PCI-Express Tutorial

Accelerated Systems 046278 - EE Technion

Amir Watad
Layer Responsibilities

- Physical - Signals etc..
- Data Link - Between 2 sides of a link
- Transaction - Between 2 endpoints
Transaction Types

- Configuration read/write
- Memory (mapped) read/write
- Memory (mapped) atomics
  - Spec defines: Fetch & Add, Swap, CAS as optional.
  - Needs support in both Device and Root Complex.
  - Not very common.
    - (supported in e.g. Ivy Bridge, Mellanox ConnectX-5)
Switches and Bridges

==

Bus N
Bus N+1
Bus N+2
Bus N+3
Bus N+4
Switching methods

- By BDF
- By Memory Address
Memory Mapping

- CPU
- Root Complex
- GPU
  - MMIO
- NIC
  - MMIO
- DRAM

Physical Address Space
Memory Mapping - Example

Write: 0xa200

Physical Address Space

- range: [0xa000..0xc000)
- range: [0xc000..0xc100)

0xe000
Memory Mapping - Example

```
<table>
<thead>
<tr>
<th>Physical Address Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>range: [0xa000..0xc000)</td>
</tr>
<tr>
<td>range: [0xc000..0xc100)</td>
</tr>
</tbody>
</table>

Where to send?
```

- GPU MMIO
- NIC MMIO
- Root Complex
- CPU
- DRAM

Write: 0xa200
Memory Mapping - Example

where to send?

Write: 0xa200

Root Complex

CPU

GW

GPU

NIC

Bridge

Bridge Configuration Header

... Memory Base Memory Limit

0xe000

range: [0xc000..0xc100)

Physical Address Space

range: [0xa000..0xc000)
Memory Mapping - Example

where to send?

Write: 0xa200

Root Complex

CPU

Bridge Configuration Header

... Base = 0x000 Limit = 0xffff

*ignoring alignment requirements for simplicity. in reality must be aligned to 1MB

Physical Address Space

range: [0xa000..0xc000)

range: [0xc000..0xc100)

0xe000

CPU

GPU

NIC

Root Complex

DRAM

Bridge

MMIO

MMIO
Memory Mapping - Example

Ignoring alignment requirements for simplicity.

in reality must be aligned to 1MB

Physical Address Space

range: [0xa000..0xc000)
range: [0xc000..0xc100)

Write: 0xa200

where to send?

Bridge Configuration Header

Base = ???

Limit = ???

Ignoring alignment requirements for simplicity. in reality must be aligned to 1MB
Memory Mapping - Example

where to send?

- **CPU**
- **Root Complex**
- **GPU**
- **NIC**
- **DRAM**
- **Bridge**
- **Physical Address Space**

**Write: 0xa200**

**Bridge Configuration Header**
- Base = 0xa000
- Limit = 0xffffffff

*ignoring alignment requirements for simplicity. in reality must be aligned to 1MB

**range:** [0xa000..0xc000)
**range:** [0xc000..0xc100)

0xe000
Memory Mapping - Example

*ignoring alignment requirements for simplicity. in reality must be aligned to 1MB.*
Memory Mapping - Example

*ignoring alignment requirements for simplicity.
in reality must be aligned to 1MB

range: [0xa000..0xc000)
range: [0xc000..0xc100)

Write: 0xa200

where to send?

Bridge

Bridge Configuration Header

... Base = 0xc000

Limit = 0xc100

*ignoring alignment requirements for simplicity. in reality must be aligned to 1MB
Memory Mapping - Example

- CPU
- Root Complex
- GPU
- NIC
- MMIO
- DRAM

Physical Address Space:
- range: [0xa000..0xc000)
- range: [0xc000..0xc100)

Write: 0xa200

what to do with this?
Memory Mapping - Example

Aha! I have a write to offset 0x200 of my MMIO. But what to do with that?

MMIO region vs. BAR
What does the device do

7.2.2 UAR Page Format

Table 13 shows the layout of UAR page.

Table 13 - UAR Page Format

| Offset | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|        | cmsd |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | ci   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | cn   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| eqn (update CI and Arm) | Consumer Index |
| eqn (update CI) | Consumer Index |

DB_BlueFlame_Buffer0_even

DB_BlueFlame_Buffer0_odd

DB_BlueFlame_Buffer1_even

DB_BlueFlame_Buffer1_odd

Table 14 - CQ DoorBell Register Field Descriptions

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>cmsdn</td>
<td>Command Sequence Number. This value should be 0 for the first DoorBell rung, and should be increment on each first DoorBell rung after a completion event. That is, cmsdn = (num_of_completion_event_delivered + 1) / 4.</td>
</tr>
<tr>
<td>1</td>
<td>cmd</td>
<td>0 - Request notification for next Solicited or Unsolicited completion event. eq_c field should specify the CQ Consumer Counter at the time of the DoorBell ring. 1 - Request notification for next Solicited completion event. eq_c field should specify the CQ Consumer Counter at the time of the DoorBell ring.</td>
</tr>
</tbody>
</table>
Peer-to-Peer MMIO Example

Write: 0xc010

Physical Address Space
- range: [0xa000..0xc000)
- range: [0xc000..0xc100)
Peer-to-Peer MMIO Example

- **CPU**
- **Root Complex**
- **GPU**
- **NIC**
- **DRAM**
- **Bridge**
  - Bridge Configuration Header
    - Base = 0xa000
    - Limit = 0xc000

**Physical Address Space**
- range: [0xa000..0xc000)
- range: [0xc000..0xc100)

Write: 0xc010
Peer-to-Peer MMIO Example

- GPU
- NIC
- Root Complex
- CPU
- DRAM
- MMIO

- Bridge Configuration Header
  - Base = 0xa000
  - Limit = 0xfff

- Physical Address Space
  - range: [0xa000..0xc000)
  - range: [0xc000..0xc100)

- Write: 0xa200
Peer-to-Peer MMIO Example

we were in this position before
Peer-to-Peer MMIO Example

Physical Address Space

- range: [0xa000..0xc000)
- range: [0xc000..0xc100)
- Write: 0xc010
- 0xe000
GPU-to-GPU: example

- GPU A wants to write to global memory of GPU B
- GPU B exposes its global memory via its BAR.
- It's just a memory write.. Simple:
  - in CUDA: *(gpu_b_bar + offset) = data;
  - unfortunately, we cannot write directly to physical addresses!
GPU-to-GPU: example

- Code in GPU A asks OS (device driver) to map GPU B’s BAR to GPU A’s virtual memory.
- OS configures GPU A’s MMU
GPU-to-GPU: example

bar_va = driver_map_gpu_bar(GPU_B);
/* bar_va is now 0x42000 */
*(bar_va + offset) = data;
DMA

• Direct Memory Access

• == Device can access memory (read/write) without CPU’s involvement.

• But not just memory. Also MMIO.

• We’ll call a device which performs DMA: DMA Initiator.

• And device which is the target: DMA Target.
Initiator-Target Example

• We want to move data from GPU A to GPU B using DMA.

  • Assuming both GPUs expose their global memory via MMIO

• Who should be initiator and who should be target?
Initiator-Target Example

- We want to move data from GPU A to GPU B using DMA.
- Who should be initiator and who should be target?
- Option 1: GPU A initiates DMA write from its memory to GPU B’s MMIO. (GPU A is initiator)
- Option 2: GPU B initiates DMA read from GPU A’s MMIO to its own memory. (GPU B is initiator)
Initiator-Target Example

• We want to send data from GPU’s global memory to the network via NIC.

• Who should be initiator and who should be target?
Initiator-Target Example

• We want to send data from GPU’s global memory to the network via NIC.

• Who should be initiator and who should be target?

• Option 1: NIC reads global memory via GPU’s MMIO and sends to network (NIC is initiator)

• Option 2: GPU writes from its memory to NICs MMIO. (GPU is initiator)

• Usually not possible. NIC doesn’t expose such an interface. NIC must be initiator, not target!
Virtualization

- Windows
  - Virtual Machine (guest)
- Linux
  - Virtual Machine (guest)
- Linux
  - Virtual Machine (guest)

Hypervisor (host)

Hardware
- NIC
- GPU
- CPU
- DRAM
Virtualization

How to multiplex resource

- Windows
  - Virtual Machine (guest)
- Linux
  - Virtual Machine (guest)
- Linux
  - Virtual Machine (guest)

Hypervisor (host)
  - hypervisor code

Hardware
- GPU
- NIC
- CPU
- DRAM
Virtualization

How to multiplex resource

- Windows
  - Virtual Machine (guest)
- Linux
  - Virtual Machine (guest)
- Linux
  - Virtual Machine (guest)

Hypervisor (host)

Hypervisor code

Hardware

- GPU
- NIC
- CPU
- DRAM

Slow!
Virtualization
How to multiplex resource

Windows
Virtual Machine (guest)

Linux
Virtual Machine (guest)

Linux
Virtual Machine (guest)

Hypervisor (host)

Hardware

vNIC vNIC vNIC

GPU NIC CPU DRAM

SR-IOV
SR-IOV

- PCIe device presents itself as a number of devices. (Virtual Functions in PCIe language)
- #Virtual functions and their attributes configured by hypervisor via their parent Physical Function
SR-IOV: Problem

- Windows
- Linux (guest)
- Linux (guest)
- Cloud Provider: e.g. AWS

Customer

Hypervisor (host)

Hardware

vNIC

GPU

NIC

CPU

DRAM

Cloud

vNIC

vNIC

vNIC
SR-IOV: Problem

Windows
Virtual Machine (guest)

Linux
Virtual Machine (guest)

Linux
Virtual Machine (guest)

Hypervisor (host)

Hardware
vNIC vNIC vNIC

GPU
NIC
CPU
DRAM $
Solution: IOMMU

Cloud Provider: e.g. AWS
IOMMU

• Similar to CPU’s MMU, just for devices.

• Prevents a device from accessing physical addresses (including MMIO) which it is not allowed to.

• Configured by the hypervisor.
Addressing By BDF

• So far we used memory addressing.

• But this can be used only after the devices were discovered and configured (e.g. BARs, Base, Limit)

• In order to discover and configure devices, we need a more “primitive” addressing.

• BDF = \{Bus, Device, Function\}
BDF

Root Complex
  Bus 0
  Dev 0
  Dev 0
  Bus 1
  Dev 0
  Dev 1
  Bus 2
  Dev 0
  Dev 1
  Bus 3
  Bus 4
  Dev 0
  Dev 0
  Bus 5
  Bus 6
  Dev 0
  Dev 0
  Dev 1
  Dev 2
  Bus 7
  Bus 8
  Dev 0
  Dev 0
  Dev 0
  Bus 9
  EP
  EP
  F 0
  EP
  F 7

Dev 0
Dev 0
Dev 0
Dev 0
Dev 0
BDF

Root Complex

Bus 0
Dev 0

Bus 0
Dev 0

Bus 5
Dev 0

Bus 1
Dev 0

Bus 1
Dev 0

Bus 2
Dev 0

Bus 2
Dev 0

Bus 2
Dev 0

Bus 3
Dev 0

Bus 3
Dev 0

Bus 4
Dev 0

Bus 4
Dev 0

Bus 6
Dev 0

Bus 6
Dev 0

Bus 7
Dev 0

Bus 7
Dev 0

Bus 8
Dev 0

Bus 8
Dev 0

Bus 9
Dev 0

Bus 9
Dev 0

BDF = 3:0:0

BDF = 6:2:0

BDF = 9:0:7
Configuration Space

• Show Example: Address —> BDF and dword

• E.g read vendor ID of device 5:0:4

• Explain difference between header and rest of the space.
  • header is same for all PCIe devices. Accessed by generic OS code.
  • rest is device specific, accessed by device driver.
Configuration Space

- CPU
- Root Complex
- GPU
- MMIO
- NIC
- MMIO
- DRAM

Physical Address Space

- config space
Configuration Space

- 4K of Physical Address space per PCIe device.
- Total 256MB. (256 Buses x 32 Devices x 8 Functions x 4K)
- Consists of
  - Configuration Header: Generic for all PCIe devices. Handled by generic OS code.
  - Rest of space: Device specific. Handled by device driver.
Configuration Space - Example

• Let’s read the vendor ID of device 9:0:0.

• Configuration Space base address (setup by BIOS, mapped to VA by OS): 0xef00000000
Configuration Space - Example

- Let’s read BAR 0 of device 9:0:0.
- Configuration Space base address (setup by BIOS, mapped to VA by OS): 0xef00000000
- BAR 0: Bytes 16-19 of header
Configuration Space - Example

• Let's read BAR 0 of device 9:0:2.

• Configuration Space base address (setup by BIOS, mapped to VA by OS): 0xef0000000

• BAR 0: Bytes 16-19 of header

• We have 32x8 = 256 functions per bus
  • = 256 x 4KB = 1MB of configuration space per bus.

• Bus 9 starts at: 9MB.

• Function 2 → 9MB + 4K x 2 = 9MB + 8K

• Required configuration header is at: 0xef0000000 + 9MB + 8K

• Required register (BAR 0) is at: 0xef0000000 + 9MB + 8K + 16 = 0xEF0902010