PCI Express Basics

Accelerated Systems 046278 - EE Technion

Amir Watad
Why?

- Connecting Peripherals
  - Disks, Network Cards, GPUs
History

• 1980’s: IBM’s AT / ISA
  • 16 bits, large connectors
  • Several attempts to improve: MCA, EISA, VESA
  • Didn’t succeed
Then came PCI

• Open Standard

• PCI = Peripheral Component Interconnect

• By PCISIG (PCI Special Interest Group)

• Higher bandwidth than ISA. Better visibility and control to software.

• PCI, PCI-X, then PCIe. We’ll talk about PCIe only.
Basics

• Link: A path between two devices.

• Lane: A send-receive pair within a link

• Link Width = \#Lanes in a link
  • x1, x2, x4, x8, x16, x32

• 3 generations so far. 4th will be this year?
  • Called “Gen x” (e.g. PCIe 2.0 = PCIe Gen 2)
## PCIe Bandwidth

<table>
<thead>
<tr>
<th>Link Width</th>
<th>x1</th>
<th>x2</th>
<th>x4</th>
<th>x8</th>
<th>x12</th>
<th>x16</th>
<th>x32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gen1 Bandwidth (GB/s)</td>
<td>0.5</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Gen2 Bandwidth (GB/s)</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Gen3 Bandwidth (GB/s)</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>64</td>
</tr>
</tbody>
</table>

(PCI Express Technology, Mike Jackson and Ravi Budruk)
Topology

• A PCIe link is a point-to-point connection between 2 devices.

• Each device is connected to one PCIe link.

• How can we make a device talk to all other devices in the system?
(PCI Express Technology, Mike Jackson and Ravi Budruk)
PCle: Just a network really

- Implements a network between peripherals and CPU.
Packet Based

• Data has a structure = Packet
• Packet has headers = control
• We’ll elaborate later
PCIe Layers

- **Transaction Layer**: TLP, QoS, Flow Control, Ordering
- **Data Link Layer**: DLLP, Error correction
- **Physical Layer**: Byte encoding, Signaling
Figure 2-14: Detailed Block Diagram of PCI Express Device’s Layers
Encapsulation

Information in core section of TLP comes from Software Layer / Device Core

Bit transmit direction

Created by Transaction Layer

Appended by Data Link Layer

Appended by PHY Layer

(PCI Express Technology, Mike Jackson and Ravi Budruk)
PCIe switches

- A switch is a PCIe component with multiple ports.
- Allows to build a tree topology where every device can talk to every device.
- How does it know to which port to forward each packet? Later on this
Switch as a collection of Bridges
Physical Layer
Differential Signaling

Receiver subtracts D- from D+ value to arrive at differential voltage.

(PCI Express Technology, Mike Jackson and Ravi Budruk)

Why?
Byte Encoding

- PCIe gen1 and gen2: Each byte is encoded with 10 bits (a.k.a 8b/10b encoding)
- PCIe gen3: 128b/130b encoding (practically negligible overhead)
- We’ll skip the reasoning
Data Link Layer
Data Link Layer

- Transmitter: Encapsulates TLPs with Sequence Number and LCRC.
- Receiver: Checks Sequence Number and LCRC
- DLLP (Data Link Layer Packet)
- Small packets. Transaction Layer is unaware of these.
- Ack/Nak, Flow Control, …
Ack/Nak Protocol

- A (hardware) mechanism to guarantee reliable delivery of packets.

- Sender saves packet in replay buffer.

- Receiver checks LCRC and Sequence Number and sends Ack DLLP or Nak DLLP. (with Sequence number of last good TLP).

- Ack = Remove packet from replay buffer.

- Nack = Retry all unacknowledged packets.
(PCI Express Technology, Mike Jackson and Ravi Budruk)
Transaction Layer
Transaction Types

- Read
- Write
- Messages (for interrupts, error reporting)
- Configuration Read/Write
- Others (legacy, we’ll ignore)
Posted and Non-Posted

- Non-Posted transaction: expects a response (completion)
- Posted: Fire and forget
- Read? Write?
Configuration

• Device Discovery

• Device Configuration
Configuration Header

Header Type 0
(used by endpoints)

Header Type 1
(used by bridges)

(PCI Express Technology, Mike Jackson and Ravi Budruk)
Device Discovery

• Configuration software tries to read vendor ID of each device.

• 0xffff == no device

• Bridges are scanned recursively. Starting from 0:0:0 which is the Root Complex.

  • When a bridge is discovered, it is assigned a bus number, and process repeated for its devices.

  • Eventually we’ll have a DFS enumeration of busses.
BDF

(PCI Express Technology, Mike Jackson and Ravi Budruk)
BDF

primary bus: 1
secondary bus: 2
subordinate bus: 4

(PCI Express Technology, Mike Jackson and Ravi Budruk)
Device Configuration Space

- Each device has a 4K configuration space
- All possible devices together 256MB (256x32x8x4K)
- BIOS tells OS where this 256MB range starts.
- Address[27:12] = {bus, dev, func}
- Address[11:2] = offset within configuration space
- Address[1:0] = size, byte enable
BARs

- Different devices need different amounts of memory address space. Or even multiple spaces.
- A device tells what size of memory space it needs via its BARs (Base Address Registers).
- Memory space is reserved by the system and written to the BARs.
- Switches are updated to span the memory spaces of their devices.
MMIO

• We know how to configure the devices and read info about them, but how do we communicate with them for actual work.

• Answer: We map “them” into the memory address space. (Hence the name: Memory Mapped I/O).

• Each device gets a piece of the memory space. Which can be read from / written to by CPU / other devices.

• What are we going to read/write from/to a device?
Use cases

• Meaning of reads/writes to the device’s memory space is defined by the device.

• e.g.: configure a disk drive to use certain encryption.
  • But we have configuration header for that. right? No.
  • But we have configuration space for that. right? Yes. But not always enough.

• e.g: A write to a network card’s memory space at a certain address means: send packet.

• e.g.: GPU exposes part of its global memory to other devices for read and write

• Bottom line: reads/writes to device addresses go to device. It decides what to do with them.
MMIO Example

• One GPU want to write data to another GPU

• Normally, GPU will write to CPU memory (DMA write). CPU will invoke cudaMemcpy(), which asks the other GPU to read from CPU memory (DMA read). Data was moved twice.

• We could instead program the first GPU to write directly to second GPU’s memory. How?

  • Known as Peer-to-Peer access, because it doesn’t involve the CPU.
MMIO example

• GPU exposes its global memory to MMIO via a BAR.

• e.g. Write to (BAR + x) writes to global memory with virtual address x

• So let's tell the first GPU to write to the second GPU's bar.

• Problem: BAR exposes physical addresses. Programs cannot access them directly.

• Solution: Map the BAR (or part of it) to the first GPU's virtual address space.
  • GPU has an internal MMU, which manages the GPU's address space.
  • Managed by the GPU's device driver.

• Similar: Write data received via network card directly to GPU.