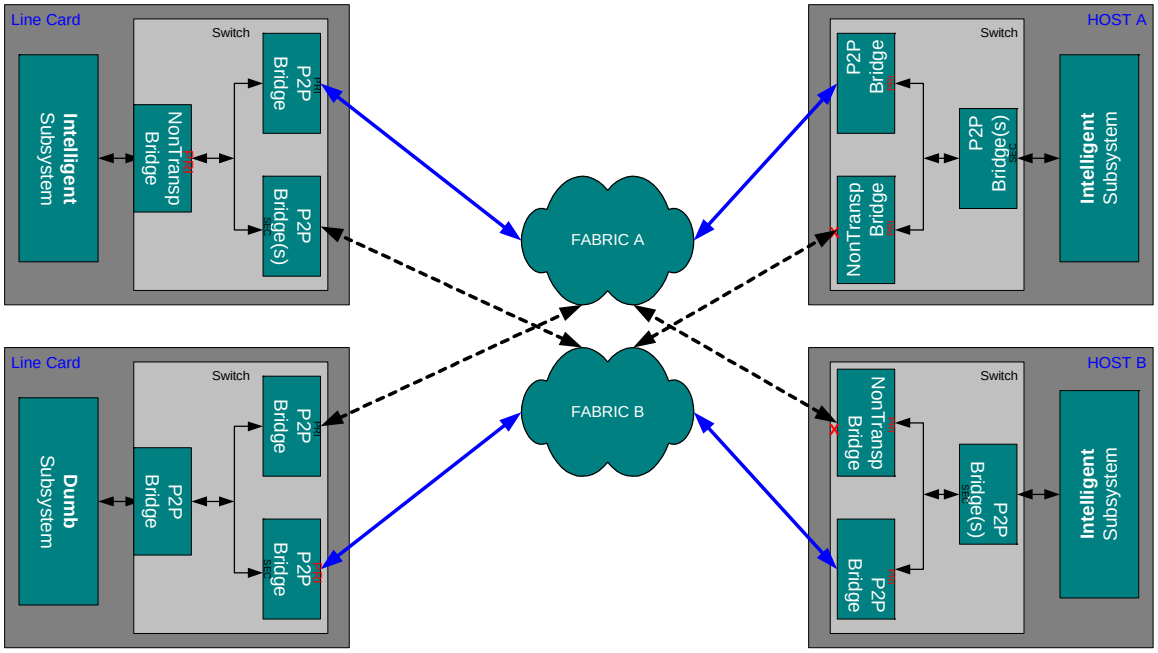

Figure 16 Dual-star Topology

Fail Over

Issues associated with Hot Reset Caused by Failure of the Primary Host

The implementation of the switch must be such that failure of the primary host can't prevent accesses by the secondary host necessary for fail over. One potential failure mode is with the PCI Express link to the primary host in the DL_DOWN state. A literal interpretation of the PCI Express specification would then reset the entire hierarchy and hold it in reset until the link comes back up. To allow fail over, we instead give a transient reset upon entry of DL_DOWN , allowing the secondary host to take over after this reset de-asserts. Since a reset of the entire hierarchy may be unnecessarily disruptive to systems that are capable of surviving host failure, we also provide a device dependent means of blocking the propagation of even a transient reset from the switch's upstream port to the rest of the hierarchy due to DL_DOWN .

The secondary host port is identified by device dependent means, typically hardware straps with a provision for over-ride via EEPROM. This causes the port to be configured as a non-transparent bridge and ensures that this configuration survives any level of reset, other than power off. The link partner of the secondary host port always has access to the port's Primary side Type0 CSR header because hot reset isn't propagated across the non-transparent bridge. It uses this access to configure the CSR BAR of the header it sees,

giving it access to all necessary switch CSRs through the CSR BAR, independent of the state of the primary host, so long as it the entire hierarchy is not held in reset.

After any reset in dual-host mode, the Primary BusAccess Enable bit is set, allowing access by the Secondary Host.

Secondary Host Requires Non-blocking Access to CSRs

The non-transparent bridge/switch port appears to its link partner as a PCI Express endpoint in CSR space. This model requires that it have guaranteed access to its CSRs. The reason for this requirement becomes clear when we consider the implications of failure of a primary host. Unless its access to the switch CSRs are assured, we can't guarantee successful fail over to the secondary host. This requirement is imposed at the architectural level; provisions for satisfying it are implementation dependent.

Providing non-blocking access to CSRs is not difficult, since CSRs by nature simply consume packets without being subject to external flow control. Some CSR request packets require completions and inability to send the completion could cause head of line blocking of additional requests. However, the PCI Express specification does not allow endpoints to flow control completions, guaranteeing forward progress for them. In cases where upstream completions (or other packets) stall due to link retry protocol, either forward progress occurs or the link is brought down, causing backlogged CSR completions to be discarded.

The Fail Over Process

When a host fails any number of things may go wrong. However, the symptoms observable in the fabric are limited to the following set. The host might:

- Stop sinking its packets
- Stop completing sunk requests
- Stop servicing interrupts
- Stop sending heartbeats to the secondary host
- Start sending rogue packets[1]

When a host fails, its link to the switch may go down. As discussed earlier, the entire local hierarchy, except for the secondary host, may receive a reset. The implementation ensures that this reset does not endure forever and prevents the de-configuration of the secondary host because of the reset. The implementation also allows configuration that blocks the propagation of such a reset. If the reset is allowed to occur then bring up of the system by the secondary host is simplified because the entire hierarchy has been reset and needs no further discussion. The rest of this section deals with the case where the link to the host stays up or the reset is not propagated.

The secondary host initiates fail over when it stops receiving heartbeat messages.

[1] The only rogue packets which create a vulnerability are those which change the non-transparent attribute of the secondary host's port (strap-selected).

Without a DL_DOWN reset, the consequences of the failure are that buffers pointing towards the failed host may be backlogged and incomplete requests may exist at endpoints. There should be no backlogged queues pointing towards the secondary host because, in the usage model, the secondary host receives only limited traffic and only from the primary host, and a premise of failure is the cessation of that traffic.

Some of the backlogged buffers may contain packets that are implicitly routed upstream - to the host port of the moment. Packets of this type include interrupts, error message and power management messages. Thus, when the secondary host promotes itself to primary, it may begin receiving these packets. This is can be handled by the fail over software. It implements "dummy" interrupt service routines to sink the backlog of interrupts and messages at fail over. After the backlog is exhausted and the host has restarted the applications it may then vector the interrupts and messages to "real" handlers.

Masking errors and interrupts at the port about to be promoted to upstream prevents new interrupts and messages from being generated but doesn't cause backlogged interrupts or message to be discarded. The "dummy" ISRs mentioned above can simply discard the interrupts and messages because either the information they contain is stale due to host failure or remains accessible to the new host in status bits downstream in the hierarchy.

PM_PME messages are also routed implicitly to the upstream port and are not maskable in the switch. The secondary host therefore must be prepared to handle latent PM_PME messages before promoting itself to primary host. These messages may be discarded safely by the secondary host during the fail over process because the PCI Power Management process ensures that they will be resent if not acknowledged in a timely fashion.

The secondary host undertakes the following sequence of operations before restarting the system to ensure that these backlogged buffers and pending transactions are flushed:

- Demotes the former host to *downstream* via its PCI Express Capabilities register
- Brings down the former host's link via the Link Disable bit of its Link Control register
- Masks interrupts to what will be the new upstream port
- Masks error messages to what will be the new upstream port
- Promotes itself to *upstream* port operating in *transparent* mode via its PCI Express Capabilities register
- Clears its own Bus Master Enable bit so that transactions may not be forwarded upstream to it
- Waits for the buffers to clear and incomplete transactions to time out.
- Reconfigures the system and restarts the application(s) using checkpoint/journal data

Packets that reach an egress port whose PCI Express link is down are discarded. The former upstream link is brought down to cause latent packets address routed to the failed host's port to be discarded.

Upstream packets that reach a port whose Bus Master enable bit is cleared are also discarded. (In some cases, an Unsupported Request packet may be returned for a discarded request.) Therefore, clearing the Bus Master Enable bit of the new upstream port is an additional protection against the new host receiving ghost packets, one latent in the switch at the onset of fail over, from reaching the new host. This provision is probably unnecessary because of the fact that all the bridges forwarding apertures are closed. Thus, the fact that the Bus Master Enable bit can't be cleared until after the secondary host is promoted, technically a potential race condition, is of no concern.

Thus, it can be seen that the preparatory steps of the fail over process cause the switch's buffers to be flushed towards either the new or old host, or perhaps some packets to each and to be discarded as they reach it. With the exception of errors, PME messages, and interrupts, packets latent in the switch at the onset of fail over are prevented from reaching the newly promoted host and are instead discarded. Once a quiescent state is reached, the new host can bring the system back up. A hot reset of the hierarchy, or portions of it, is not necessary but potentially can be used to accelerate the fail over process. If an inadvertent reset occurs because of the transitioning of the upstream link to the DL_DOWN state because of the failure of the host, fail over is again only accelerated, not inhibited.

Software Initialization and Configuration

The configuration of a non-transparent bridge involves a few more steps than a transparent bridge configuration. Configuration registers can be programmed from EEPROM alone, or through a combination of EEPROM and software running on the local (secondary side) of the Bridge. The following discusses the initialization sequence of a Non-Transparent bridge or switch port. It applies to the intelligent adapter usage model, in which the system host is connected to the outward facing Primary side of the bridge.[2]

The Non-transparent bridge has two separate configuration register sets, one accessible from the primary side and one accessible from the secondary side. Note that the contents of some registers are common. Some registers are RW from one side but RO from the other side. A Primary Bus Access Bit must be set from the local side or EEPROM to allow access from the primary side. Similarly, a Secondary Bus Access Bit must be set by default in the absence of EEPROM or from EEPROM to allow access from the secondary side.

There are four groups of registers that require programming in Non-Transparent mode.

- The Primary Interface's Standard CSRs (first 64 bytes of the configuration header) – excluding the address BARs
- PCI Express Interface configuration (this is mostly done from EEPROM)
- Address BAR configuration (BAR setup/translations registers)

[2] In dual-host mode, the hardware allows accesses from both sides of the bridge after EEPROM load; relying on software cooperation/communication to co-ordinate configuration activities.

- Communication CSR Configuration (Doorbells, scratchpad)

The standard CSRs of the primary interface are usually assigned values from EEPROM. These registers give the device its “personality” (Vendor/DeviceID and Device Class).

The PCI Express Interface Registers are also usually assigned values from EEPROM. These registers describe PCI Express attributes like max payload size and link width.

The communications CSRs are used to setup/define the behavior of the signaling & messaging interfaces used by the host & local processors to communicate with each other. These can be read or written at any time without side effects.

In any case, prior to primary side access, e.g. by the system host, a number of configuration registers need to be initialized. The forwarding BAR types and sizes must be programmed, the vendor/device/class all must be set, and any communications mechanisms used must be configured. Once the Primary Bus Access Bit is set, none of the registers that affect the primary side interface should be changed.

Serial EEPROM Load

At power on reset, all internal registers except sticky bits are set to their power on reset value. This includes setting the Primary Bus Access and Secondary Bus Access bits to 0, which will disallow PCI configuration access from both sides of the non-transparent switch port or bridge. During the time that the Primary and secondary Bus Access bit are set to 0, all accesses will be retried.

The CSR contents are then loaded from EEPROM at which point the secondary Bus Access bit is set, allowing access to the CSRs from the local side.

In Non-Transparent mode, the EEPROM load configures base aspects of the bridge along with the majority of the standard CSR space of its Primary Interface, which contains the VendorID, DeviceID, and Class of the device. The attributes of the BARs and the setting of the Primary Bus Access bit are generally left for software on the local side to configure, but it is possible to load all these values from EEPROM so that primary bus access can be enabled without intervention from software on the local side.

Initialization by the Local Processor

At completion of the Serial EEPROM load, the Secondary Bus Access bit is set, allowing CSR access from the Local Processor. While the Primary Bus Access bit is clear, the bridge continues to return target retry for any configuration accesses from the Primary interface.

At this point, the local processor can configure resources needed by the Primary interface like the attributes of the primary side BARs (size, type, forwarding translation). When the local processor initialization is complete, it sets the Primary Bus Access bit, which allows host access to the primary side registers.

The Primary Bus Access bit can also be set during the Serial EEPROM load. Using this method requires that EEPROM contain all the BAR type, size and forwarding information. It is important to note that once the Primary Bus Access bit is set, the local processor should not change any primary interface registers that can affect host configuration (BAR sizes for example).

Initialization by the Host Processor

Once the EEPROM and the local side have both completed their configuration(s), the Primary Bus Access bit must be set to a 1. This will allow the PCI Express host to commence with standard PCI discovery and configuration.

To the host side, the Non-transparent Forward Bridge appears to be a PCI Device with a type 0 header. It is configured normally by writing all ones to the address BARs, reading back the aperture sizes/types and finally assigning base addresses to each valid BAR.

The local host is responsible for creating and setting up the resource sizes for the Primary side BARs along with their forwarding translations. If all the resources on the local side are static, it is possible that the values could be loaded from EEPROM, but the values are typically programmed by the local side processor which also goes through its own initialization and self-test sequence.

Detailed initialization Sequences

Here are three variations on the initialization sequence for the intelligent adapter usage model of the non-transparent bridge/switch port.

Initialization Sequence 1 – Configuration duties shared between EEPROM and local side software

1. Standard CSRs for the primary interface along with PCI Express Interface CSRs are loaded from EEPROM. At the completion of EEPROM load, the local access bit is set.
2. On the secondary side, the local processor programs the BAR setup registers, completes internal initialization and sets the primary Bus Access bit.
3. The host can now configure the BARs of the bridge and assign it resources during PCI enumeration.
4. The host loads the driver for Bridge, which goes through an initialization dialog with the local processor that sits on the Bridge's secondary bus at which point they configure the communication resources.

Initialization Sequence 2 – Configuration by local side software (no EEPROM)

1. The local access bit is set after no EEPROM is detected.
2. On the secondary side, the local processor programs the standard CSRs for the primary interface along with PCI Express Interface CSRs and BAR setup registers, completes internal initialization and sets the primary Bus Access bit.
3. The host can now configure the BARs of the Bridge and assign it resources during PCI enumeration.

4. The host loads the driver for Bridge which goes through an initialization dialog with the local processor that sits on the Bridge's secondary bus at which point they configure the communication resources.

Initialization Sequence 3 – EEPROM Only configuration

1. Standard CSRs for the primary interface along with PCI Express Interface CSRs and BAR setup registers are loaded from EEPROM. The Primary Bus Access bit is also loaded from EEPROM. At the completion of EEPROM load, the local access bit is set.
2. The host can now configure the BARs of the Bridge and assign it resources during PCI enumeration. The local side can also complete its initialization at this time.
3. The host loads the driver for Bridge, which goes through an initialization dialog with the local processor that sits on the Bridge's secondary bus at which point they configure the communication resources.

Appendix B: Mode Configuration Straps

- Host/upstream identified by straps or EEPROM at reset
  - Any port can be identified as upstream
  - Reflected in that port's PCI Express Capabilities Register
  - In dual-star, the line cards are upstream strapped to either primary or secondary fabric on their local switch

- Straps to configure any port as non-transparent
  - Reflected in that port's PCI Express Capabilities Register

- Strap to indicate dual-host vs intelligent adapter
  - Changes default/post EEPROM load state of Primary and Secondary Bus Access Enables