Computer Architecture
Lecture No.7

Lecture Outlines

3.6 PCI Express

PCI Physical and Logical Architecture
PCIe Physical Layer
PCIe Transaction Layer
PCIe Data Link Layer

3.6 PCI EXPRESS

The **peripheral component interconnect (PCI)** is a popular high-bandwidth, processor-independent bus that can function as a mezzanine or peripheral bus. Compared with other common bus specifications, PCI delivers better system performance for high-speed I/O subsystems (e.g., graphic display adapters, network interface controllers, and disk controllers).

Intel began work on PCI in 1990 for its Pentium-based systems. Intel soon released all the patents to the public domain and promoted the creation of an industry association, the PCI Special Interest Group (SIG), to develop further and maintain the compatibility of the PCI specifications. The result is that PCI has been widely adopted and is finding increasing use in personal computer, workstation, and server systems. Because the specification is in the public domain and is supported by a broad cross-section of the microprocessor and peripheral industry, PCI products built by different vendors are compatible.

As with the system bus discussed in the preceding sections, the bus-based PCI scheme has not been able to keep pace with the data rate demands of attached devices. Accordingly, a new version, known as **PCI Express (PCIe)** has been developed. PCIe, as with QPI, is a point-to-point interconnect scheme intended to replace bus-based schemes such as PCI.

A key requirement for PCIe is high capacity to support the needs of higher data rate I/O devices, such as Gigabit Ethernet. Another requirement deals with the need to support time-dependent data streams. Applications such as video-on-demand and audio redistribution are putting real-time constraints on servers too. Many communications applications and embedded PC control systems also process data in real-time. Today's platforms must also deal with multiple concurrent

transfers at ever-increasing data rates. It is no longer acceptable to treat all data as equal—it is more important, for example, to process streaming data first since late real-time data is as useless as no data. Data needs to be tagged so that an I/O system can prioritize its flow throughout the platform.

PCI Physical and Logical Architecture

Figure 3.21 shows a typical configuration that supports the use of PCIe. A **root complex** device, also referred to as a *chipset* or a *host bridge*, connects the processor and memory subsystem to the PCI Express switch fabric comprising one or more PCIe and PCIe switch devices. The root complex acts as a buffering device, to deal with difference in data rates between I/O controllers and memory and processor components. The root complex also translates between PCIe transaction formats and the processor and memory signal and control requirements. The chipset will typically support multiple PCIe ports, some of which attach directly to a PCIe device, and one or more that attach to a switch that manages multiple PCIe streams. PCIe links from the chipset may attach to the following kinds of devices that implement PCIe:

- **Switch:** The switch manages multiple PCIe streams.
- **PCIe endpoint:** An I/O device or controller that implements PCIe, such as a Gigabit ethernet switch, a graphics or video controller, disk interface, or a communications controller.

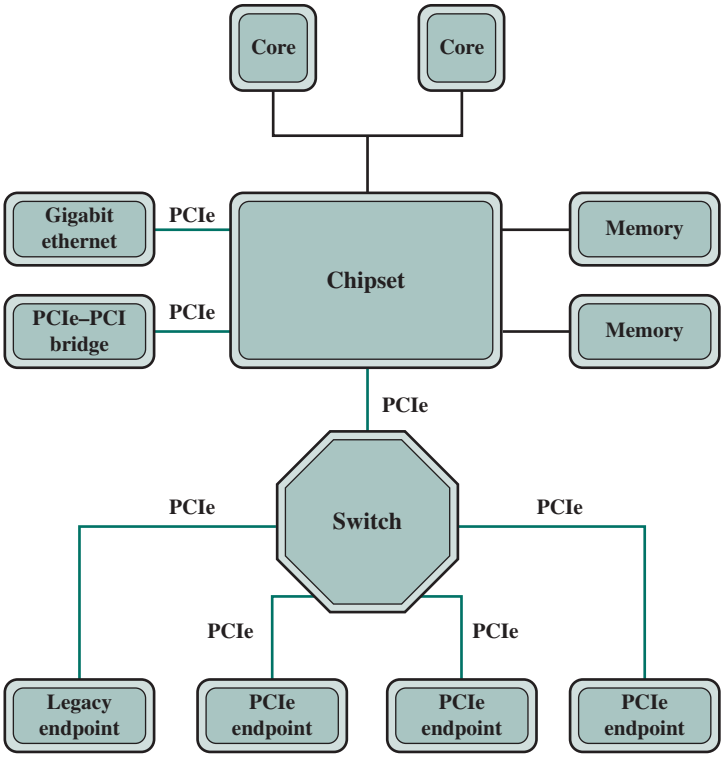


Figure 3.21 Typical Configuration Using PCIe

- **Legacy endpoint:** Legacy endpoint category is intended for existing designs that have been migrated to PCI Express, and it allows legacy behaviors such as use of I/O space and locked transactions. PCI Express endpoints are not permitted to require the use of I/O space at runtime and must not use locked transactions. By distinguishing these categories, it is possible for a system designer to restrict or eliminate legacy behaviors that have negative impacts on system performance and robustness.
- **PCIe/PCI bridge:** Allows older PCI devices to be connected to PCIe-based systems.

As with QPI, PCIe interactions are defined using a protocol architecture. The PCIe protocol architecture encompasses the following layers (Figure 3.22):

- **Physical:** Consists of the actual wires carrying the signals, as well as circuitry and logic to support ancillary features required in the transmission and receipt of the 1s and 0s.
- **Data link:** Is responsible for reliable transmission and flow control. Data packets generated and consumed by the DLL are called Data Link Layer Packets (DLLPs).
- **Transaction:** Generates and consumes data packets used to implement load/store data transfer mechanisms and also manages the flow control of those packets between the two components on a link. Data packets generated and consumed by the TL are called Transaction Layer Packets (TLPs).

Above the TL are software layers that generate read and write requests that are transported by the transaction layer to the I/O devices using a packet-based transaction protocol.

PCIe Physical Layer

Similar to QPI, PCIe is a point-to-point architecture. Each PCIe port consists of a number of bidirectional lanes (note that in QPI, the lane refers to transfer in one direction only). Transfer in each direction in a lane is by means of differential signaling over a pair of wires. A PCI port can provide 1, 4, 6, 16, or 32 lanes. In what follows, we refer to the PCIe 3.0 specification, introduced in late 2010.

As with QPI, PCIe uses a multilane distribution technique. Figure 3.23 shows an example for a PCIe port consisting of four lanes. Data are distributed to the four

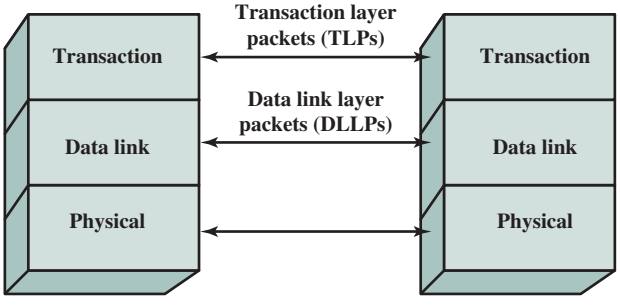


Figure 3.22 PCIe Protocol Layers

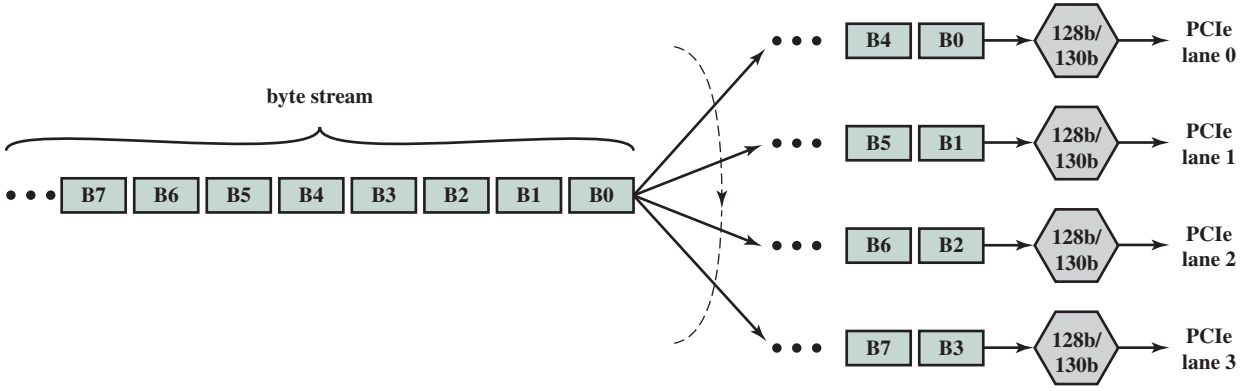


Figure 3.23 PCIe Multilane Distribution

lanes 1 byte at a time using a simple round-robin scheme. At each physical lane, data are buffered and processed 16 bytes (128 bits) at a time. Each block of 128 bits is encoded into a unique 130-bit codeword for transmission; this is referred to as 128b/130b encoding. Thus, the effective data rate of an individual lane is reduced by a factor of 128/130.

To understand the rationale for the 128b/130b encoding, note that unlike QPI, PCIe does not use its clock line to synchronize the bit stream. That is, the clock line is not used to determine the start and end point of each incoming bit; it is used for other signaling purposes only. However, it is necessary for the receiver to be synchronized with the transmitter, so that the receiver knows when each bit begins and ends. If there is any drift between the clocks used for bit transmission and reception of the transmitter and receiver, errors may occur. To compensate for the possibility of drift, PCIe relies on the receiver synchronizing with the transmitter based on the transmitted signal. As with QPI, PCIe uses differential signaling over a pair of wires. Synchronization can be achieved by the receiver looking for transitions in the data and synchronizing its clock to the transition. However, consider that with a long string of 1s or 0s using differential signaling, the output is a constant voltage over a long period of time. Under these circumstances, any drift between the clocks of transmitter and receiver will result in loss of synchronization between the two.

A common approach, and the one used in PCIe 3.0, to overcoming the problem of a long string of bits of one value is scrambling. Scrambling, which does not increase the number of bits to be transmitted, is a mapping technique that tends to make the data appear more random. The scrambling tends to spread out the number of transitions so that they appear at the receiver more uniformly spaced, which is good for synchronization. Also, other transmission properties, such as spectral properties, are enhanced if the data are more nearly of a random nature rather than constant or repetitive. For more discussion of scrambling, see Appendix E.

Another technique that can aid in synchronization is encoding, in which additional bits are inserted into the bit stream to force transitions. For PCIe 3.0, each group of 128 bits of input is mapped into a 130-bit block by adding a 2-bit block sync header. The value of the header is 10 for a data block and 01 for what is called an *ordered set block*, which refers to a link-level information block.

Figure 3.24 illustrates the use of scrambling and encoding. Data to be transmitted are fed into a scrambler. The scrambled output is then fed into a 128b/130b encoder, which buffers 128 bits and then maps the 128-bit block into a 130-bit block. This block then passes through a parallel-to-serial converter and transmitted one bit at a time using differential signaling.

At the receiver, a clock is synchronized to the incoming data to recover the bit stream. This then passes through a serial-to-parallel converter to produce a stream of 130-bit blocks. Each block is passed through a 128b/130b decoder to recover the original scrambled bit pattern, which is then descrambled to produce the original bit stream.

Using these techniques, a data rate of 16 GB/s can be achieved. One final detail to mention; each transmission of a block of data over a PCI link begins and ends with an 8-bit framing sequence intended to give the receiver time to synchronize with the incoming physical layer bit stream.

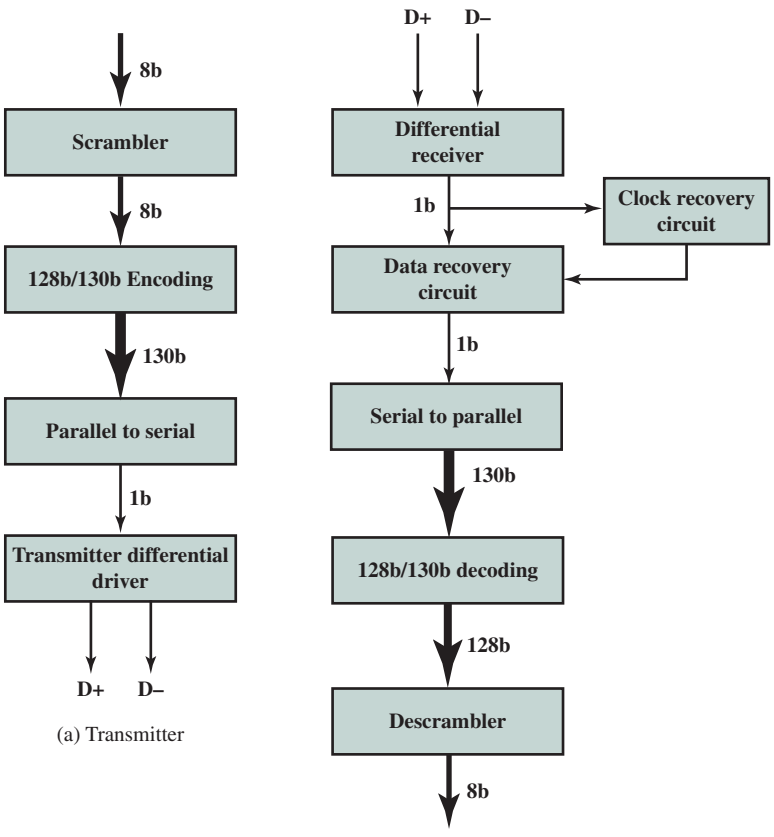


Figure 3.24 PCIe Transmit and Receive Block Diagrams

PCIe Transaction Layer

The transaction layer (TL) receives read and write requests from the software above the TL and creates request packets for transmission to a destination via the link layer. Most transactions use a *split transaction* technique, which works in the following fashion. A request packet is sent out by a source PCIe device, which then waits for a response, called a *completion* packet. The completion following a request is initiated by the completer only when it has the data and/or status ready for delivery. Each packet has a unique identifier that enables completion packets to be directed to the correct originator. With the split transaction technique, the completion is separated in time from the request, in contrast to a typical bus operation in which both sides of a transaction must be available to seize and use the bus. Between the request and the completion, other PCIe traffic may use the link.

TL messages and some write transactions are *posted transactions*, meaning that no response is expected.

The TL packet format supports 32-bit memory addressing and extended 64-bit memory addressing. Packets also have attributes such as "no-snoop,"

"relaxedordering," and "priority," which may be used to optimally route these packets through the I/O subsystem.

ADDRESS SPACES AND TRANSACTION TYPES The TL supports four address spaces:

- **Memory:** The memory space includes system main memory. It also includes PCIe I/O devices. Certain ranges of memory addresses map into I/O devices.
- **I/O:** This address space is used for legacy PCI devices, with reserved memory address ranges used to address legacy I/O devices.
- **Configuration:** This address space enables the TL to read/write configuration registers associated with I/O devices.
- **Message:** This address space is for control signals related to interrupts, error handling, and power management.

Table 3.2 shows the transaction types provided by the TL. For memory, I/O, and configuration address spaces, there are read and write transactions. In the case of memory transactions, there is also a read lock request function. Locked operations occur as a result of device drivers requesting atomic access to registers on a PCIe device. A device driver, for example, can atomically read, modify, and then write to a device register. To accomplish this, the device driver causes the processor to execute an instruction or set of instructions. The root complex converts these processor instructions into a sequence of PCIe transactions, which perform individual read and write requests for the device driver. If these transactions must be executed atomically, the root complex locks the PCIe link while executing the transactions. This locking prevents transactions that are not part of the sequence from occurring. This sequence of transactions is called a locked operation. The particular set

Table 3.2 PCIe TLP Transaction Types

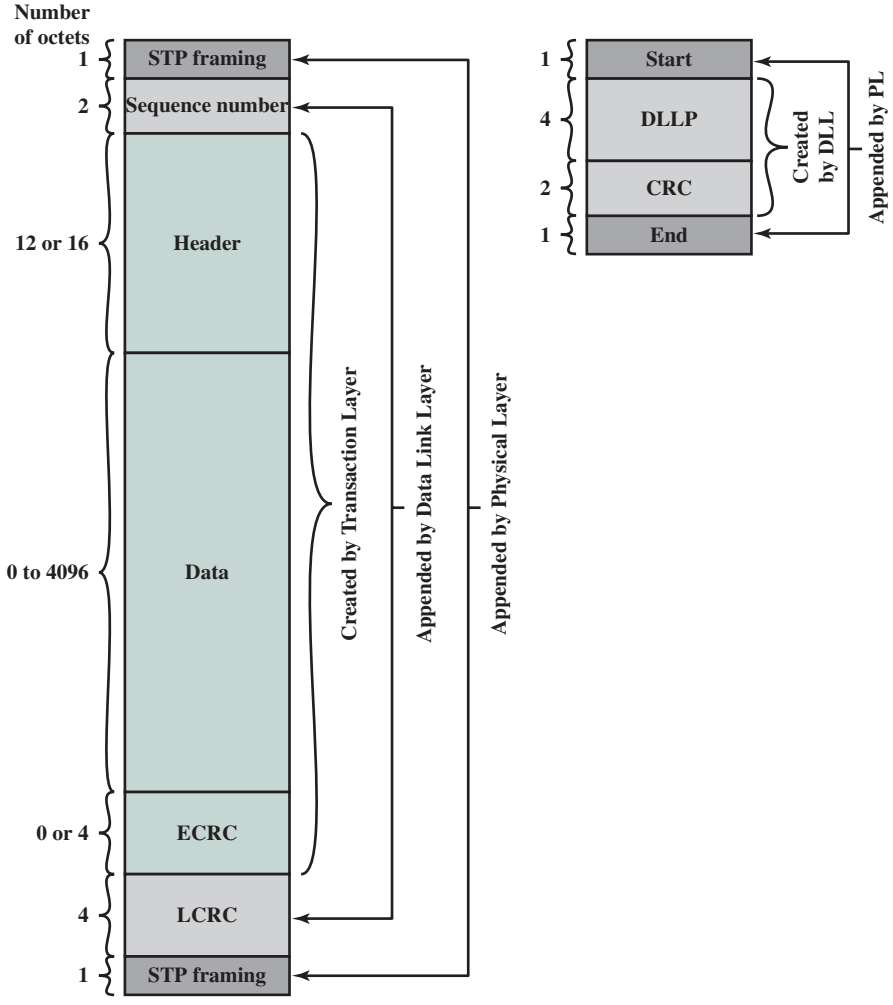
Address Space	TLP Type	Purpose
Memory	Memory Read Request	Transfer data to or from a location in the system memory map.
	Memory Read Lock Request	
	Memory Write Request	
I/O	I/O Read Request	Transfer data to or from a location in the system memory map for legacy devices.
	I/O Write Request	
Configuration	Config Type 0 Read Request	Transfer data to or from a location in the configuration space of a PCIe device.
	Config Type 0 Write Request	
	Config Type 1 Read Request	
	Config Type 1 Write Request	
Message	Message Request	Provides in-band messaging and event reporting.
	Message Request with Data	
Memory, I/O, Configuration	Completion	Returned for certain requests.
	Completion with Data	
	Completion Locked	
	Completion Locked with Data	

of processor instructions that can cause a locked operation to occur depends on the system chip set and processor architecture.

To maintain compatibility with PCI, PCIe supports both Type 0 and Type 1 configuration cycles. A Type 1 cycle propagates downstream until it reaches the bridge interface hosting the bus (link) that the target device resides on. The configuration transaction is converted on the destination link from Type 1 to Type 0 by the bridge.

Finally, completion messages are used with split transactions for memory, I/O, and configuration transactions.

TLP PACKET ASSEMBLY PCIe transactions are conveyed using transaction layer packets, which are illustrated in Figure 3.25a. A TLP originates in the transaction layer of the sending device and terminates at the transaction layer of the receiving device.



(a) Transaction Layer Packet

(b) Data Link Layer Packet

Figure 3.25 PCIe Protocol Data Unit Format

Upper layer software sends to the TL the information needed for the TL to create the core of the TLP, which consists of the following fields:

- **Header:** The header describes the type of packet and includes information needed by the receiver to process the packet, including any needed routing information. The internal header format is discussed subsequently.
- **Data:** A data field of up to 4096 bytes may be included in the TLP. Some TLPs do not contain a data field.
- **ECRC:** An optional end-to-end CRC field enables the destination TL layer to check for errors in the header and data portions of the TLP.

PCIe Data Link Layer

The purpose of the PCIe data link layer is to ensure reliable delivery of packets across the PCIe link. The DLL participates in the formation of TLPs and also transmits DLLPs.

DATA LINK LAYER PACKETS Data link layer packets originate at the data link layer of a transmitting device and terminate at the DLL of the device on the other end of the link. Figure 3.25b shows the format of a DLLP. There are three important groups of DLLPs used in managing a link: flow control packets, power management packets, and TLP ACK and NAK packets. Power management packets are used in managing power platform budgeting. Flow control packets regulate the rate at which TLPs and DLLPs can be transmitted across a link. The ACK and NAK packets are used in TLP processing, discussed in the following paragraphs.

TRANSACTION LAYER PACKET PROCESSING The DLL adds two fields to the core of the TLP created by the TL (Figure 3.25a): a 16-bit sequence number and a 32-bit link-layer CRC (LCRC). Whereas the core fields created at the TL are only used at the destination TL, the two fields added by the DLL are processed at each intermediate node on the way from source to destination.

When a TLP arrives at a device, the DLL strips off the sequence number and LCRC fields and checks the LCRC. There are two possibilities:

1. If no errors are detected, the core portion of the TLP is handed up to the local transaction layer. If this receiving device is the intended destination, then the TL processes the TLP. Otherwise, the TL determines a route for the TLP and passes it back down to the DLL for transmission over the next link on the way to the destination.
2. If an error is detected, the DLL schedules an NAK DLL packet to return back to the remote transmitter. The TLP is eliminated.

When the DLL transmits a TLP, it retains a copy of the TLP. If it receives an NAK for the TLP with this sequence number, it retransmits the TLP. When it receives an ACK, it discards the buffered TLP.