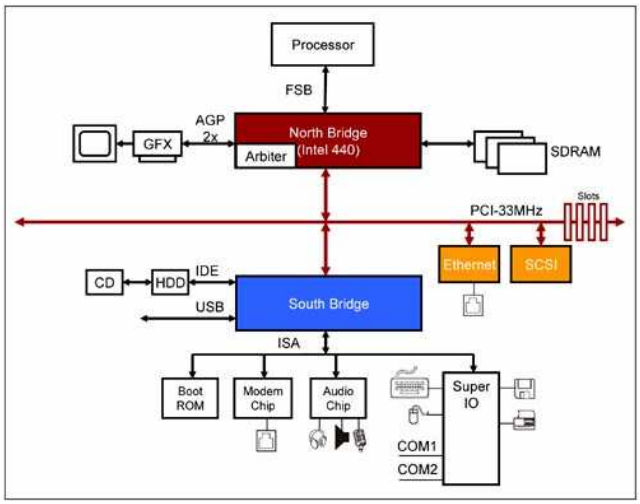
Introduction to the PCI Express interface

The PCI Express interface is a third-generation, high-speed interconnect for use in computer systems. It is supposed to take over from current second-generation interfaces like PCI, PCI-X and AGP.
This document is based on the book *PCI Express System Architecture*[1] (PCIeSA), and will provide an introduction to key features and concepts of the PCI Express interface, mainly by comparing it to the older PCI interface.

1 PCI Express vs. PCI

One of the main differences between PCI Express and its predecessor PCI lies in the physical design. The PCI interface is a parallel bus running at 33 or 66 MHz, with support for attaching several PCI devices to the same bus. The parallel structure and the support for several devices (hence the term bus), makes it hard to raise the clock speed above what is currently used while still maintaining signal integrity.
PCI Express however, uses point-to-point links, consisting of one or more serial differential signaling lanes each capable of transferring 2.5 Gb/s. (Although often being referred to as a bus, it actually is not. PCI Express devices do not share links or lanes - connecting several devices is supported through the use of so-called PCI Express "switches" in the motherboard chipset instead.)
The distinction between lanes and links is important: a lane is a pair of differential signal-pairs (one pair running in each direction, so a total of 4 signals per lane), while a link is the actual interface, which can consist of one or more lanes. A link can for example be designated as a PCI Express x16 interface, indicating that it uses 16 lanes.
An example PCI-based platform can be seen in figure 1.a, and an example PCI Express platform in figure 1.b.



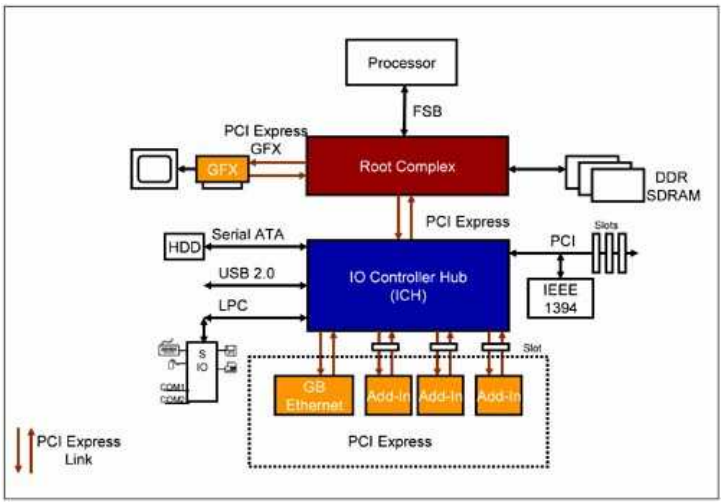


Figure 1.a (left): An example of a 33 MHz PCI bus based platform (Figure 1-2 from PCIeSA).
Figure 1.b (right): An example of a PCI Express system (Figure 1-23 from PCIeSA).

Another difference is in the way PCI Express uses a packet-based protocol instead of the address/data-bus used in PCI. PCI Express also uses in-band signaling through the packet-based protocol for sending signals (interrupts, power management, etc), where the PCI interface uses dedicated pins. The clock signal is incorporated into the serial data stream by using 8b/10b encoding to guarantee either a 0-to-1 or a 1-to-0 transition every 10 bits. A PLL in either end of the transmission line can then recover the clock signal from these transitions.
There are however, also similarities between PCI Express and PCI. The most notable being that PCI Express implements the same usage and load-store communication models as PCI, which results in

[1] See the literature list for further information.

PCI Express being software backwards compatible with PCI. PCI device drivers for the host OS will not need to be changed to work with a similar PCI Express device.

2 Performance

The current standard, PCI Express v1.1, has a maximum transfer speed of 2.5 Gb/s for each lane. With the 8b/10b encoding, this results in a maximum performance of 250 MB/s in each direction, per lane (10 bits are used to transmit 1 byte). In comparison, a standard 33 MHz, 32 bit PCI interface has a maximum performance of 133 MB/s (it is a bus, so this is the total for both directions).
As PCI Express links can consist of up to 32 lanes, much higher speeds are achievable. The packets are byte-striped across available lanes, so the speed scales linearly with the number of lanes (from the 250 MB/s/direction for a 1x link to 4 GB/s/direction for a 16x link, like those commonly used for graphic cards).

3 The configuration registers

PCI and PCI Express devices have a number of configuration registers which store important information specific to the devices. This includes Device/Vendor IDs, class codes that specify what the device does (graphics card/network interface card/etc), and information about the Base Address Registers (BARs), see figure 3.a.

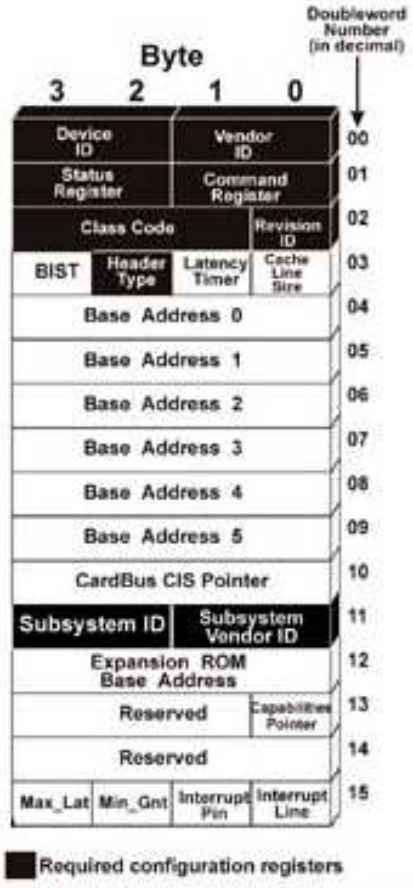


Figure 3.a: The PCI/PCI Express configuration header (Adapted from figure 1-13 in PCIeSA).

In addition to this configuration header, 192 bytes of device specific configuration registers are also available, providing in total 256 bytes of configuration space. PCI Express devices also have an additional 3840 bytes of extended configuration space available, which can be used for extended capabilities such as advanced error reporting and power budgeting.
The BARs are the primary means of communication from a host system to a PCI Express device. These can be of type I/O or memory. Memory BARs offer better performance than I/O BARs, as

they can be prefetched, and allow posted writes (see section 5). Two memory BARs can also be combined into one 64 bit BAR, providing up to 8 exabytes of memoryspace. The use of I/O-type BARs is discouraged by for instance Microsoft[1], as the amount of I/O space available (64 kB) on x86 systems is very scarce compared to the amount of memory space (4 GB).

4 PCI Express layered protocol

PCI Express uses a three-layered protocol. The layers are the transaction layer, the data link layer, and the physical layer. See figure 4.a.

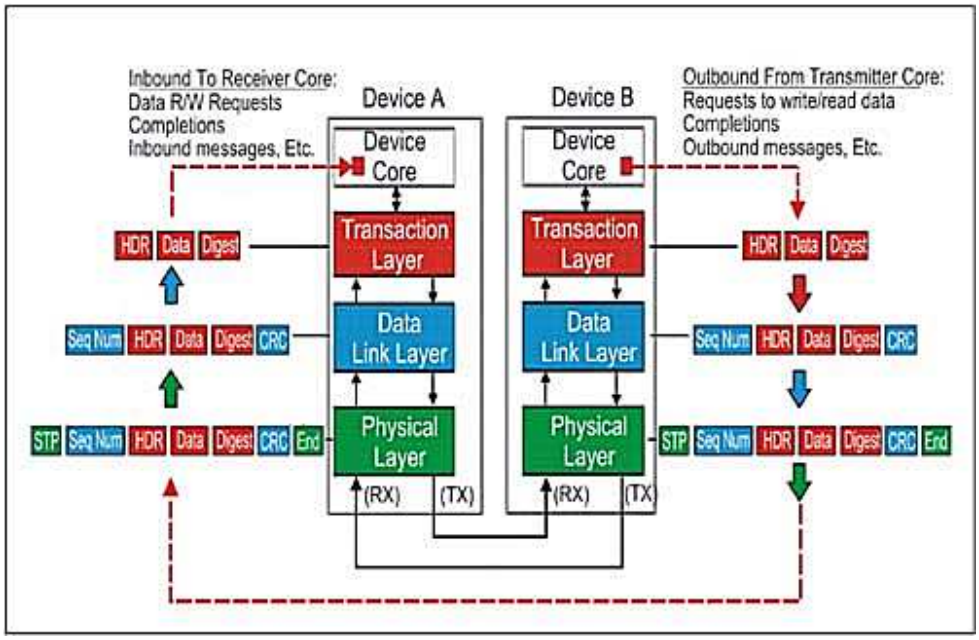


Figure 4.a: Two PCI Express devices, each with the three-layered protocol (Adapted from figure 4-2 in PCIeSA).

The layers have the following purposes:

- *Transaction layer, handles TLP's (Transaction Layer Packets):* The transaction layer accepts, buffers and disseminates TLP's from and to the device core. It creates a header containing information about the command to be performed, addresses, and various control flags. This layer also takes care of ordering and certain parts of the flow control.
- *Data link layer, handles DLLP's (Data Link Layer Packets):* The data link layer provides a reliable method for transferring TLP's between two devices, and handles initialization, flow control, error detection and recovery. The data link layer adds a CRC value and a sequence number to the TLP.
- *Physical layer:* The physical layer converts the DLLP's to an appropriate format, for transmission over the PCI Express interface. Certain implementations (like the Spartan3 PCI Express Starter kit) have the physical layer split in two, connected with a PXPIPE interface. Here a separate PHY chip converts the serial 2.5 Gb/s 8b/10b encoded stream to for instance an 8 bit 250 MB/s parallel stream. This way only the PHY chip needs to be able to handle the very high frequencies (GHz range) experienced on the PCI Express interface, while the rest of the physical layer implementation can run at a few hundred MHz.

5 PCI Express transaction model

There are basically four different types of transactions available in a PCI Express system: Memory, I/O, configuration and message transactions (all these, except for message transactions, are also available in PCI systems). These can be divided into Non-posted or posted transactions. Non-posted

[1] See the file "*Misc\IO Resource usage reduction.pdf*" on CD1

transactions use a split transaction protocol with a requester and a completer. The requester sends a request packet (for example a memory read request), and the completer responds with a completion packet containing the requested data. Posted transactions consist only of a request (for example a memory write), without a response.
The split transaction protocol also allows for delayed responses. Using a PCI interface, if a device is too busy to handle a request at the time it is received, the request will have to be resent later, which can be rather ineffective. However, using the split transaction protocol, requests can be memorized and stored until they can be processed. After receiving a request that can not be serviced immediately, a split response is sent to the requester. The requester can then go back to working on other tasks, until the completer is able to process the stored request, and responds with a completion packet.
Requesters are able to handle several outstanding request packets by using a tag on each request packet. The requester/completer model can be seen on figure 5.a.

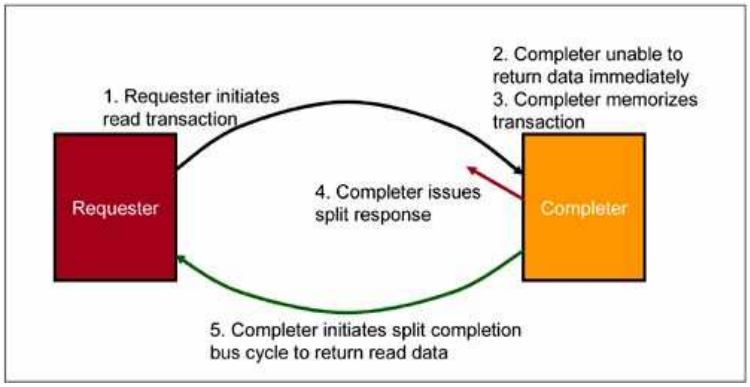


Figure 5.a: An example showing the requester/completer structure of a PCI Express system (Figure 1-18 from PCIeSA).

A complete list of the different packet types in the PCI Express protocol can be seen in table 5.b.

TLP Packet Types	Abbreviated Name
Requests	
Memory Read Request (Non-posted)	MRd
Memory Read Request - Locked access (Non-posted)	MRdLk
Memory Write Request (Posted)	MWr
IO Read (Non-posted)	IORd
IO Write (Non-posted)	IOWr
Configuration Read (Type 0 and Type 1) (Non-posted)	CfgRd0, CfgRd1
Configuration Write (Type 0 and Type 1) (Non-posted)	CfgWr0, CfgWr1
Message Request without Data (Posted)	Msg
Message Request with Data (Posted)	MsgD
Responses	
Completion without Data	Cpl
Completion with Data	CplD
Completion without Data - associated with Locked Memory Read Requests	CplLk
Completion with Data - associated with Locked Memory Read Requests	CplDLk

Table 5.b: The different packet types in a PCI Express system (Adapted from table 4-2 in PCIeSA).

Sending a locked memory read request causes the path from requester to completer to become locked, until an unlock message is sent. Only a root complex is allowed to initiate a locked request.

6 PCI Express Transaction Layer packets

The TLP packets of most interest are the 3DW memory requests (read or write), and their completions. These are used for reading and writing 32 bit memory BARs, and for certain, simple systems, these are all that are needed to communicate between a PCI Express device and a host system. The headers for these TLPs can be seen in figure 6.a and 6.c.

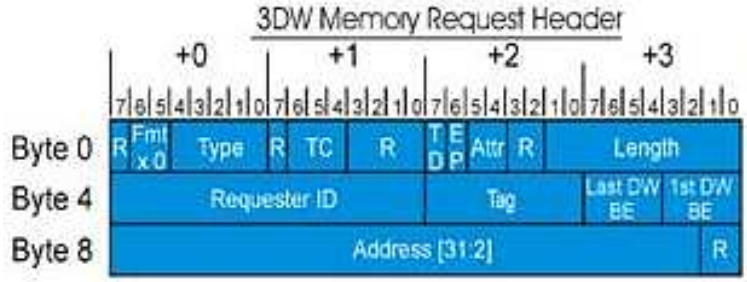


Figure 6.a: Headers for 3DW memory request (Adapted from figure 4-8 in PCIeSA).

The contents of the various fields can be seen in table 6.b.

Field Name	Byte/Bit	Function
Length [9:0]	Byte 3 Bit 7:0 Byte 2 Bit 1:0	TLP data payload transfer size, in DW. Maximum transfer size is 10 bits, 2^{10} = 1024 DW (4KB). Encoding: 00 0000 0001b = 1DW, 00 0000 0010b = 2DW, (…), 11 1111 1111b = 1023DW, 00 0000 0000b = 1024DW
Attr (Attributes)	Byte 2 Bit 5:4	Bit 5 = Relaxed ordering: When set = 1, PCI-X relaxed ordering is enabled for this TLP. If set = 0, then strict PCI ordering is used. Bit 4 = No Snoop: When set = 1, requester is indicating that no host cache coherency issues exist with respect to this TLP. System hardware is not required to cause processor cache snoop for coherency. When set = 0, PCI -type cache snoop protection is required.
EP (Poisoned Data)	Byte 2 Bit 6	If set = 1, the data accompanying this data should be considered invalid although the transaction is being allowed to complete normally.
TD (TLP Digest Field Present)	Byte 2 Bit 7	If set = 1, the optional 1 DW TLP Digest field is included with this TLP.
TC (Traffic Class)	Byte 1 Bit 6:4	These three bits are used to encode the traffic class to be applied to this TLP and to the completion associated with it (if any). 000b = Traffic Class 0 (Default), (…), 111b = Traffic Class 7 TC 0 is the default class, and TC 1-7 are used in providing differentiated services.
Type[4:0]	Byte 0 Bit 4:0	TLP packet Type field: 00000b = Memory Read or Write, 00001b = Memory Read Locked Type field is used with Fmt [1:0] field to specify transaction type, header size, and whether data payload is present.
Fmt 1:0 (Format)	Byte 0 Bit 6:5	Packet Format: 00b = Memory Read (3DW w/o data), 10b = Memory Write (3DW w/ data)
1st DW BE 3:0	Byte 7 Bit 3:0	These high true bits map one-to-one to qualify bytes within the DW payload.
Last DW BE 3:0	Byte 7 Bit 7:4	These high true bits map one-to-one to qualify bytes within the last DW transferred.
Tag 7:0	Byte 6 Bit 7:0	These bits are used to identify each outstanding request issued by the requester. As non-posted requests are sent, the next sequential tag is assigned. Default: only bits 4:0 are used (32 outstanding transactions at a time) If Extended Tag bit in PCI Express Control Register is set = 1, then all 8 bits may be used (256 tags).
Requester ID 15:0	Byte 5 Bit 7:0 Byte 4 Bit 7:0	Identifies the requester so a completion may be returned, etc. Byte 4, 7:0 = Bus Number Byte 5, 7:3 = Device Number Byte 5, 2:0 = Function Number
Address 31:2	Byte 11 Bit 7:2 Byte 10 Bit 7:0 Byte 9 Bit 7:0 Byte 8 Bit 7:0	The start address for the memory transfer. Note that the lower two bits of the 32 bit address are reserved (00b), forcing the start address to be DW aligned.

Table 6.b: Description of the various fields in a 3DW memory request header (Adapted from table 4-7 in PCIeSA).

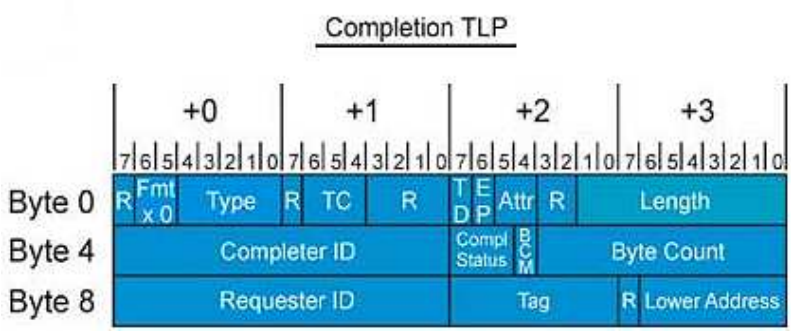


Figure 6.c: Headers for 3DW memory completion (Adapted from figure 4-10 in PCIeSA).

The contents of the various fields can be seen in table 6.d.

Field Name	Byte/Bit	Function
Length 9:0	Byte 3 Bit 7:0 Byte 2 Bit 1:0	Indicates data payload size in DW. For completions, this field reflects the size of the data payload associated with this completion.
Attr 1:0 (Attributes)	Byte 2 Bit 5:4	Attribute 1: Relaxed Ordering Bit Attribute 0: No Snoop Bit For a completion, both of these bits are set to same state as in the request.
EP	Byte 2 Bit 6	If = 1, indicates the data payload is poisoned.
TD	Byte 2 Bit 7	If = 1, indicates the presence of a digest field (1 DW) at the end of the TLP (preceding LCRC and END)
TC 2:0 (Transfer Class)	Byte 2 Bit 6:4	Indicates transfer class for the packet. For a completion, TC is set to same value as in the request.
Type 4:0	Byte 0 Bit 4:0	TLP packet type field. Always set to 01010b for a completion.
Fmt 1:0 (Format)	Byte 0 Bit 6:5	Packet Format. Always a 3DW header 00b = Completion without data (Cpl), 10b = Completion with data (CplD)
Byte Count	Byte 7 Bit 7:0 Byte 6 Bit 3:0	This is the remaining byte count until a read request is satisfied. Generally, it is derived from the original request Length field.
BCM (Byte Count Modified)	Byte 6 Bit 4	Set = 1 only by PCI-X completers. Indicates that the byte count field (see previous field) reflects the first transfer payload rather than total payload remaining.
CS 2:0 (Completion Status Code)	Byte 6 Bit 7:5	These bits encoded by the completer to indicate success in fulfilling the request. 000b = Successful Completion (SC), 001b = Unsupported Request (UR) 010b = Config Req Retry Status (CR S), 100b = Completer abort. (CA) others: reserved
Completer ID 15:0	Byte 5 Bit 7:0 Byte 4 Bit 7:0	Identifies the completer. While not needed for routing a completion, this information may be useful if debugging bus traffic. Byte 4 7:0 = Completer Bus # Byte 5 7:3 = Completer Dev # Byte 5 2:0 = Completer Function #
Lower Address 6:0	Byte 11 Bit 6:0	The lower 7 bits of address for the first enabled byte of data returned with a read. Calculated from request Length and Byte enables, it is used to determine next legal Read Completion Boundary
Tag 7:0	Byte 10 Bit 7:0	These bits are set to reflect the Tag received with the request. The requester uses them to associate inbound completion with an outstanding request.
Requester ID 15:0	Byte 9 Bit 7:0 Byte 8 Bit 7:0	Copied from the request into this field to be used in routing the completion back to the original requester. Byte 4, 7:0 = Requester Bus # Byte 5, 7:3 = Requester Device # Byte 5, 2:0 = Requester Function #

Table 6.d: Description of the various fields in a completion header (Adapted from table 4-9 in PCIeSA).

7 Communicating with a PCI Express device

When the host system installs and configures its PCI Express devices during startup, it reads the configuration headers, and typically maps the BARs into a virtual memory space. They can then be accessed through standard memory read and write functions from device drivers, or be mapped further on to the address space of a user-mode application.
On Windows systems, BARs can be accessed from device drivers by using kernel mode calls such as *READ_REGISTER_<datatype>* (where <datatype> is the type of data to read, for instance "ULONG" for a 32 bit value). The device driver can then be queried from user-mode applications using the *DeviceIoControl()* function. This can be seen in figure 7.a.

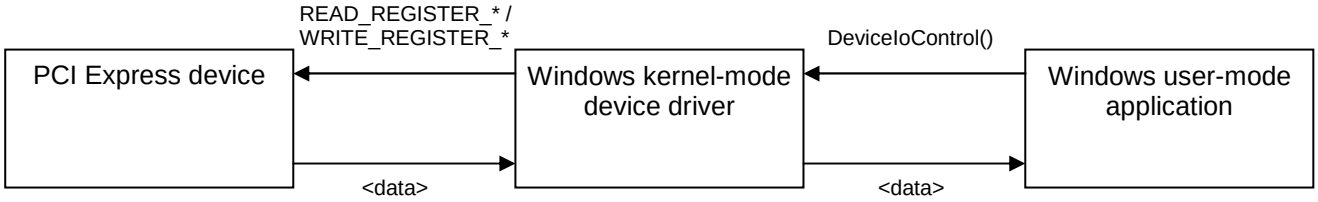


Figure 7.a: An example communication with a PCI Express device on a Windows system.

On Linux, communication happens almost the same way, just using different system calls. Between the device driver and the device *ioread<size>* and *iowrite<size>* are used (where <size> is the amount of bits to read, 8, 16 or 32). Between the application and the device driver, the call *ioctl()* is used.

8 Summary

To sum up:

- PCI Express uses serial point-to-point communication opposed to PCIs parallel bus structure
- PCI Express uses packet-based communication with signals such as interrupt, power management and similar inband, while PCI uses an address/data-bus and dedicated pins for most signals
- PCI Express and PCI use the same configuration space and communication model as far as the host system is concerned. PCI Express is therefore software backwards compatible with PCI.
- For a PCI Express 1x link, the maximum theoretical bandwidth is 250 MB/s/direction.
- For communication, PCI Express devices use base address registers that are mapped into the virtual memory space of the host system.