Enabling Multi-peer Support with a Standard-Based PCI Express® Multi-ported Switch

White Paper

Notes

By Kwok Kong

Introduction

There are basically three different types of devices in a native PCI Express (PCIe®) system; Root Complexes, PCIe switches, and Endpoints. There is only a single Root Complex in a PCIe tree. A Root Complex should be thought of as a single processor sub-system with a single PCIe port even though it consists of one or more CPUs, plus their associated RAM and memory controller, as well as other inter-connect and/or bridging functions.

PCI Express routes are based on memory address or ID, depending on the transaction type. Thus, every register and device (or function within a device) must be uniquely identified within the scope of the PCI Express tree. This requires a process called enumeration.

During system initialization, the Root Complex performs the enumeration process to determine the various buses that exist and the devices that reside on each bus, as well as the required address space for the device's registers and memory. The Root Complex allocates bus numbers to all the PCIe Buses and configures the bus numbers to be used by the PCIe switches. A PCIe switch behaves as if it were multiple PCI-PCI Bridges (see inset in Figure 1). The Root Complex allocates and configures the Memory and I/O address space for each PCIe Switch and Endpoint device. A PCIe tree topology is shown in Figure 1.

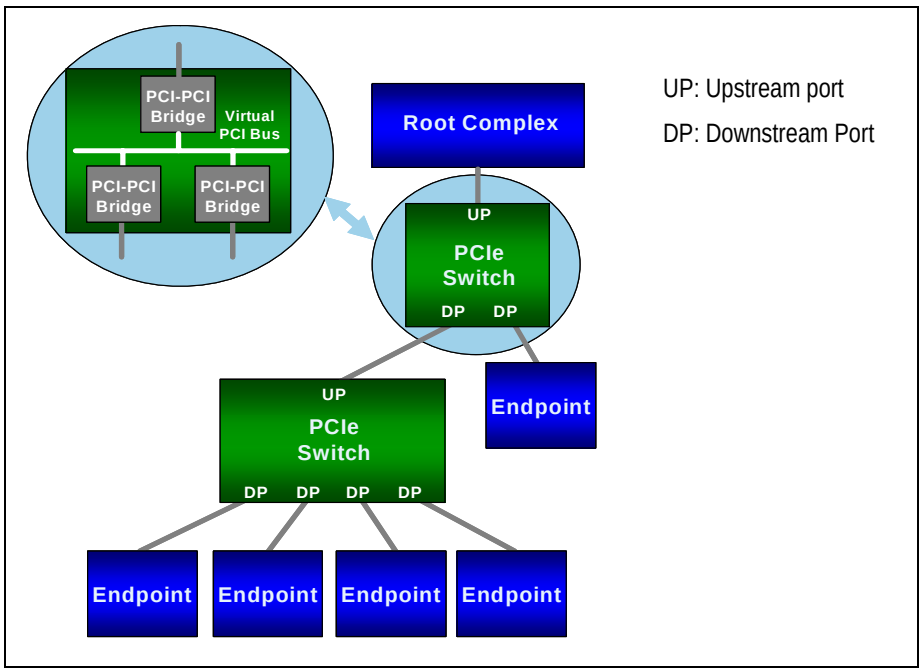

Figure 1 PCIe Tree Topology

A multi-peer system is one in which more than one processor sub-system exists in the scope of a PCI Express tree. For example, a second Root Complex may be added to the system via the Downstream Port (DP) of a PCIe switch, possibly to act as a warm stand-by to the primary Root Complex. However, an issue arises when the second Root Complex also attempts the enumeration process. Assuming that the link trains (i.e. using crosslink training), it sends out Configuration Read Messages to discover other PCIe devices on the system. Configuration transactions can only move from Upstream to Downstream. A PCIe

Notes

switch does not forward or respond to Configuration Messages that are received on its Downstream Port (DP); it ignores and silently drops all the configuration messages. Thus, the second Root Complex is isolated from the rest of the PCIe tree and will not detect any PCIe devices in the system. So, simply adding processors to a Downstream Port of a PCIe switch will not provide a multi-peer processing solution.

Supporting multiple processors in a PCI system by using proprietary solutions, such as Non-transparent Bridging (NTB), has been around for some time. This paper proposes a solution to support multiple processors in a PCIe system using a standards-based PCIe switch.

Multi-peer Systems

A multi-peer system topology is shown in Figure 2. There is only a single Root Complex (RC) processor in the topology. The RC processor is attached to the single Upstream Port (UP) of the PCIe switch. The RC processor is responsible for the system initialization and enumeration process as in any other PCIe system. A 16-port PCIe switch is used as an example in this paper, allowing up to 15 End Point (EP) processors to be connected in this system.

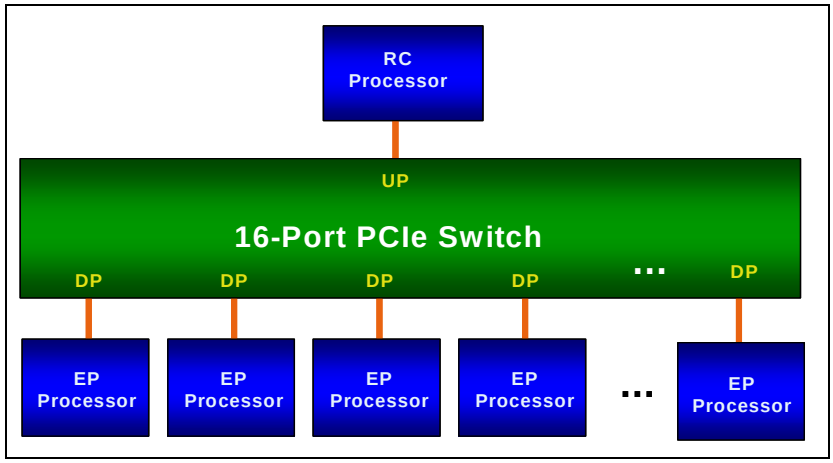

Figure 2 Multi-peer PCIe System Topology

An EP processor is a processor with one of its PCIe interfaces configured as a PCIe endpoint. Two examples of an EP processor are the AMCC PowerPC® 440Spe and the FreeScale MPC8641. An EP processor may also be an intelligent I/O adapter which is able to process I/O transactions, such as RAID controllers for storage. An example of an intelligent I/O is the Intel® IOP333. Figure 3 defines PCIe Address Domains.

Notes

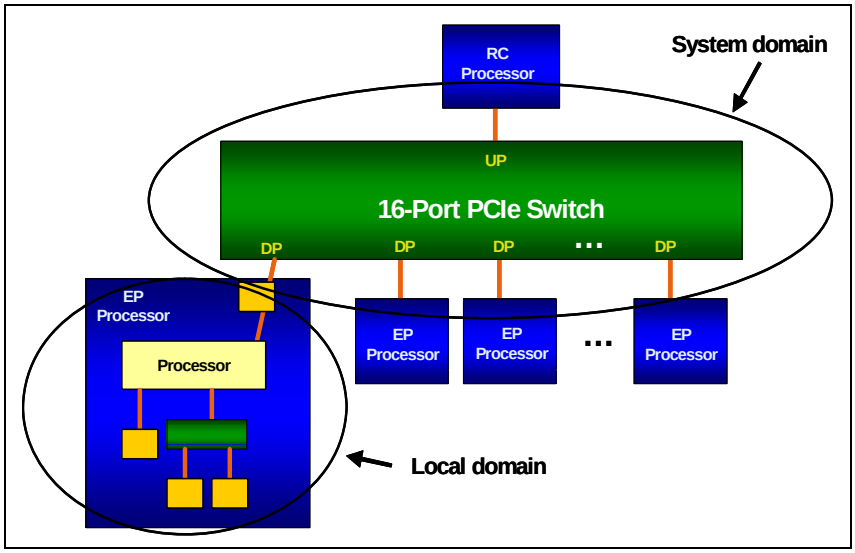

Figure 3 PCIe Address Domains

An EP processor may have local PCI or PCIe devices on its other port(s). The EP processor is the Root Complex processor for the local PCI and PCIe devices. In a multi-peer system, there are multiple PCIe domains. The system domain is owned by its RC processor and includes the RC processor, the 16-port PCIe Switch, and the PCIe endpoints on each of the EP processors in the system domain. The RC processor is responsible for the allocation of the PCI bus number and address space in the system domain. The RC processor stops the device discovery process on an EP processor when it detects the endpoint on the EP processor. The RC processor treats the EP processor as a PCIe endpoint device, in particular during its enumeration process. The class code as reported in the Configuration Register is typically 05H (Memory controller).

The EP processor is the Root Complex processor for its local PCI and PCIe devices; it is responsible for the allocation of PCI bus numbers and address space in the local domain. The PCIe endpoint device that connects to the system domain is typically an integrated local device in the EP processor. The EP processor does not detect any device in the system domain during its enumeration process. The PCIe endpoint device on the EP processor isolates the local domain from the system domain and it presents one endpoint configuration space to the system domain and another endpoint configuration space to the EP processor in the local domain.

Implementation Issues in Multi-Peer Systems

There are several issues involved in supporting multi-peer communication:

- Memory map management
- Enumeration and Initialization
- Peer-to-peer communication mechanisms
- Interrupt and error reporting
- Redundancy

Memory Map Management

Address routing is used to transfer data to or from memory, memory mapped I/O, or I/O locations. The RC processor is responsible for the allocation of memory addresses for the system domain. The EP processor is responsible for the allocation of memory addresses for its local domain. To avoid any memory address conflicts, a general implementation uses address translation between the system and local domain. In the system domain, the RC processor reserves a block of memory address space for all the EP processors. This reserved block of memory is divided equally, for example, into 16 smaller blocks of

Notes

address ranges. Each smaller block of addresses is reserved for one EP processor that connects to one of the ports of the 16-port PCIe switch. The actual physical memory of each smaller block exists in an EP processor. An example of the address map is shown in Figure 4. This example assumes that the EP processors are implemented on processing blades, one slot per PCI Express port, and that 1 MByte of address space is reserved per slot. A total of 16 MBytes of address space is reserved and the starting address is assumed to be 0x80000000. Whenever the RC processor detects an EP processor in a particular slot (or port of the PCIe switch) during system initialization, the RC processor assigns the address space to the EP processor based on what slot number the EP processor is in. The address space allocation and assignment is fixed based on the slot number and doesn't change when a EP processor blade is added or removed from a slot. The starting address, size of the address block that is reserved per slot, and the number of PCIe ports in the system should be adjusted based on system architecture. I/O transactions are not used for multi-peer communication and hence I/O address translation is not required.

The EP processor is responsible for the address map of its local domain; it allocates and assigns memory address ranges to its local devices. This is independent of the memory map of the system domain. (Only the address ranges should correlate, although the system range may be larger than the range required by the EP processor.) An address translation unit is required on the EP processor to translate address space between the system domain and the local domain. The EP processor supports both inbound and outbound address translation. Transactions initiated in the system domain and targeted at the EP processor's local domain are referred to as inbound transactions. Transactions initiated on the EP processor's local domain and targeted at the system domain are referred to as outbound transactions.

An example of the address translation is shown in Figure 5. This example assumes the EP processor is in slot 2. Two outbound address translation windows are created. One outbound address window is created to access the system domain address for slot 1. The second outbound address window is created to access the system domain address for slots 3 to 16. Note that since this EP is in Slot 2, there is no outbound window to address itself. Whenever the EP processor accesses any local address space that falls within one of the two outbound address windows, the address translation unit forwards the request to the system domain. The address is also translated from the local domain address space to the system domain address space. A single outbound address window may be used instead of creating two outbound address windows. An outbound address window is created for the entire system domain address that is reserved from slot 1 to slot 16. The software than runs on the EP processor makes sure that there is no local memory access to the address range that will be translated to its own slot (slot 2 in this example) in the system domain address map.

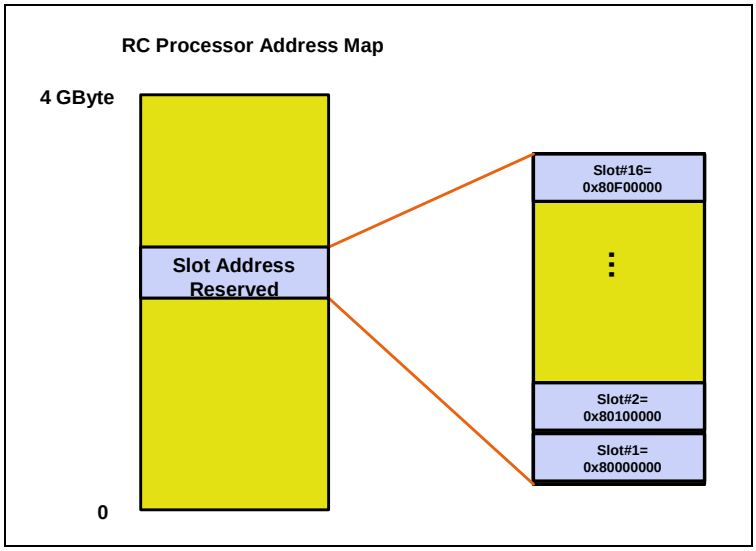

Figure 4 System Domain Address Map

Notes

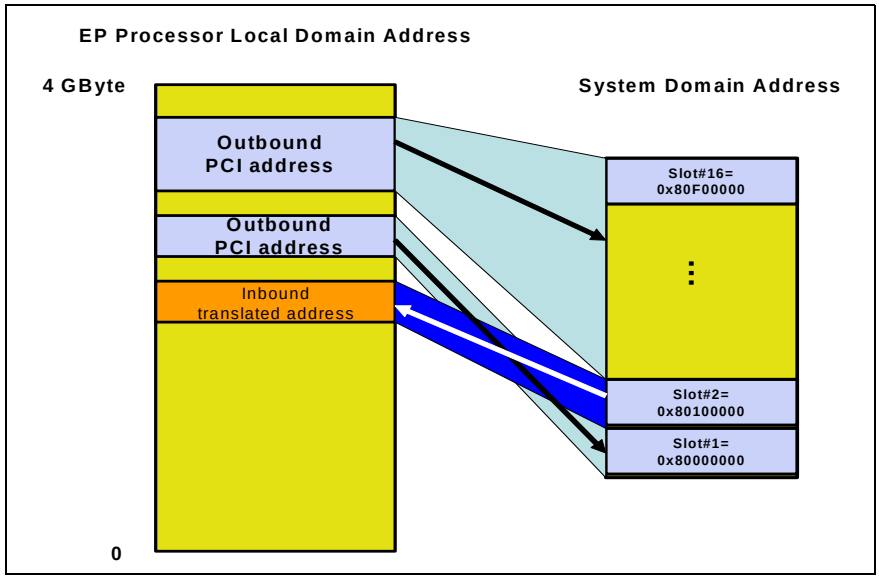

Figure 5 Address Translation Unit

A single inbound address translation window is set up to allow initiators in the system domain to directly access the local domain within the address range. The EP processor sets up the inbound and outbound address translation units (ATU's) to translate addresses between its local domain and the system domain, as shown in Figure 5. When the EP processor receives a memory request from the system domain, it receives the packet only if the address in the packet header is within the memory range assigned to the EP in the system domain. (The ATU may further check if the requested address is within a tighter address window as configured in the inbound address translation unit.) In this example, the address window is between 0x80100000 and 0x801FFFFF. If the requested address is within the address window, the request packet is forwarded from the system domain to the local domain. The address field in the requested packet is also translated to the inbound local address. The inbound local address may represent a local buffer in memory that the EP processor will read and respond to, or it may represent a local register that affects the EP processor directly.

When the EP processor accesses the ATU via its PCIe interface, then the Requester ID and Completer ID in the PCIe packet also have to be translated between the local domain and the system domain. If the processor accesses the address translation unit via a different bus, such as the local CPU bus, then there is no need to translate the Requester and Completer IDs.

Enumeration and Initialization

During system initialization, the RC processor scans the system domain for PCIe devices. The RC processor assigns unique bus numbers to all the PCIe links and the virtual buses within PCIe switches. The process of discovering the PCIe tree topology and the devices and functions which reside on this topology is referred to as the enumeration process. The normal PCIe (or PCI) enumeration process reserves bus numbers and address space for empty slots. There is no change in the normal enumeration process for multi-peer support. All PCIe switches and endpoints within the system domain are detected at the end of the enumeration process. Remember that all EP processors are detected as PCIe endpoints in the system domain and that devices locally attached to the EPs are not detected in the system domain at all.

After the enumeration process has completed, the RC processor assigns memory address space to each endpoint device in the system domain. The absolute memory address range (in the system domain) allocated to an EP processor is based on which slot the EP processor is plugged into, as shown in Figure 4. This example assumes that an EP processor requests no more than 1 Mbyte of address space in its Base Address Register (BAR). If an EP processor requires more than 1 MByte of address space, then the address map has to be adjusted accordingly. The corresponding address space is assigned to an EP by writing to its Base Address Register (BAR).

Notes

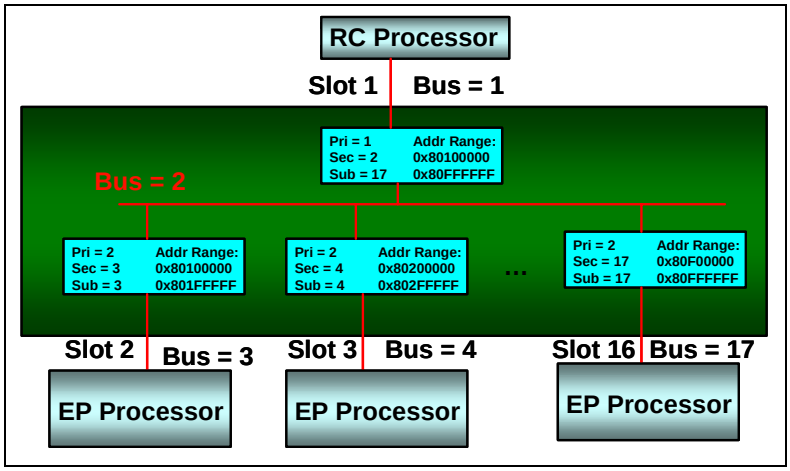

Figure 6 PCIe Switch Configuration after Enumeration and Initialization

There are multiple virtual PCI-PCI bridges in a PCIe switch. The configuration of the PCI-PCI bridges after enumeration and initialization is shown in Figure 6. The Primary Bus number (Pri), Secondary Bus number (Sec), Subordinate Bus number (Sub), and Address Range for a particular slot (or port) are shown in this example. After the enumeration and initialization process is completed, memory requests can be forwarded to the targeted EP processor using address routing in the PCIe switch.

Each EP processor also runs its own enumeration and initialization process on its local domain. Note that due to the isolation provided by the endpoint function within the EP processor, the local and system domain enumeration and initialization processes are independent and may proceed in parallel. The EP treats the PCIe interface to the system domain as a PCIe endpoint (i.e. just another local device), depending on the architecture of the EP processor.

The EP processor relies on the RC processor to detect the existence of other EP processors in the system domain. The RC processor has a complete topology map of the system domain. It notifies an EP processor of the existence of other EP processors.

The RC processor is also responsible for hot plug events in the system domain. A bus number and a memory address range are reserved for each slot in the PCIe switch. When an EP processor is plugged into the system, the pre-assigned bus number is used to access this EP processor. The BAR of the newly inserted EP processor is configured to the pre-assigned memory address offset. As shown in Figure 6, Bus number 4 is used, and memory address 0x80200000 is programmed into the BAR of an EP processor when an EP processor is plugged into slot 3. When the initialization of the EP processor is complete, the RC processor notifies all other EP processors that a new EP processor has been plugged into slot 3.

Conversely, when the EP processor is unplugged from slot 3, the RC processor notifies all other EP processors that the EP processor in slot 3 has been removed so that the resources associated with the removed EP processor can be released and cleaned up. However, the bus number and memory address are not re-assigned since they are reserved for slot 3.

Peer-to-Peer Communication Mechanisms

An interprocessor communication protocol has to be defined to enable peer-to-peer data transfer. The protocol has to define the handshake between the peers and the message format. This paper does not attempt to define the message format. Rather, it discusses methods of passing a block of data from one EP processor to another.

There are three basic categories of communication mechanisms:

- low volume of data traffic between an EP processor and the RC processor
- low volume of data traffic between the EP processors
- high volume of data traffic between the EP processors (including the RC processor)

Notes

For low volume data traffic between an EP processor and an RC processor, scratchpad registers, such as those implemented by a typical EP processor, may be used. Both the EP processor and the RC processor may read and write to/from the scratchpad registers. The EP processor usually implements multiple (for example, 8-16) scratchpad registers. Doorbell registers are used to send interrupts to both the EP processor and the RC processor. When the RC processor needs to send some data to an EP processor, the RC processor writes the data to the scratchpad registers. It then writes to the Doorbell register to interrupt the EP processor to indicate that a message is ready to be consumed by the EP processor in the scratchpad. The same procedure applies when an EP processor sends data to the RC processor.

For low volume data traffic between an EP processor and another EP processor, both scratchpad and doorbell registers can be used. A semaphore register with a lock/unlock mechanism is required to control the usage of the scratchpad registers since they are shared by all the EP processors. Before an EP processor can use the scratchpad, it has to acquire ownership of the scratchpad using the semaphore register. Once the semaphore is acquired, it has the ownership of the scratchpad and can start writing to it. Once the data is consumed, the sender releases the semaphore. An alternative method is for the RC processor to relay the data between the EP processors.

Advanced queueing is needed for high volume data traffic between the EP processors. A possible queueing structure is to follow the Intelligent I/O Architecture[1] which offers two paths for data messages: an inbound queue structure and an outbound queue structure. The inbound queue structure is used to receive data messages from any EP or RC processor in the system domain. The outbound queue structure is used to send data messages to the RC processor only. It is assumed that all EP processors implement the inbound and outbound queue structures and the RC processor does not implement any inbound/outbound queue structure. When an EP processor sends data messages to another processor, the local EP processor uses the inbound queue structure of the remote EP processor. When an EP processor sends data messages to the RC processor, the local EP processor uses its local outbound queue structure. The queue structure contains a pair of First In First Out (FIFO) Queues; FreeQ and PostQ. A read from the FIFO queue removes the first entry in the queue and a write to the FIFO queue adds an entry to the end of the queue. The FreeQ contains free buffer locations into which data messages can be written. The PostQ contains the locations of data messages that have been written to it (i.e., and posted by the sender).

Hardware-based Queue Structure

If the FIFO queue is implemented in hardware, such as the Intel IOP332/3 I/O processor, then a single memory read operation removes the first entry from the queue and a single memory write operation adds an entry to the queue. An EP processor needs to have just a single pair of inbound FreeQ and PostQ to receive data from multiple EP processors.

The inbound translated address is configured as shown in Figure 7. The translated address is basically divided into three different regions: the Queue Structure, some other hardware dependent control structures, and data buffers. The queue structure contains only a single address location to be accessed per queue. A memory read removes the first entry from a queue and a memory write moves an entry to the end of the queue. As an example, when the EP 1 processor needs to send a data message to the EP 2 processor, the EP 1 processor performs a memory read operation from the Inbound FreeQ on EP 2 to get a free buffer. After the EP1 processor has transferred all the data to the buffer, it performs a memory write operation to the inbound PostQ on EP 2 to notify it that a data message is ready for processing.

Notes

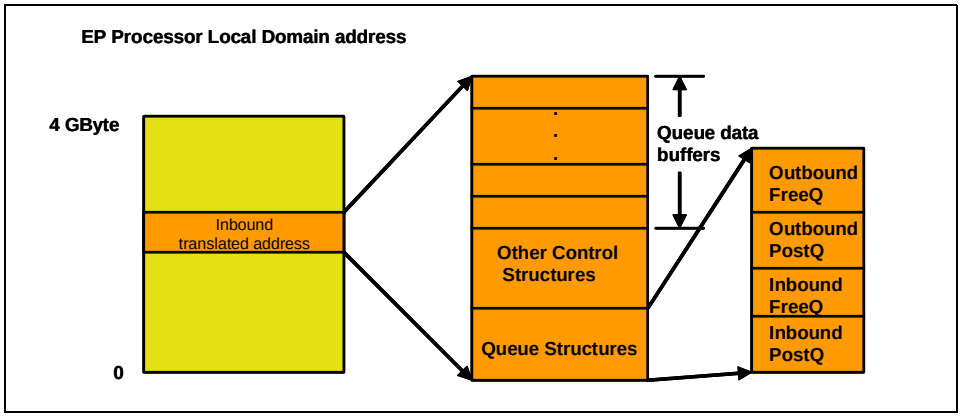

Figure 7 Hardware-based Inbound Translated Address Usage

Software-based Queue Structure

If hardware does not support the FIFO queue structure in the EP processor, the FIFO queue structure has to be implemented in software. Four addresses of the queue have to be maintained in the queue structure: Qstart, Qend, Qread, and Qwrite. Qstart points to the beginning of the queue and Qend points to the end of the queue. Qstart and Qend together specify the maximum number of entries in the queue. Qstart and Qend are static and do not change after initialization. An entry is removed from the location pointed to by Qread and an entry is added to the location pointed to by Qwrite. Qread is updated to point to the next location in the FIFO queue after removing an entry, and Qwrite is updated to point to the next location after adding an entry. The FIFO queue is empty when Qread is the same as Qwrite. The FIFO queue is full when Qread is one entry ahead of Qwrite. The operation to remove an entry from a FIFO queue requires multiple memory read/write operations:

- Read Qread
- Read Qwrite
- If Qread != Qwrite, move on to next step. Otherwise, FIFO is empty and stops here
- Read the location pointed to by Qread
- Adjust Qread to point to the next location.
- Write new value of Qread to FIFO queue structure.

The operation to add an entry to a FIFO queue requires similar operations:

- Read Qread
- Read Qwrite
- If Qread is one entry ahead of Qwrite, FIFO is full and stops here. Otherwise, move to the next step
- Write new entry to the location pointed to by Qwrite
- Adjust Qwrite to point to the next location.
- Write new value of Qwrite to FIFO queue structure.

Multiple Read and Write operations are required to add an entry to or remove an entry from a FIFO queue. Once an EP processor performs the first Read operation on a FIFO queue, no other EP processor should access the FIFO queue until all the Read and Write operations to add an entry or remove an entry from a FIFO queue are completed. Otherwise, the FIFO queue is corrupted. In order to guarantee that the queue add/remove operation does not get interrupted in the middle of an adding or removing operation, either a semaphore or multiple queues are needed. Assuming there is no semaphore support in the hardware, multiple FIFO queues are proposed here. A single pair of inbound FreeQ and PostQ has to be reserved per sender (EP or RC processor). In other words, a pair of inbound FreeQ and PostQ is dedicated to a single sender. For a 16-port (or slot) system, 16 pairs of inbound FreeQ and PostQ are required. 15 pairs of inbound FreeQ and PostQ should be sufficient because an EP processor does not need to use the inbound Queue structure to send data to itself. In this example,16 pairs of inbound Queue Structures are used.

Notes

The inbound translated address is configured as shown in Figure 8. The translated address is basically divided into three different regions: the Queue Structure, some other hardware dependent control structures, and data buffers.

The queue structure contains queue pointers plus the queue itself to store entries. At initialization time, queue pointers are configured to be empty. There are 2 queue structures, one for inbound and the other one for outbound, per EP processor (or per slot) in the system. There are 16 inbound and 16 outbound Queue Structures.

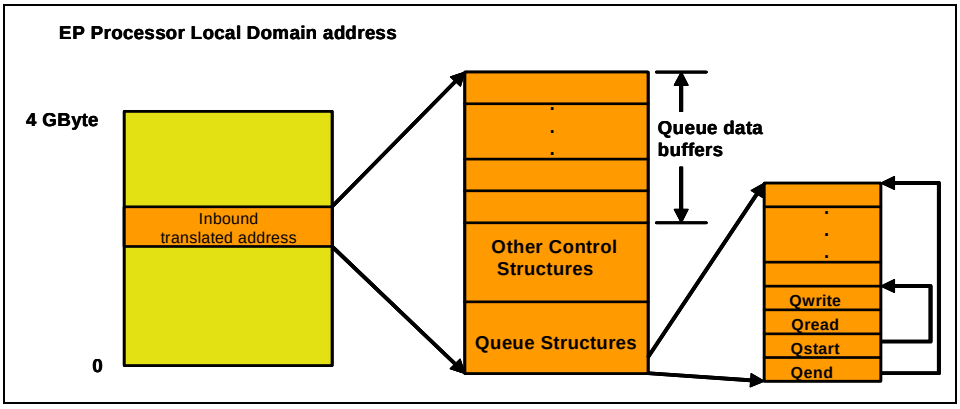

Figure 8 Software-based Inbound Translated Address Usage

Data Buffer Structure and Date Transfer Protocol

A local inbound queue is used to receive data from a remote EP processor. The inbound queue and data buffer usage are shown in Figure 9. In this example, the queue data space is divided into 128 equal size buffers. The buffer size is system dependent but 4 KBytes is reasonable since 4 KBytes is a common page size. The local EP processor adds the data buffer addresses (pointers) into the FreeQ to make buffer space available for re-use. The buffer addresses may be added in any order to the FreeQ. When a remote EP processor needs to send data to this local EP processor, the remote EP processor gets a data buffer by removing the first entry from the FreeQ of the local EP processor. The data is then transferred to the data buffer referenced by that entry. Next, the data buffer address is written into the PostQ of the local EP processor. The local EP processor reads the data buffer address from its PostQ and processes the data.

The local EP processor may poll the PostQ periodically to check if there is any new entry in the PostQ. If doorbells are supported, the remote EP processor can assert a doorbell after adding an entry to the PostQ of the local EP processor to notify the local EP processor that a new entry has been added to its PostQ.

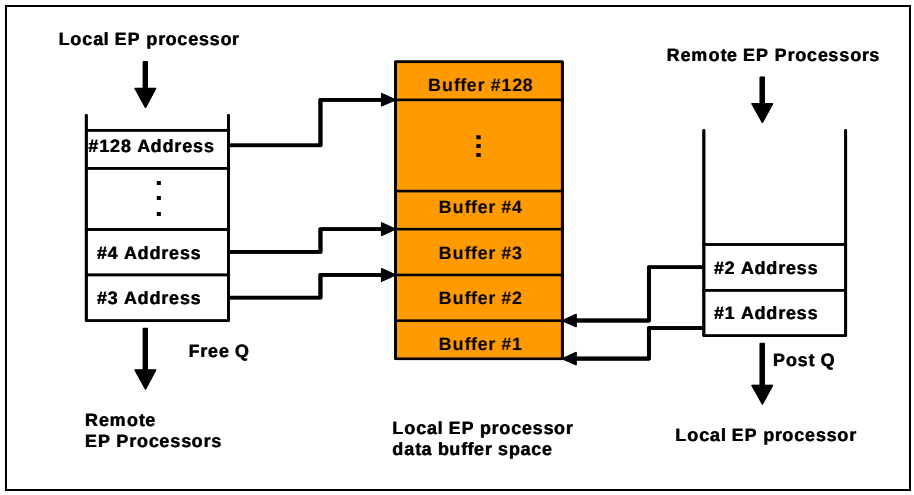

Figure 9 Inbound Queue and Data Buffer Usage

Notes

The data transfer protocol is summarized in Figure 10. In this example, the EP processor in slot 3 (EPP 3) transfers a block of data to the EP processor in slot 2 (EPP 2). EPP 3 first gets a free buffer from the FreeQ on EPP 2. EPP 3 then writes data into the buffer. After all the data has been written, the buffer address is written into the PostQ of EPP 2. EPP 2 gets the buffer address from its PostQ. Then it reads and processes the data in the data buffer. Finally, the buffer is returned to the FreeQ.

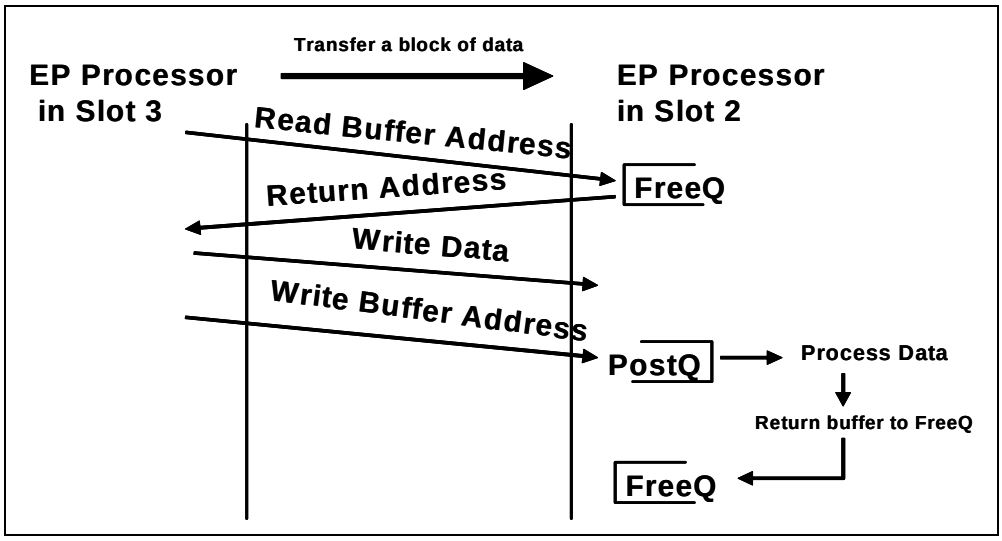

Figure 10 Data Transfer Protocol

Interrupt and Error Reporting

Interrupts in the system domain are handled in the normal PCIe way. Because the EP processor is a native PCIe device, a Message Signaled Interrupt (MSI) is used to deliver the interrupt to the RC processor. A device-dependent method is needed to deliver interrupts to an EP processor. Typically, an EP processor supports a doorbell register which can be used to interrupt an EP processor.

There are two methods to report errors: error messages or a completion status. Error messages are routed to the RC processor. A completion status is a field within the completion header that enables the transaction completer to report errors back to the requester.

An EP processor only makes memory read or write requests to another EP processor. When an error occurs in any EP processor to EP processor request, an error status message is reported back to the requester via the completion status, enabling the EP processor to begin its error handling procedure.

All other system domain related errors, such as data link layer and physical layer errors, are reported to the RC processor which initiates the error handling procedure.

Redundancy

A multi-peer application often requires some level of redundancy. A dual RC processor topology has both an active and a standby RC processor. An example of this dual RC processors topology is shown in Figure 11. A Mux/DeMux device such as the IDT™ PS421 multiplexer/demultiplexer switch connects the Upstream Port (UP) of the PCIe switch to either the RC 1 or 2 processor. The RC processor that is connected to the UP of the PCIe switch is the active RC processor, and the one not connected to the UP is the standby RC processor. During normal operation, the active RC processor is in control of the system and is the Root Complex of the system domain. The standby RC processor has no connection to the system domain. There is an out-of-band connection (not shown) between the RC processors. Heart beat and checkpoint messages are sent periodically from the active RC processor to the standby RC processor. The standby processor monitors the state of the active RC processor.

Notes

The standby RC processor takes over as the active RC processor when a managed switchover is requested or a failure is detected in the active RC processor by the standby RC processor. A managed switchover is one that is initiated by the user for scheduled maintenance or software upgrades or in response to some form of demerit checking within the primary RC processor.

Managed switchover procedure:

- Active RC 1 processor requests all EP processors to stop making requests to the system domain.
- Wait for a short period of time for all outstanding requests to finish. The actual waiting time is system dependent.
- The multiplexer/demultiplexer is switched to connect to the standby RC 2 processor.
- During the switchover, the Upstream port of the switch transitions from the Up state to the Down state, and then back to the Up state. This causes all the Downstream ports to be reset. The EP processors should not reset their data/queue structures during this downstream port reset, so that normal processing can resume from where the EP processors stop before the switchover.
- After the switchover, RC 2 processor becomes the active RC processor. It initializes the PCIe switch to the same state as before the switchover.
- RC 2 processor notifies all EP processors that the system is back to normal and that requests can now be made.

Failure switchover procedure:

- Standby RC 2 processor monitors the state of the active RC 1 processor through heart beat and checkpoint messages.
- When there is a failure in the active RC 1 processor, the standby RC 2 processor takes over as the active RC processor. The RC 2 processor configures the multiplexer/demultiplexer to switch its connection to the RC 2 processor.
- During the switchover, the Upstream port of the switch transitions from the Up state to the Down state, and then back to the Up state. This causes all the Downstream ports to be reset. The EP processors should not reset their data/queue structures during this downstream port reset so that normal processing can resume from where the EP processors stop before the switchover.
- After the switchover, the RC 2 processor becomes the active RC processor. It initializes the PCIe switch to the same state as before the switchover.

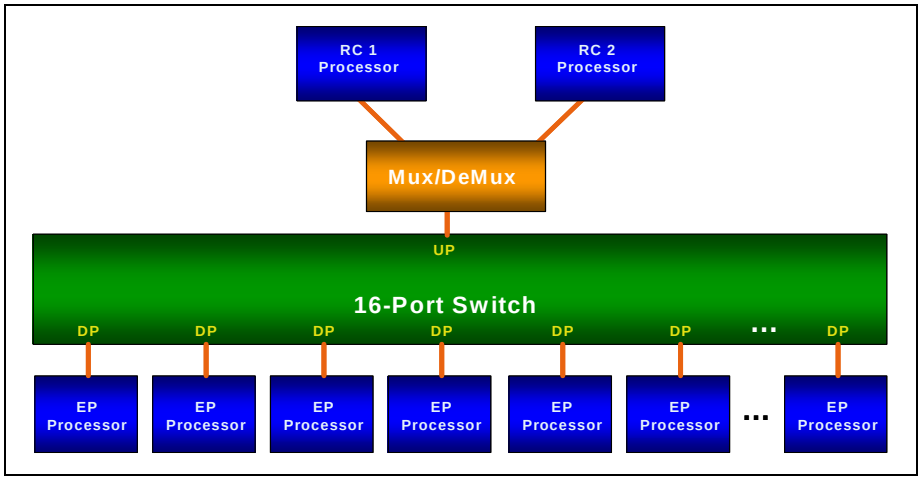

Figure 11 Dual RC Processors Topology

The switch is still a single point of failure in the dual RC processors topology. For a fully redundant system, a dual-star topology may be deployed as shown in Figure 12. In this topology, an additional PCIe switch is added and the multiplexer/demultiplexer is added to the EP processor blade. The PCIe switch is

Notes

added to the RC processor blade, so that the RC processor blade has the RC processor and a 16-port PCIe switch. Each EP processor blade connects to two PCIe switches but only one of the two connections is active.

During normal operation, all EP processors connect to the active RC 1 processor blade, and the RC 2 processor blade is in standby mode. It has the same configuration as the RC 1 processor for the system domain. During a switchover, there is no need to re-initialize the PCIe switch. The active RC processor is in control of the system and is the Root Complex of the system domain. The standby RC processor has no connection to the system domain. There is an out-of-band connection (not shown) between the RC processors. Heart beat and checkpoint messages are sent periodically from the active RC processor to the standby RC processor. The standby RC processor monitors the state of the active RC processor.

The switchover procedure is similar for the Dual RC processors and the Dual-star topology.

Managed switchover procedure:

- Active RC 1 processor requests all EP processors to stop making requests to the system domain.
- Wait for a short period of time for all outstanding requests to finish. The actual waiting time is system dependent.
- The multiplexer/demultiplexers are switched to connect to the standby RC 2 processor.
- After the switchover, the RC 2 processor becomes the active RC processor.
- RC 2 processor notifies all EP processors that the system is back to normal and that requests can now be made.

Failure switchover procedure:

- Standby RC 2 processor monitors the state of the active RC 1 processor through heart beat and checkpoint messages.
- When there is a failure in the active RC 1 processor, the standby RC 2 processor takes over as the active RC processor. The RC 2 processor configures the multiplexer/demultiplexers to switch the EP connections to the RC 2 processor.
- After the switchover, the RC 2 processor becomes the active RC processor.

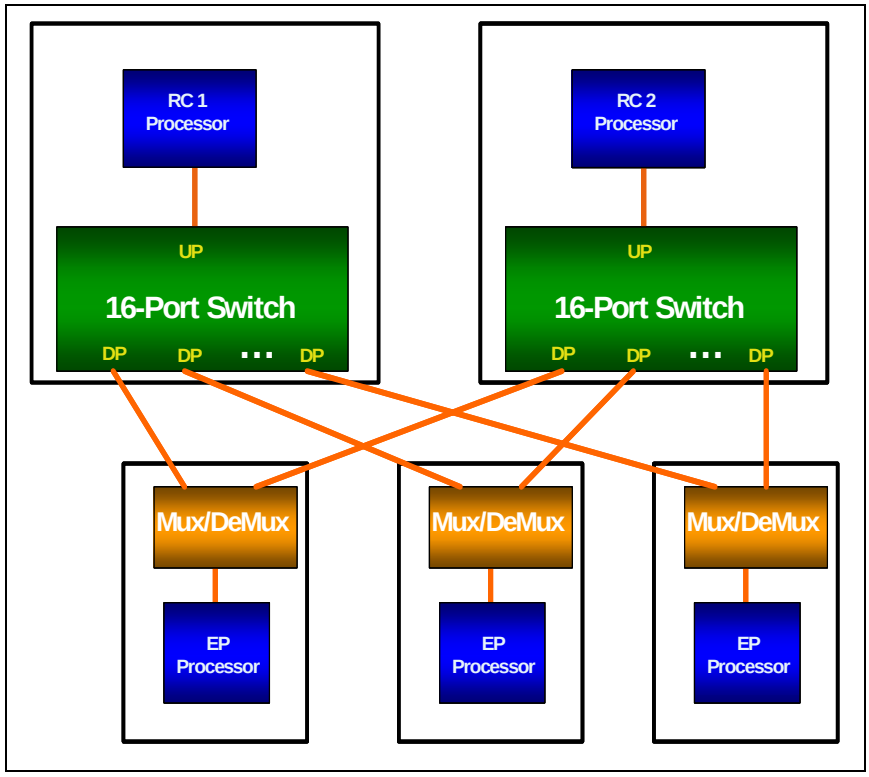

Figure 12 Dual-star Topology

Notes

Summary

Multi-peer support can be achieved using a standard PCIe switch without any proprietary solutions. A few EP processors have been identified to be part of the solution of a multi-peer system. System software in the RC processor has to be extended to support the allocation of the memory map in the system domain. A simple peer-to-peer communication protocol has been shown to transfer large blocks of data between EP processors using PCIe memory read and write operations.

Redundancy is important in a multi-peer system. A dual RC processor topology and a dual-star topology have been identified as providing different levels of redundancy.

Reference

[1] PCI Express Base specification Revision 1.1

[2] Intelligent I/O Architecture Specification Version 2.0, I2O Special Interest Group