An Introduction to PCIe

17TH FEB, 2020

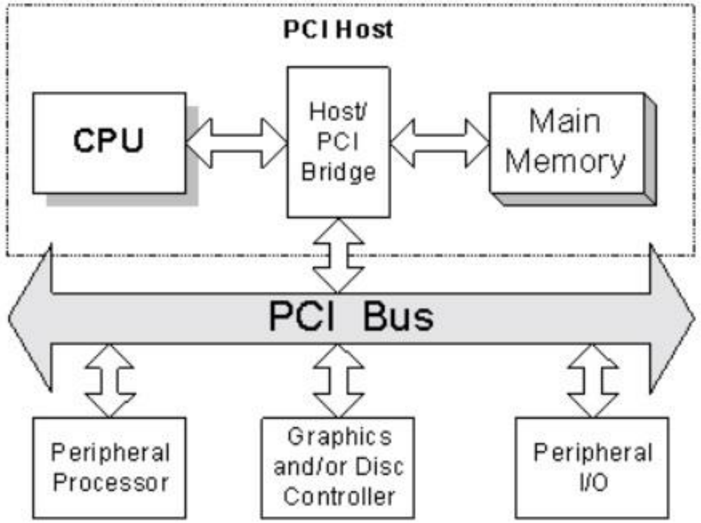
SAHIL SEMICONDUCTOR

PCIe (Peripheral Component Interconnect Express)

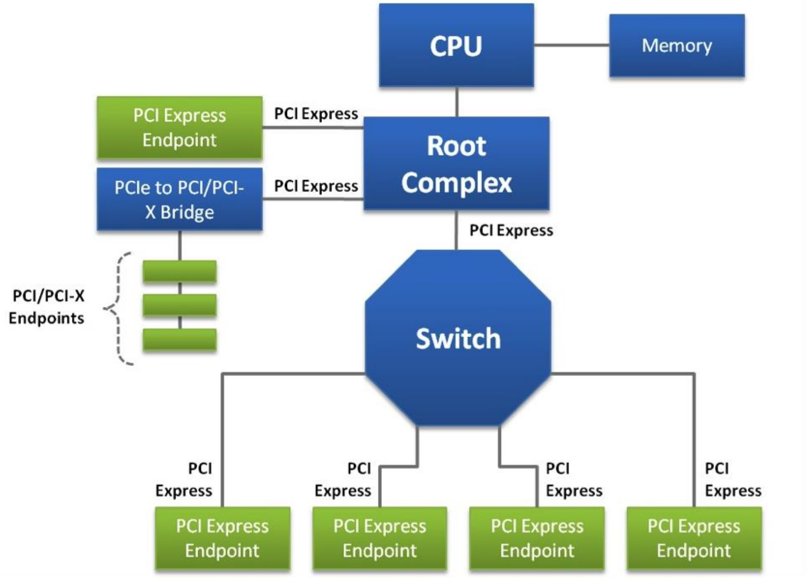
- High speed serial bus designed to replace older PCI and PCI-X standards.
- It provides common interface to various devices including hard drives, SSDs, Ethernet, USBs etc.
- PCIe is a **serial** bus while older PCI and PCI-X buses were **parallel**.

PCIe vs PCI

- PCI uses a shared parallel bus architecture in which the PCI host and all devices share a common set of address, data and control lines.
- In contrast, PCI Express is based on point-to-point topology with separate serial links connecting every device to the host.



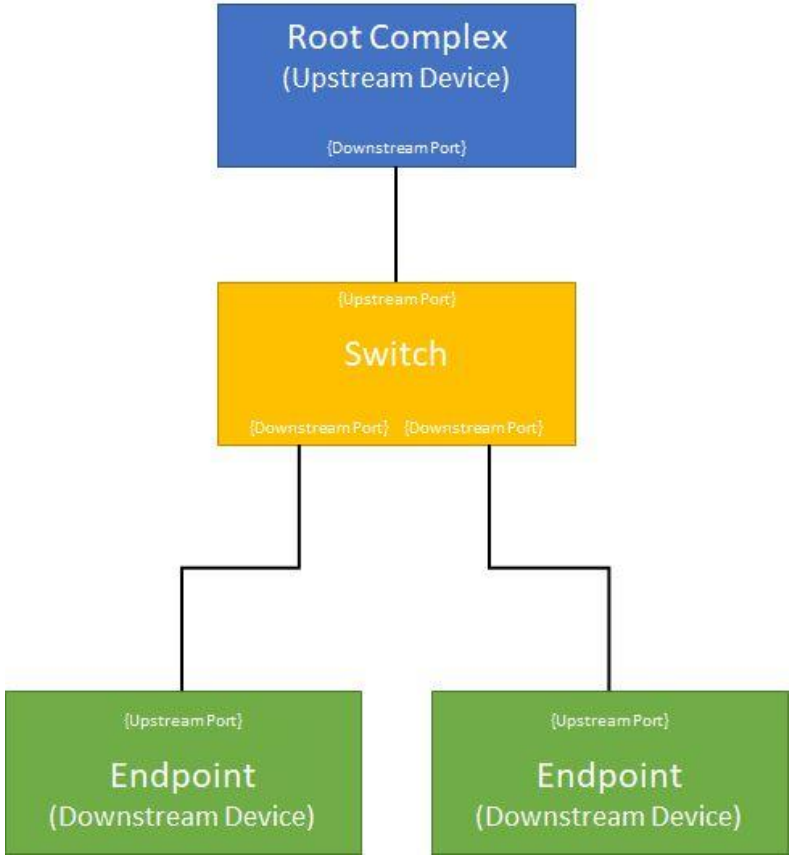
PCI



PCI Express

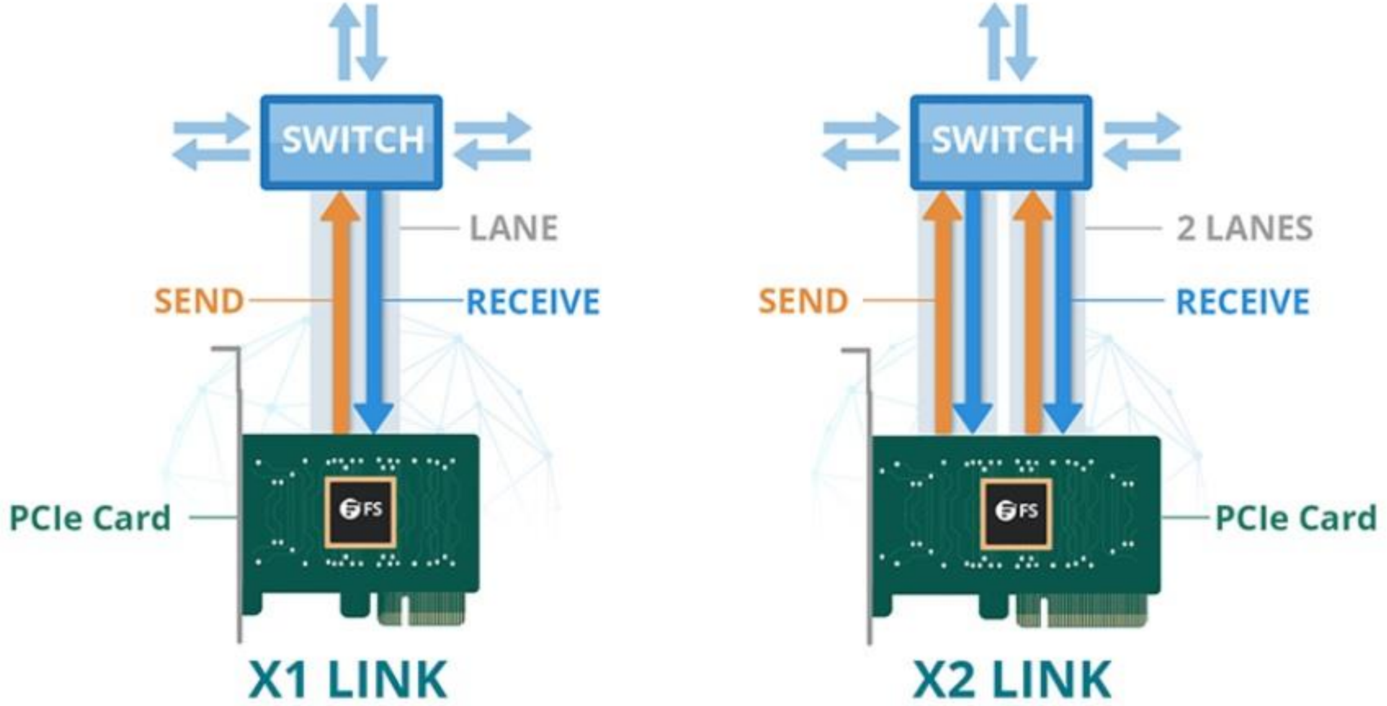
Types of PCIe Devices

- Root Complex:
  - Device which connects the processor and memory subsystem to the PCI Express switch fabric composed of one or more switch devices.
  - Offloads CPU from handling PCIe transactions directly.
  - Root complex has only downstream ports (also called Root ports)
- PCIe Switch:
  - PCIe switch is a device which used to connect large number of PCIe devices to a processor system.
  - A switch has one upstream and one or more downstream ports.
  - A packet received from upstream port is routed to a downstream port based on the destination address.
- PCIe End Point (EP):
  - An Endpoint is a device that resides at the bottom of PCIe tree topology.
  - It implements a single Upstream Port toward the Root.
  - Examples are NICs, Graphic cards etc.



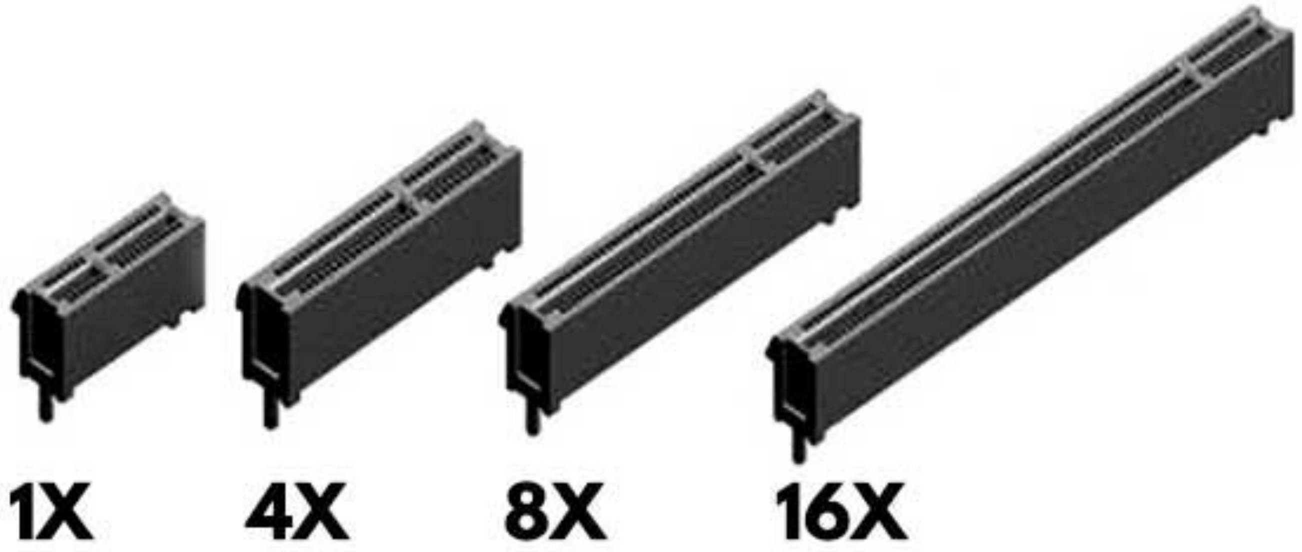
PCIe Lanes

- A PCIe connection consists of one or more data-transmission lanes, connected serially.
- Each lane consists of two pairs of wires, one for transmitting and one for receiving.
- There are 1, 4, 8 or 16 lanes in a single PCIe slot – denoted as x1, x4, x8, or x16.
- Data transmitted on multiple lane link is interleaved. Each successive byte is sent down on successive lanes (Data Striping).



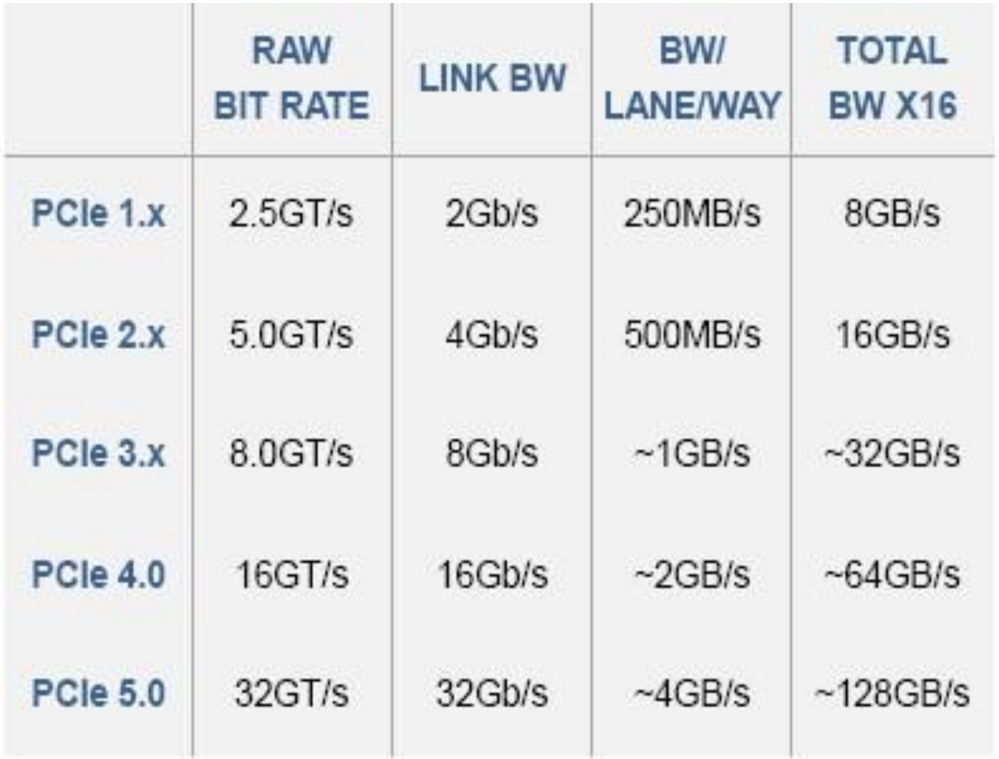
PCIe Slots

- PCIe slots come in different physical configurations: x1, x4, x8, x16, x32.
- The number after the x tells us how many lanes (how data travels to and from the PCIe card) that PCIe slot has.
- A PCIe x1 slot has one lane and can move data at one bit per cycle.
- A PCIe x2 slot has two lanes and can move data at two bits per cycle (and so on).



PCIe Generations and Throughput

- PCIe started with 1.0 and over the period of time, kept on evolving.
- PCIe Gen4 is the latest PCIe generation.
- Each generation doubled the bandwidth of the PCIe bus.
- The table shows theoretical max throughput of different PCIe generations.
- Actual measured throughput will lower be than these numbers. This will be dependent on various factors including:
  - Encoding technique (8b/10b for PCIe Gen1 and Gen2, 128b/130b for PCIe Gen3)
  - Transaction layer packet overhead (PCIe uses packets which have headers which consume PCIe bandwidth)
  - Link protocol overhead



	RAW BIT RATE	LINK BW	BW/ LANE/WAY	TOTAL BW X16
PCIe 1.x	2.5GT/s	2Gb/s	250MB/s	8GB/s
PCIe 2.x	5.0GT/s	4Gb/s	500MB/s	16GB/s
PCIe 3.x	8.0GT/s	8Gb/s	~1GB/s	~32GB/s
PCIe 4.0	16GT/s	16Gb/s	~2GB/s	~64GB/s
PCIe 5.0	32GT/s	32Gb/s	~4GB/s	~128GB/s

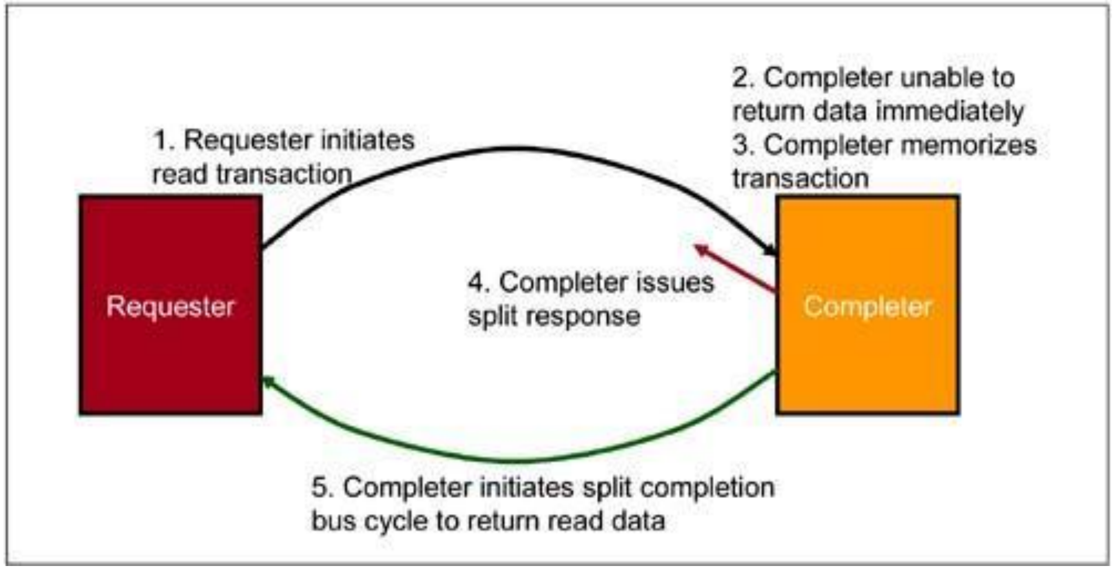
PCIe Transactions

- PCIe uses packets to accomplish data transfer between devices.
- These packets are called Transaction Layer Packets (TLP).
- There are four types of PCIe transactions:

Address Space	Transaction Types	Purpose
Memory	Read, Write	Transfer data to or from a location in the system memory map
IO	Read, Write	Transfer data to or from a location in the system IO map
Configuration	Read, Write	Transfer data to or from a location in the configuration space of a PCI-compatible device.
Message	Vendor Specific	General in-band messaging and event reporting (without consuming memory or IO address resources). Eliminate the need for an extra side band channel. Examples are interrupts, error signaling.

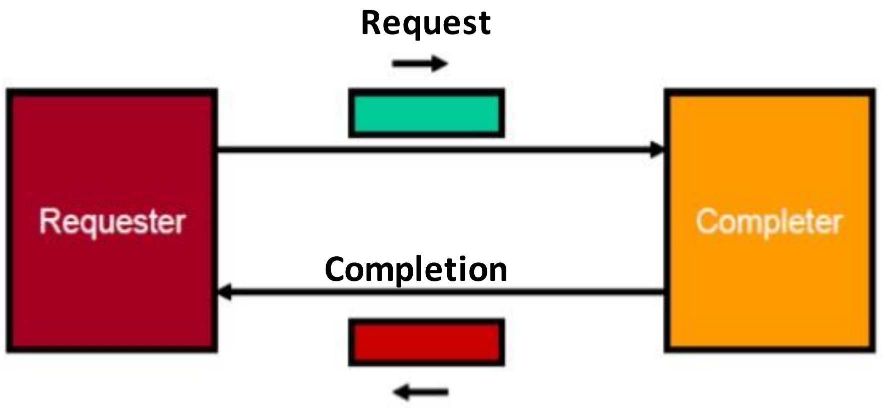
PCIe Transactions

- All PCIe transactions are split-transactions
  - Request and completion are independent of each other.
  - A requester does not need to wait for the completion.
  - For example, a read request is sent to an End Point device. The completion will come later with the requested data.

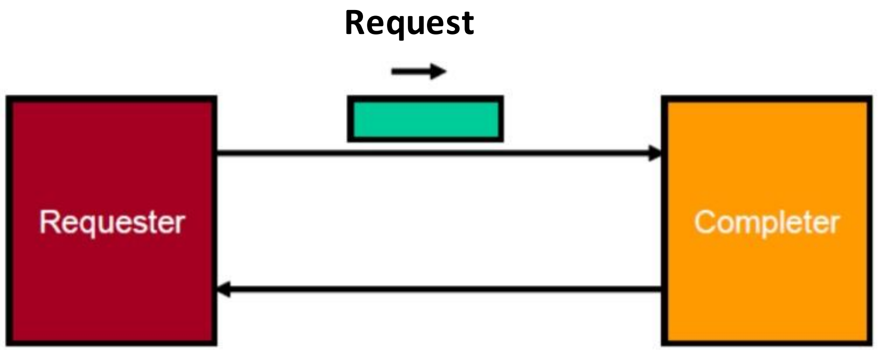

1. Requester initiates read transaction
2. Completer unable to return data immediately
3. Completer memorizes transaction
Requester
4. Completer issues split response
Completer
5. Completer initiates split completion bus cycle to return read data

Posted vs Non-posted Transactions

- There are two types of transactions based on completion:
  - Non-posted transactions: Transactions which require a completion – for example, a read request.
  - Posted transactions: Transactions which do not require a completion. For example, a write to a memory location.
- Posted transactions improve the system performance significantly – sender of the request does not need to wait for the completion of the transaction.

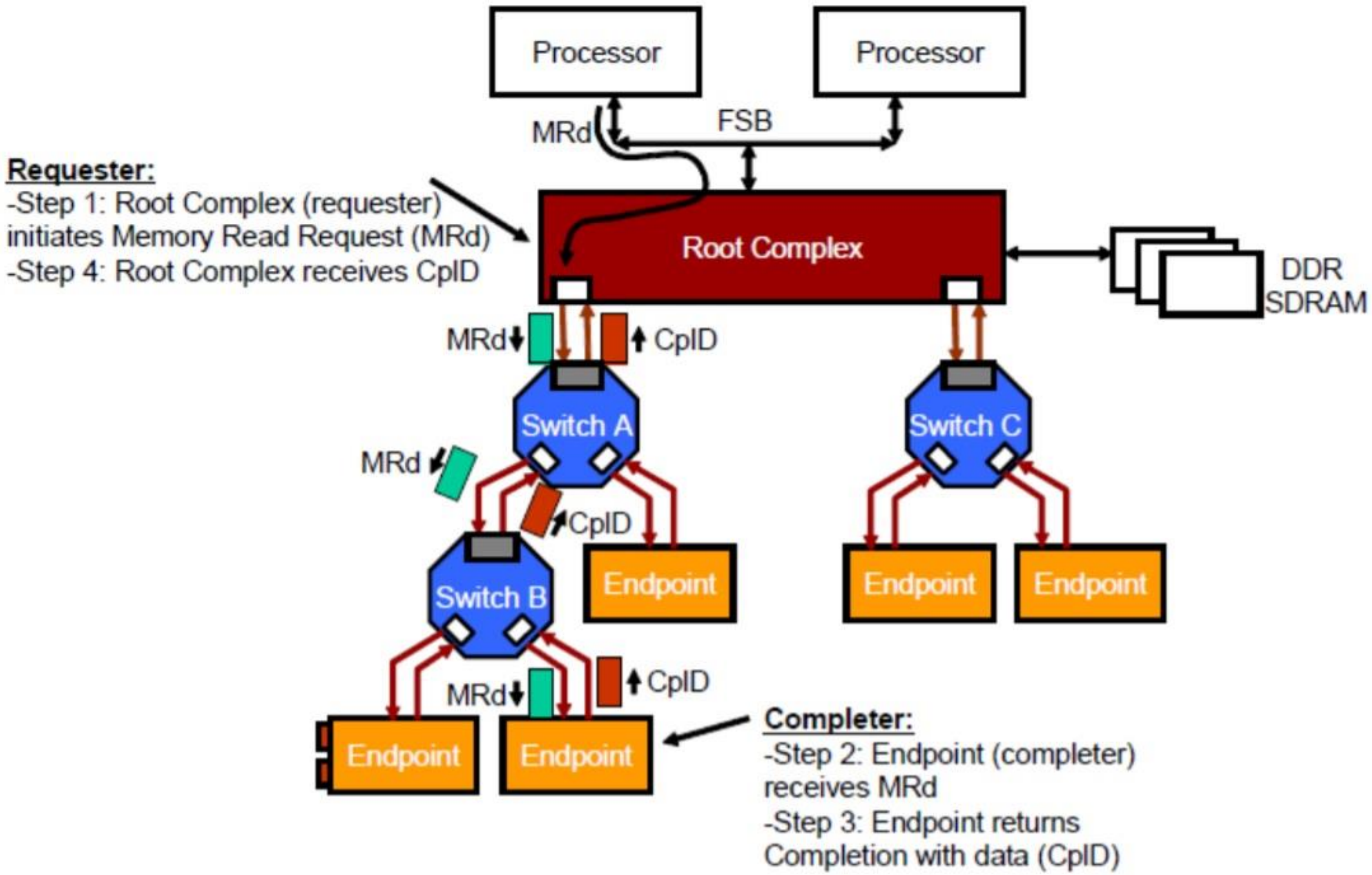


Non-posted Transaction



Posted Transaction

Simple PCIe Transaction (CPU memory read targeting an EP)



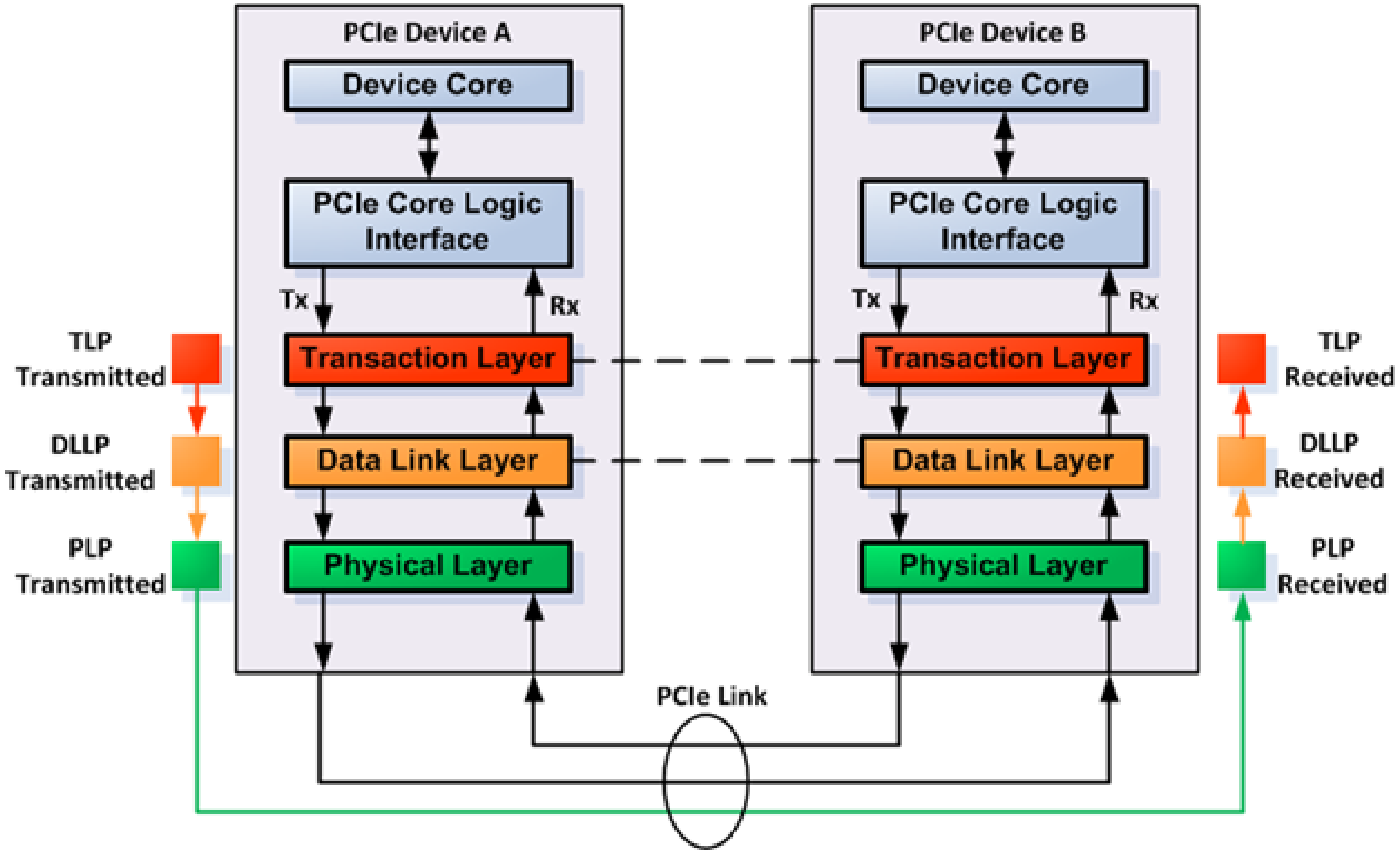
Requester:
-Step 1: Root Complex (requester) initiates Memory Read Request (MRd)
-Step 4: Root Complex receives CplD

Completer:
-Step 2: Endpoint (completer) receives MRd
-Step 3: Endpoint returns Completion with data (CplD)

PCIe Device Layers

- PCIe uses 3 layered protocol
- Each layer inserts its own header

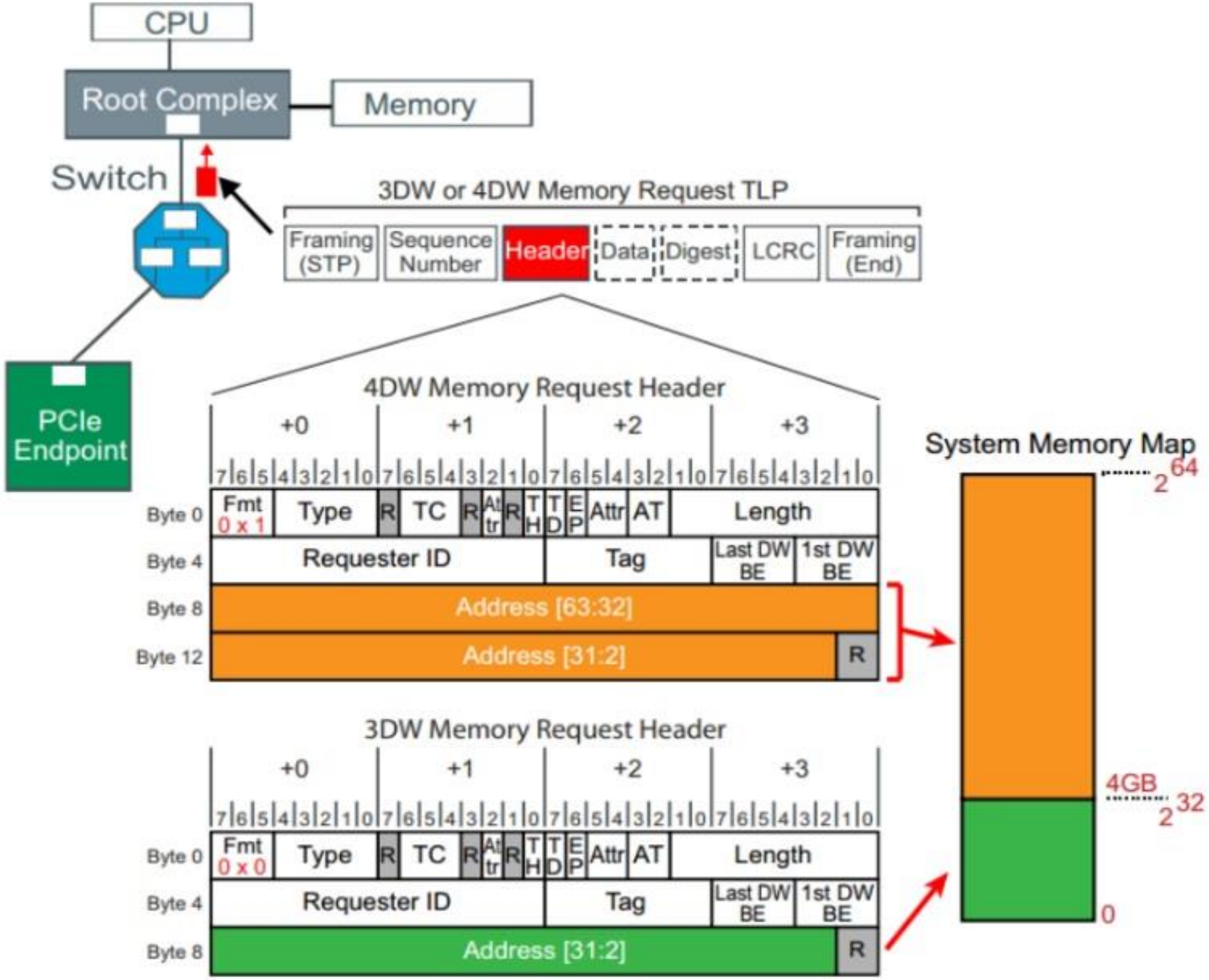
1. Transaction Layer:
   - It turns user data or completion data into PCIe transactions (TLPs)
2. Data Link Layer:
   - Responsible for reliable transport of TLPs
3. Physical:
   - Contains necessary digital and analog circuits



TLP Format

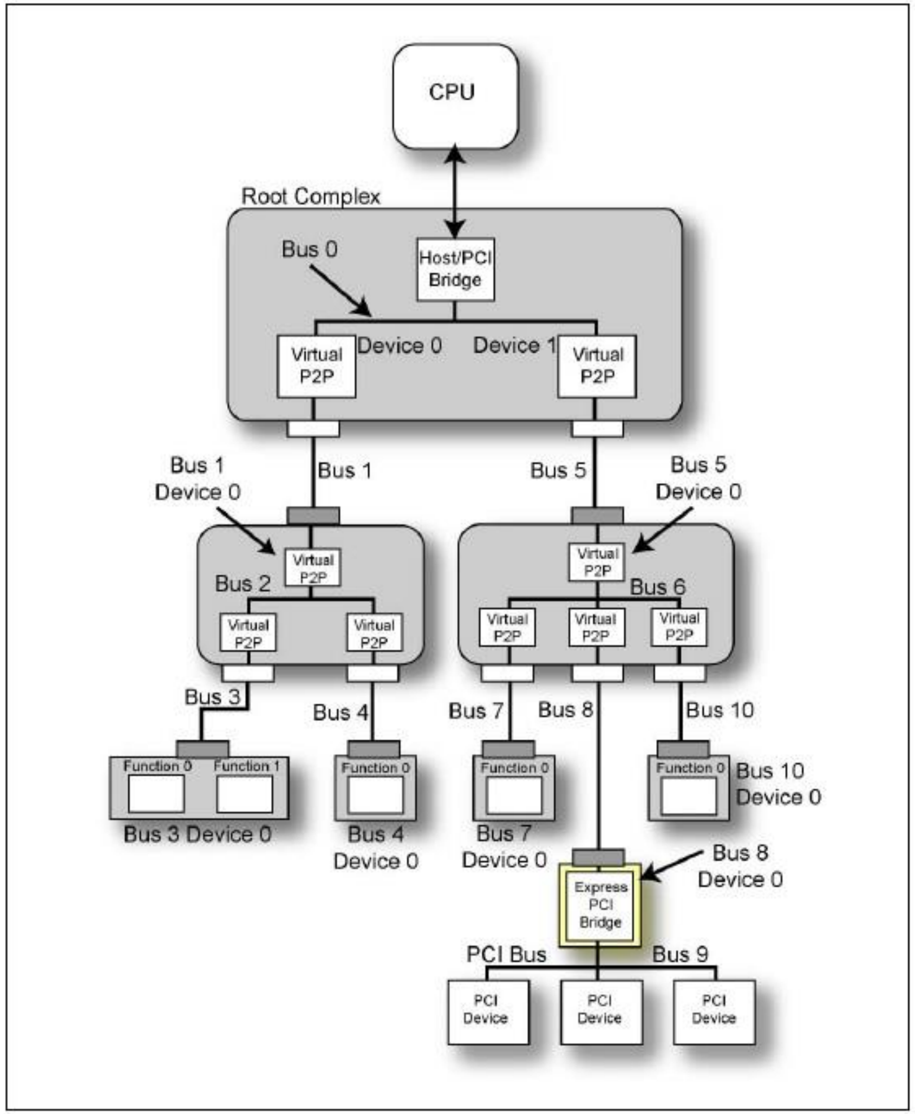
- TLP uses addresses for routing:
  - Mapped memory addresses (32-bit or 64-bit)
  - IO addresses (Bus/Device/Function)

- Each device support a max size payload size in TLP – it is called MPS (Maximum Payload Size)
- Typical MPS values could be from 128 to 4096 bytes.
- Higher MPS will give better performance (more payload, less TLP header overhead).



Bus, Device and Functions

- A device resides on a bus and contains one or more functions.
- Each function within a single device provides a standalone functionality. For example, one function could be a graphic controller while the other may be a network interface.
- There are:
  - 256 buses
  - 32 devices/bus
  - 8 functions/device



PCIe Address Spaces

- PCIe implements four address spaces:

1. PCIe Configuration Space (up to 4KBytes):
    - Required/standard. Defined in the specifications. Every PCIe device has its configuration space mapped to memory.
    - Also provides the first 256 bytes of compatible PCI (memory-mapped and via port IO for backwards compatibility).
2. PCIe Memory-mapped space:
    - Optional. Dependent on whether the device manufacturer needs to map system memory to the PCI device
3. PCIe I/O-mapped space:
    - Optional. Same as PCI Memory Space
4. PCIe Message Space:
    - For low-level protocol messaging/interrupts.

TLP Routing

- Mainly three types of TLP routings are used by PCIe:

1. Address Routing:
   - Address routing is used to transfer data to and from memory, memory mapped IO and IO locations.
   - Addresses could be 32-bit or 64-bit depending on the system configuration.
   - Root Complex and switches have memory base + limit for address switching.
2. ID Routing (Also known as BDF routing):
   - ID routing is based on the logical positioning of the device (Bus number, Device number, Function number)
   - Address field in TLP now uses Bus-Device-Function.
3. Implicit Routing:
   - Used for broadcasts from Root Complex and messages which are going towards Root Complex.
   - It takes the advantage of the fact that switches have only one upstream port.

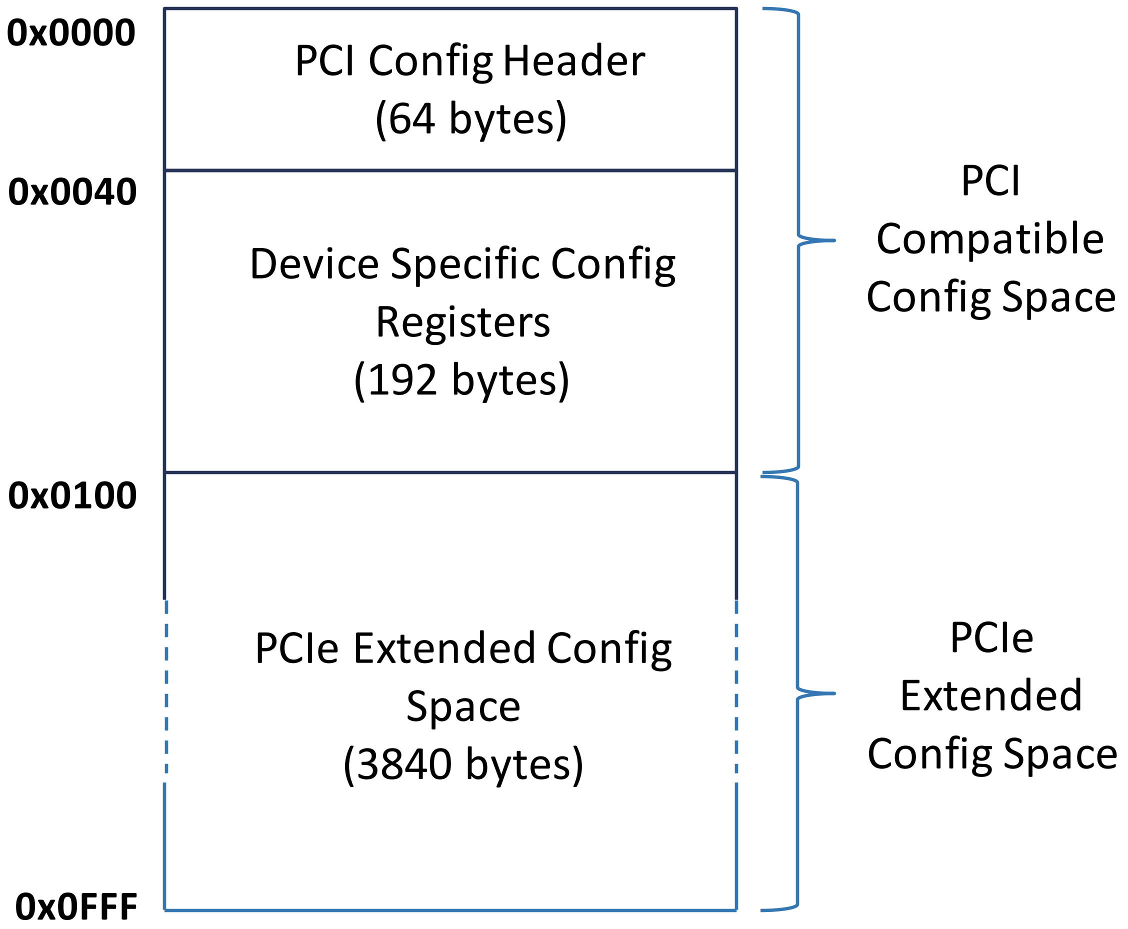
PCIe Configuration Space

- PCI devices have a set of registers referred as Configuration Space
- These registers define the type and nature of the device and how it should be used.
- As part of the PCIe discovery, the OS scans through all the PCIe devices during the boot time and allocates required resources for the device to operate properly.

- Configuration space registers are mapped to the main memory.
  - PCI → Configurations space of each device is 256 bytes
  - PCIe → PCIe has introduced extended configuration space. The total size for each device is 4k.

- Total size of PCIe config space: 4k x 256 buses x 32 devices/bus x 8 functions/device
  - **256MB**

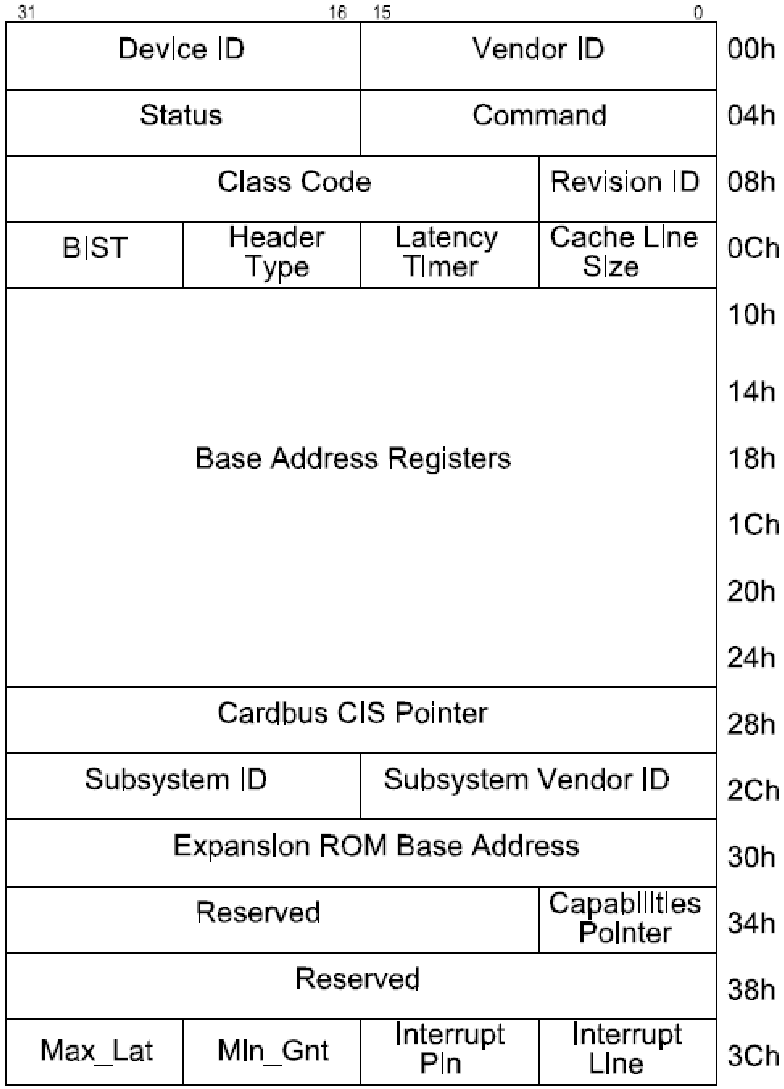
PCIe Configuration Space



- PCI compatible configuration space is 256 bytes
- Every PCI device implements this space
- First 64 bytes are header
- The remaining 192 bytes are device dependent region

- PCI Extended Config Space is 3840 bytes
- Defines extended capabilities of a device

PCI Config Header

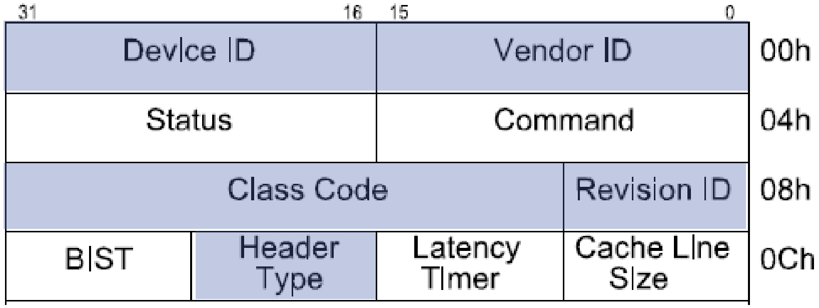


31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency Timer	Cache Line Size	0Ch
Base Address Registers				10h
				14h
				18h
				1Ch
				20h
				24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Capabilities Pointer	34h
Reserved				38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch

- Implemented in PCIe too
- Three header types (0-2)
  1. Type 0 = General Device
  2. Type 1 = PCI-to-PCI Bridge
  3. Type 2 = CardBus Bridge
- Shown is Type 0

- Divided into 2 parts:
  - First 16 bytes (0-F) are standard and defined the same for all devices
  - The remaining header bytes are optional per the vendor, depending on what function the device performs

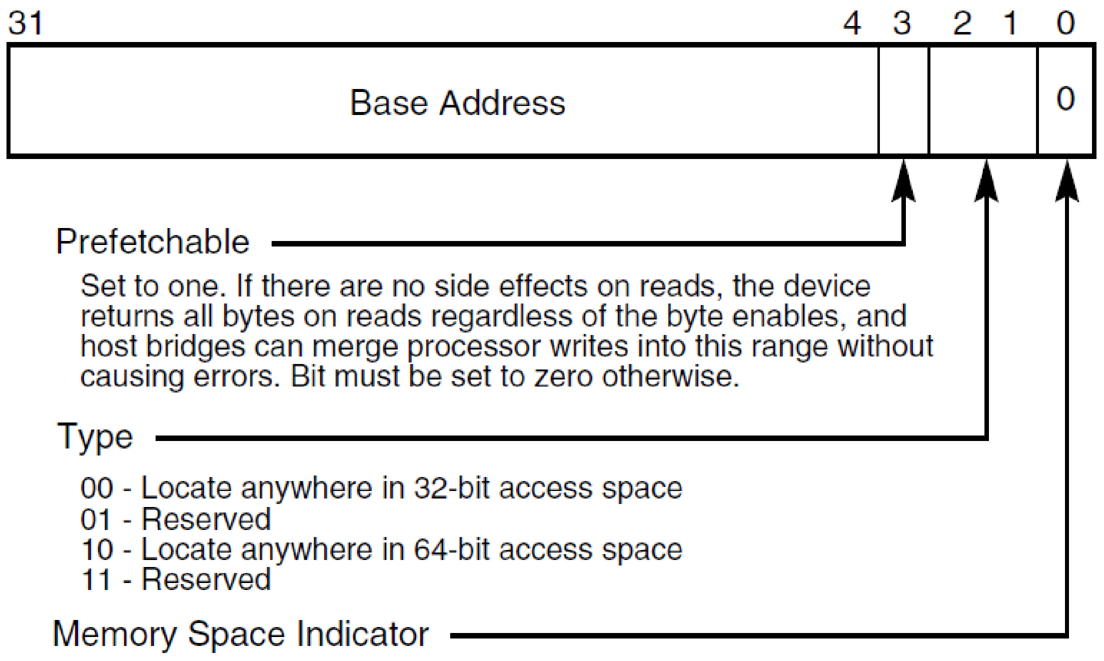
PCI Device Identification



31	16	15	0	
Devlce ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency TImer	Cache LIne SIze	0Ch

- Five fields (all required) can be used to identify the device and its basic functionality
- Vendor ID identifies the manufacturer of the device
  - Each ID is unique
- Device ID identifies the particular device, set by the vendor
- Revision ID is set by the vendor, viewed as an extension to Device ID (Intel is 8086h, AMD microcontrollers is 1022h)
- Class Code used to identify the generic functionality of the device
- Header Type identifies what type of header to expect (General, PCI bridge, CardBus bridge)
  - Bit 7 being set (0x80) indicates device is a multi-function device

Base Address Registers (Memory Space)

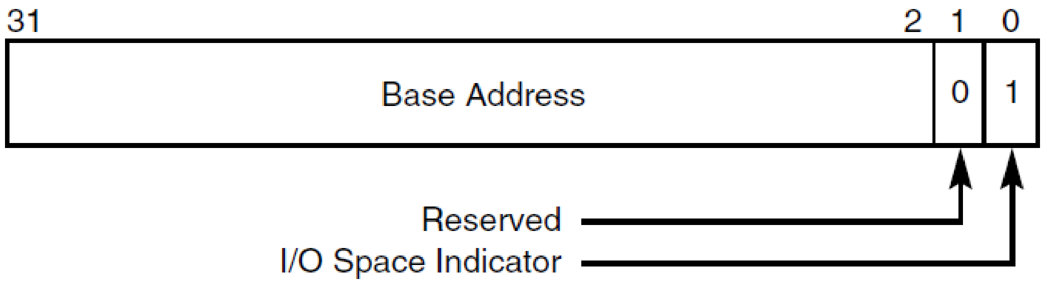


Set to one. If there are no side effects on reads, the device returns all bytes on reads regardless of the byte enables, and host bridges can merge processor writes into this range without causing errors. Bit must be set to zero otherwise.

00 - Locate anywhere in 32-bit access space
01 - Reserved
10 - Locate anywhere in 64-bit access space
11 - Reserved

- Actual Base address is obtained by bitwise ANDing the value in bits 31:4 with FFFF_FFF0h (mask lower 4 bits)
- A cleared Bit 0 indicates this will be located in memory address space, otherwise IO space
- 64-bit accessibility is provided by programming back-to-back BARs:
    - BAR[x] is the upper 32 bits, BAR[x+1] is the lower 32 bits

Base Address Registers (IO Space)



- Base address is obtained by logically ANDing the value in bits 31:4 with FFFF_FFF0h (mask lower 4 bits)
- If Bit 0 is 1, the Base Address will be an offset in the port I/O address space
- PCI SIG recommends that devices are mapped to memory rather than I/O, because I/O can be fragmented

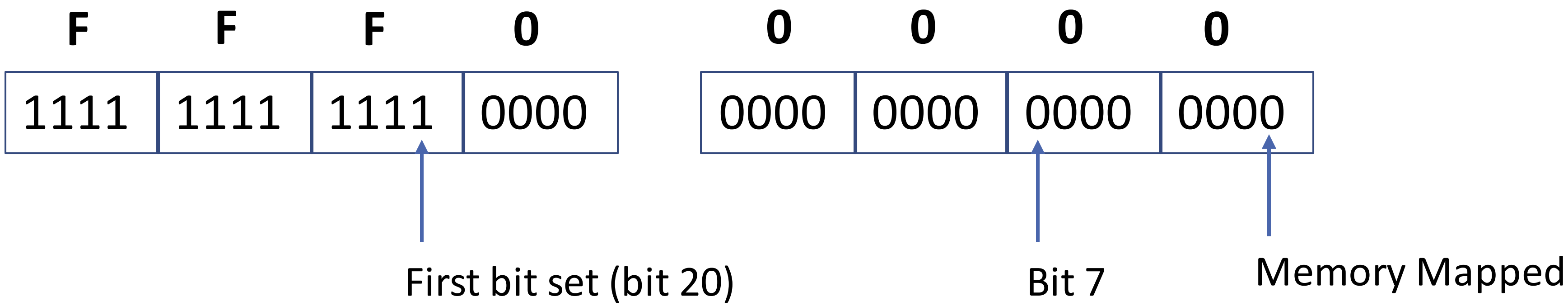
Finding required mapped memory size using BAR

- PCI card manufacturer will specify through each BAR how much memory it wants the Operating System to allocate for the device.
- Once Operating System has allocated the required memory, it will write the start address of the allocated memory block in the corresponding BAR.

- This is how the required memory is calculated:
  1. Store the BAR to a local variable.
  2. Write all 1s to the BAR register.
  3. Read back the contents of the BAR register.
  4. If the read value is non-zero, start scanning bits, starting bit 7, until the first bit set to 1 is found.
  5. The binary weight value of this bit represents the size of the memory required.
  6. If the value read back is zero, it indicates that the BAR is not implemented.

Finding required mapped memory size using BAR

- For example, value read back from a BAR after writing all 1s is 0xFFF0 0000:



F	F	F	0	0	0	0	0
1111	1111	1111	0000	0000	0000	0000	0000

First bit set (bit 20)

Bit 7

Memory Mapped

- Memory required for this device = 2^20 = 1MB

Configuration Space Accesses

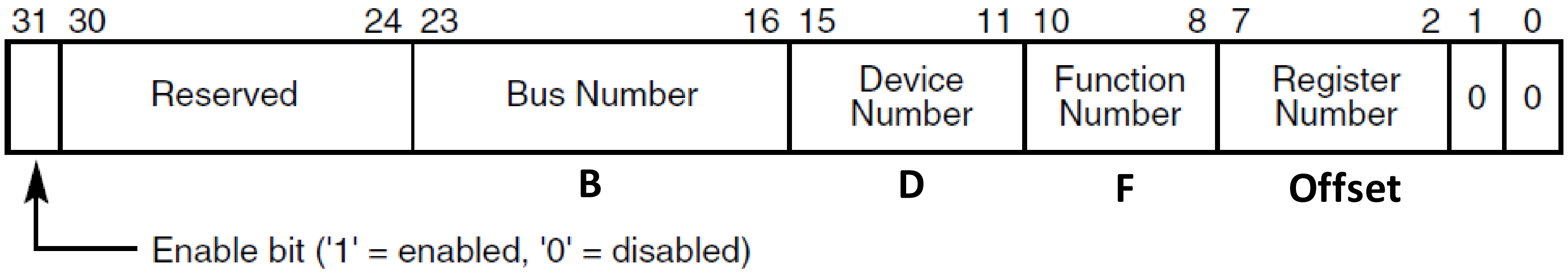
- There are two ways to access PCI Compatible Config Space (0x0000 – 0x00FF):

  1. **Port IO**
     - Used in older systems
     - Due to IO space mapping limitations, it was not possible to map the whole PCIe config space to host memory.
     - Uses two IO ports for reading/writing the config space.
  2. **Memory Mapped IO**
     - Whole PCIe config space is mapped to a region in host memory.
     - Host CPU can read/write this memory directly.
     - Root complex knows the memory address range and initiates a PCIe config read/write transaction (using TLPs) when host CPU reads this memory region.

- There is only one way to access PCIe Extended Config Space (0x0100 – 0x0FFF) which is Memory Mapped IO.

Configuration Space Accesses – Port IO

- Access is performed using two IO ports in Root Complex:
  1. Config Address Port: 0x0CF8
  2. Config Data Port: 0x0CFC
- A 32-bit address of the required config space register is written at Config Address Port:



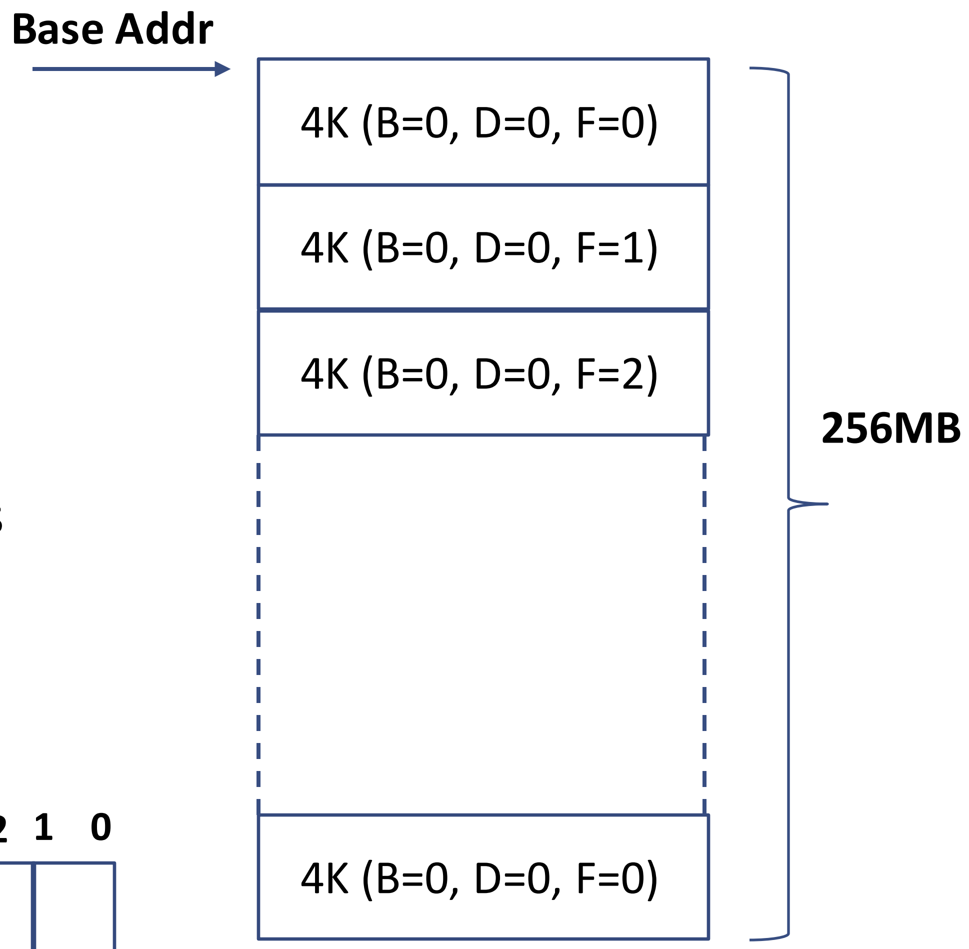
31	30 24	23 16	15 11	10 8	7 2	1	0
	Reserved	Bus Number	Device Number	Function Number	Register Number	0	0
		B	D	F	Offset		

Enable bit ('1' = enabled, '0' = disabled)

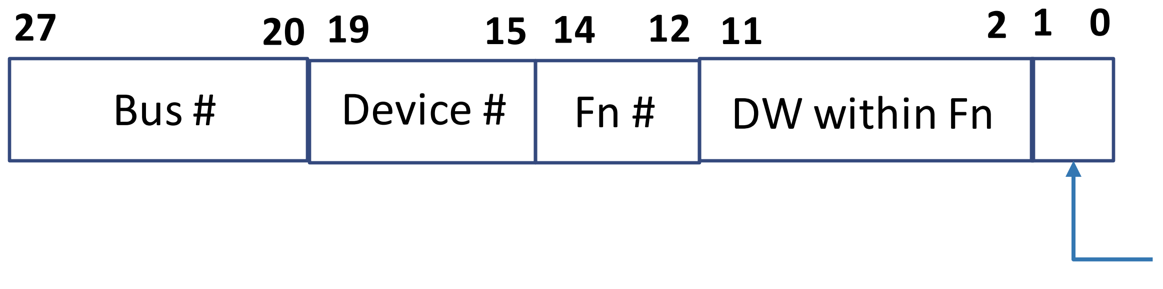
- Bit 0-7 indicates offset within the Config Space (256 bytes)
- Once config address write is done, a corresponding read or write data is performed at Config Data Port.

Configuration Space Accesses – Memory Mapped IO

- PCIe configuration space is mapped to host memory’s address space.
- At system initialization, BIOS determines the memory area base address, allocates memory and communicates this address to both OS and Root Complex.
- Each PCIe configuration register is assigned a unique address (base address + offset).
- Root Complex monitors memory accesses and if it detects an access with in the allocated memory area, it initiates a config PCIe transaction using TLPs.
- Register Config Addr = Base Addr + Offset



Offset =



27 … 20	19 … 15	14 … 12	11 … 2	1 0
Bus #	Device #	Fn #	DW within Fn	

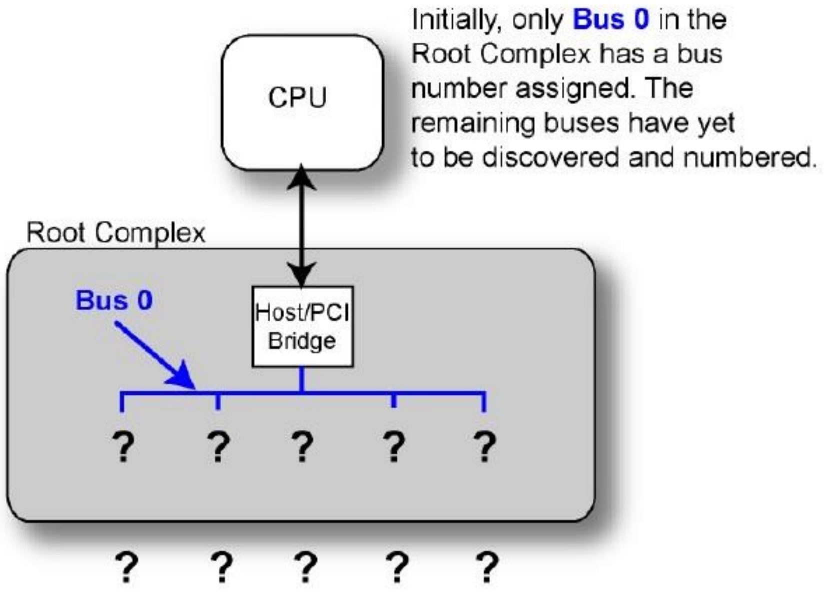
Byte offset within the selected DW

Reading Configuration Space - lspci

- You can use ‘lspci’ in Linux to read PCI configuration space
- Go through lspci man page
- Another lspci guide:
  - https://diego.assencio.com/?index=649b7a71b35fc7ad41e03b6d0e825f07

PCIe Device Enumeration

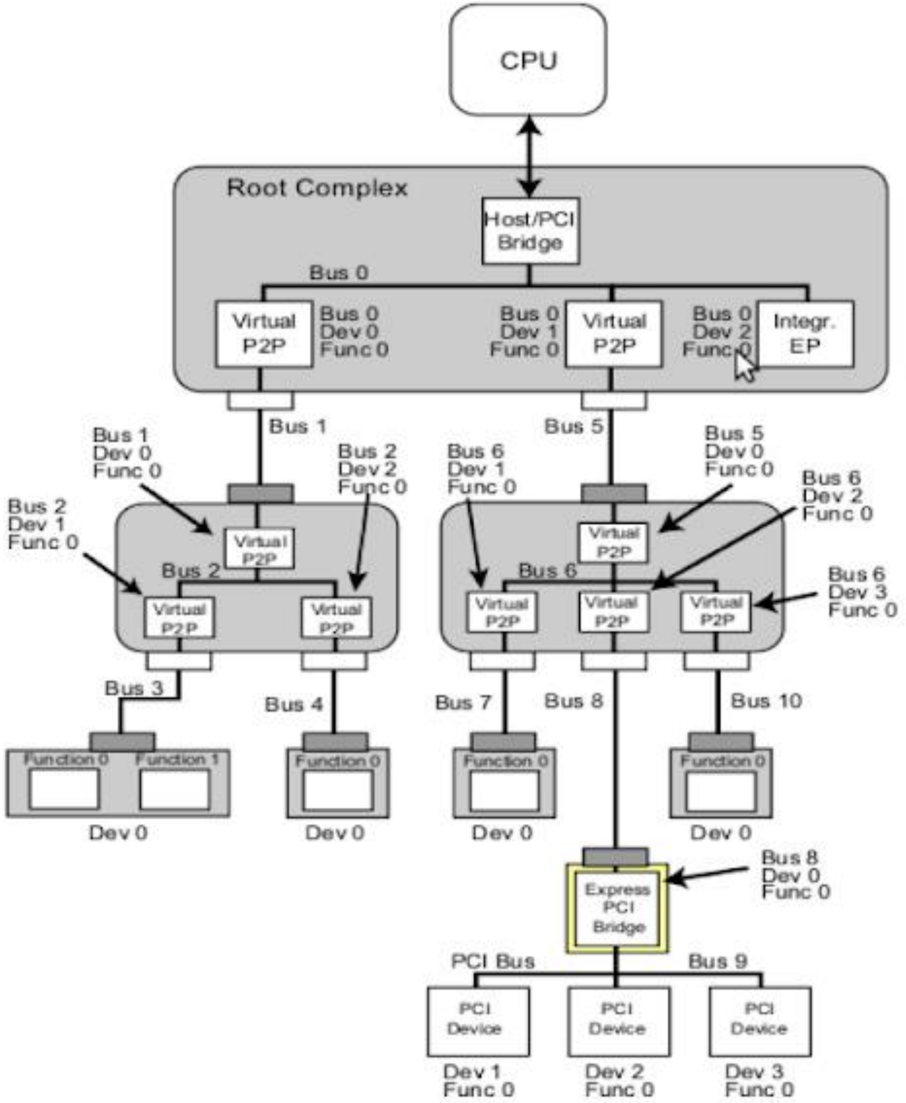
- PCIe device enumeration is a process of detecting devices connected to a host CPU.
- Software reads all expected configuration locations to leans which PCIe functions are present.
- At power up, configuration software only knows about Bus 0 (the Bus which resides on the downstream side of Host/PCI bridge)



Initially, only **Bus 0** in the Root Complex has a bus number assigned. The remaining buses have yet to be discovered and numbered.

PCIe Device Enumeration

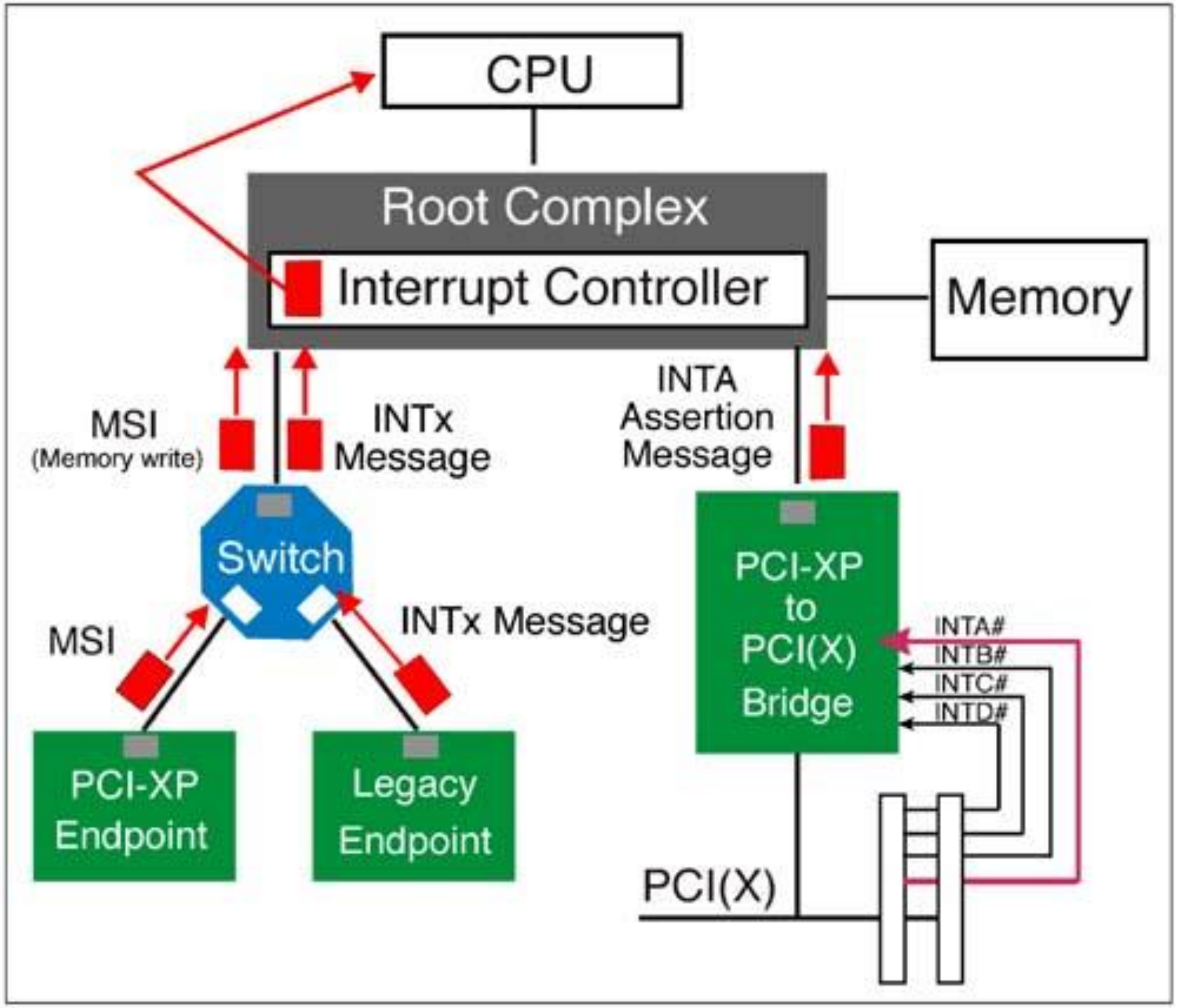
- Device probing starts with reading Vendor ID in PCIe config space of Bus 0, Device 0, Function 0.
- If Vendor ID for function 0 is 0xFFFF, skip the while device and go to the next device.
- If a valid Vendor ID is returned, detect the rest of the functions within this device.
- If the device is a switch, assign bus number accordingly.
- The process is repeated for all buses (256), devices (32) and functions (8).
- Bus, Device and Function numbers are associated to each device found during this process.



PCIe Interrupts

- Three types of interrupts in a PCIs system:
  1. Legacy Interrupts:
     - PCI uses legacy interrupts
     - Four dedicated signal lines (wires) [INTx; also known as INTA, INTB, INTC and INTD]
     - PCI device asserts one of the interrupt lines which indicates an interrupt request to the interrupt controller.
     - PCIe implements these through in-band messages (no dedicated signals)
  2. MSI (Message Signaled Interrupts):
     - Simple memory write transaction – pointed to a well-known host address.
     - Each interrupt vector has a dedicated Interrupt Service Routine.
     - Each device can support up to 32 interrupts.
  3. MSI-X (Message Signaled Interrupt – Extension):
     - Similar to MSI interrupt but increases number of interrupts to 2048 for each device.

PCIe Interrupts



Legacy Interrupts

- All four interrupt signals (INTx; INTA-INTD) are defined as in-band messages
- When a core needs to generate a legacy interrupt, it sends INTA-INTD message upstream which is routed to the system interrupt controller.
- The assert INTx message will result in the assertion of INTx line to the Interrupt controller and the Deassert INTx message will result in the deassertion of the INTx line.

Message Signaled Interrupt (MSI)

- Native PCI Express device use message signaled interrupt (MSI).
- Message Signaled Interrupts (MSIs) are delivered to the Root Complex via memory write transactions.
- This write is distinguished from normal write by target address which is reserved for MSI interrupt delivery.
- A MSI enabled device will interrupt the CPU by writing to a specific address in memory with a payload of 1 DW (double word).
- A memory write with an exclusive host address and data field is commonly referred to as a MSI vector.
- The device may support 1, 2, 4, 8, 16, or 32 interrupt vectors.
- All MSI capable devices implement the MSI capability structure defined in PCIe Extended Config Space.

Advantages of Message Signaled Interrupt

- No IRQ sharing:
  - As number of devices keep increasing, devices generating legacy interrupts have to share IRQ numbers.
  - When an interrupt is raised, host CPU will sequentially call every device driver routine associated with that IRQ. Each service routing will check its device state to find if that device has generated an interrupt.
  - Devices implementing MSI will have a unique address associated to all interrupts which removes the need of interrupt sharing and performance degradation due to device polling overhead.
- Latency Reduction:
  - In legacy interrupt implementation, each interrupt handler its device registers to find out why the device has raised an interrupt. This adds additional overhead to the interrupt service routine.
  - MSI vector allows a device to provide more information when interrupting including the reason.
  - No need to poll the device to find out why interrupt was generated.

Advantages of Message Signaled Interrupt

- Mechanical Reduction:
  - MSI interrupt are sent in-band.
  - No need for additional pins or signals.
- No more race conditions:
  - Typically an interrupt notification when an operation (for example data transfer) is complete.
  - In legacy interrupt mode, an interrupt (INTA) is sent once the final data block transfer is complete.
  - Due to the parallel nature of these two events, it is possible that interrupt becomes visible to the CPU before the actual data transfer completes. This can create a race condition where interrupt handler can read stale data.
  - MSI prevents this due to in-band nature of interrupts. Data transfer and interrupt transactions become serialized.

Useful links

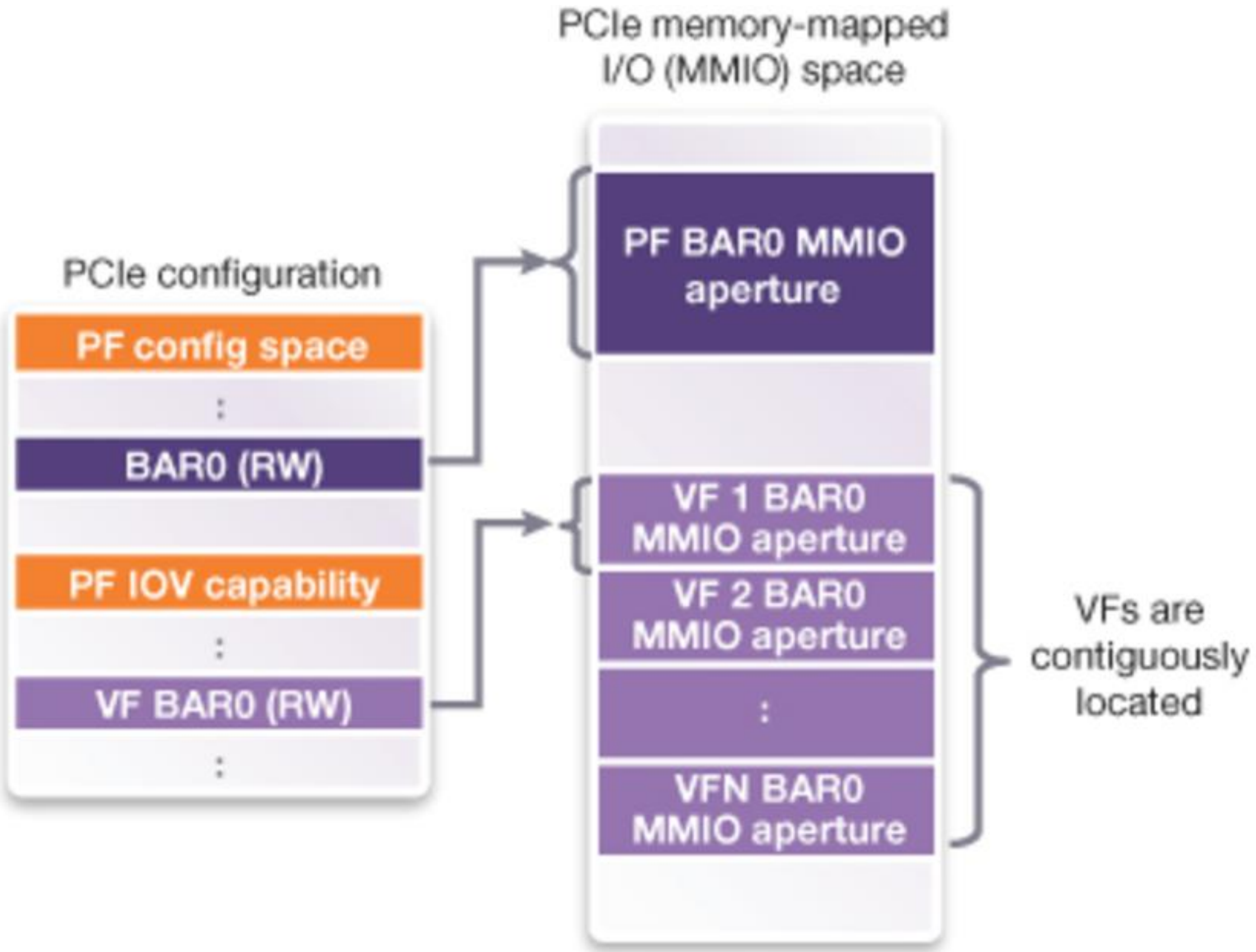
- MSI:
- http://fundasbykrishna.blogspot.com/2013/05/message-signaled-interrupt.html
- https://www.xilinx.com/Attachment/Xilinx_Answer_58495_PCIe_Interrupt_Debugging_Guide.pdf
- https://www.slideshare.net/anshumanbiswal/cjb0412001-anshumanaspesd2528presentation

- TLP:
- http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1
- https://www.xilinx.com/support/documentation/white_papers/wp350.pdf

SR-IOV (Single Root I/O Virtualization)

- A specification that allows PCIe devices to present itself as multiple virtual interfaces.
- Basically, you get the ability to share a single PCIe device as if there were multiple separate physical instances of it.
- High performance and scalable.
- Introduces the concept of physical functions (PFs) and virtual functions (VFs):
  - PF: full-featured PCIe functions. Discovered, managed, and manipulated like any other PCIe device.
  - VF: VFs are “lightweight” functions that lack configuration resources. They also have the ability to move data in and out like PFs. However, they cannot be configure as that would change the underlying configuration of the PF. They have their own PCIe config space to manage their resources.

SR-IOV (Single Root I/O Virtualization)



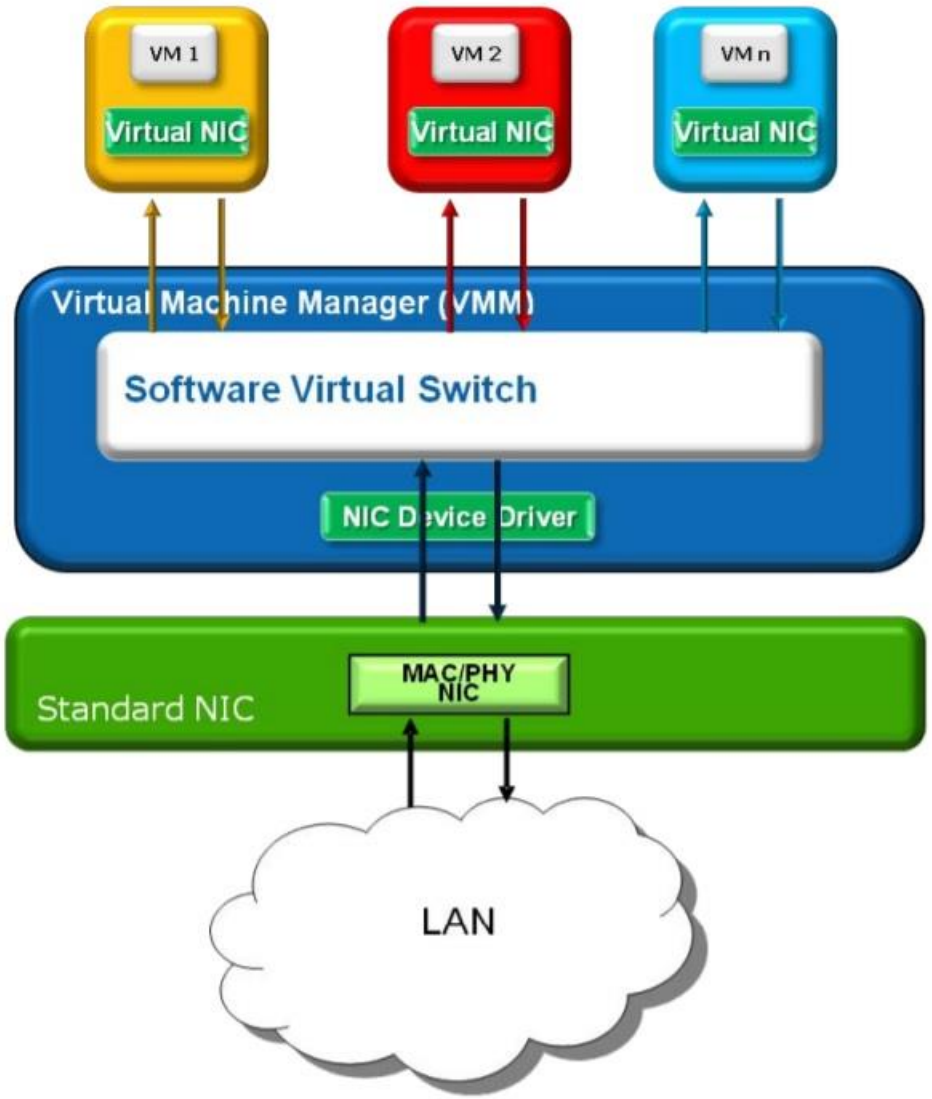
Why I/O Virtualization?

- For the systems with multiple VMs, we need a way to provide:
- Scalability to support I/O to all VMs
- Near native performance for all IO operations.
- Isolation:
  - Memory
  - Interrupts
  - Control operations
  - I/O operations

I/O Sharing Approaches

1. Software Based Sharing:
   - Utilizes emulation techniques to provide a logical I/O device to VM.
   - VMM/Hypervisor intercepts all the traffic issues by the guest driver running on VM.
   - Emulation software also performs address translation (guest addresses to host physical addresses)

   - Drawbacks:
     - Only provides sub-set of the physical device functionality
     - Significant CPU overhead to emulate the device and do data copy to and from the VM.



I/O Sharing Approaches

2. Direct Assignment:

- Provides a way for the physical device to directly to/from the VM guest memory.
- Bypasses VMMs emulation layer and significantly improves the overall throughput.
- Address translation is done by the platform hardware (Intel VT-x and VT-d)

- Drawbacks:
  - Limited scalability – a physical device can only be assigned to one VM.

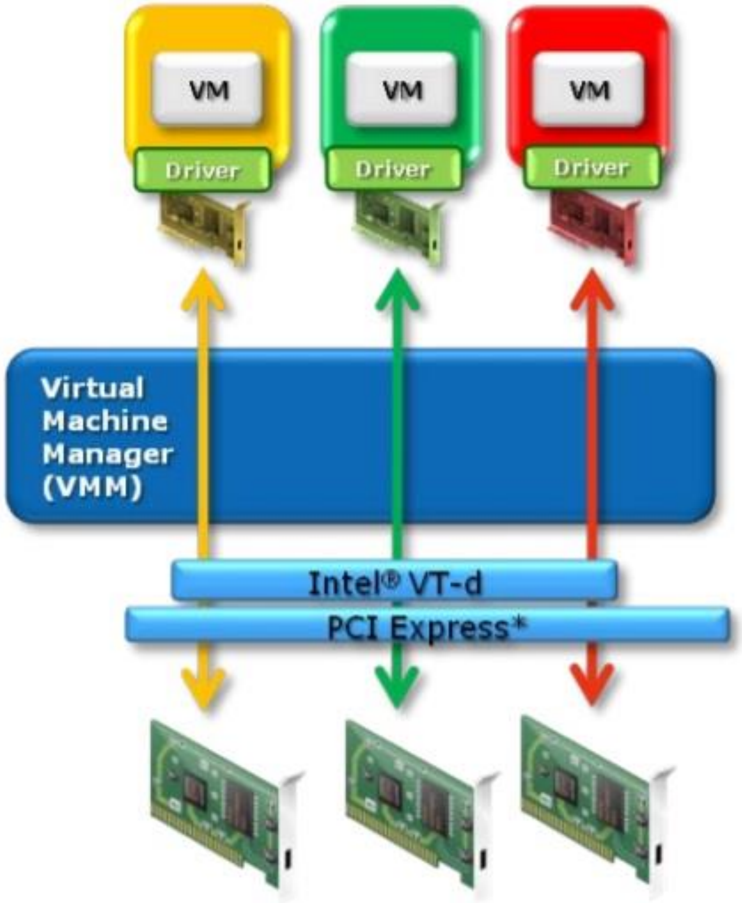
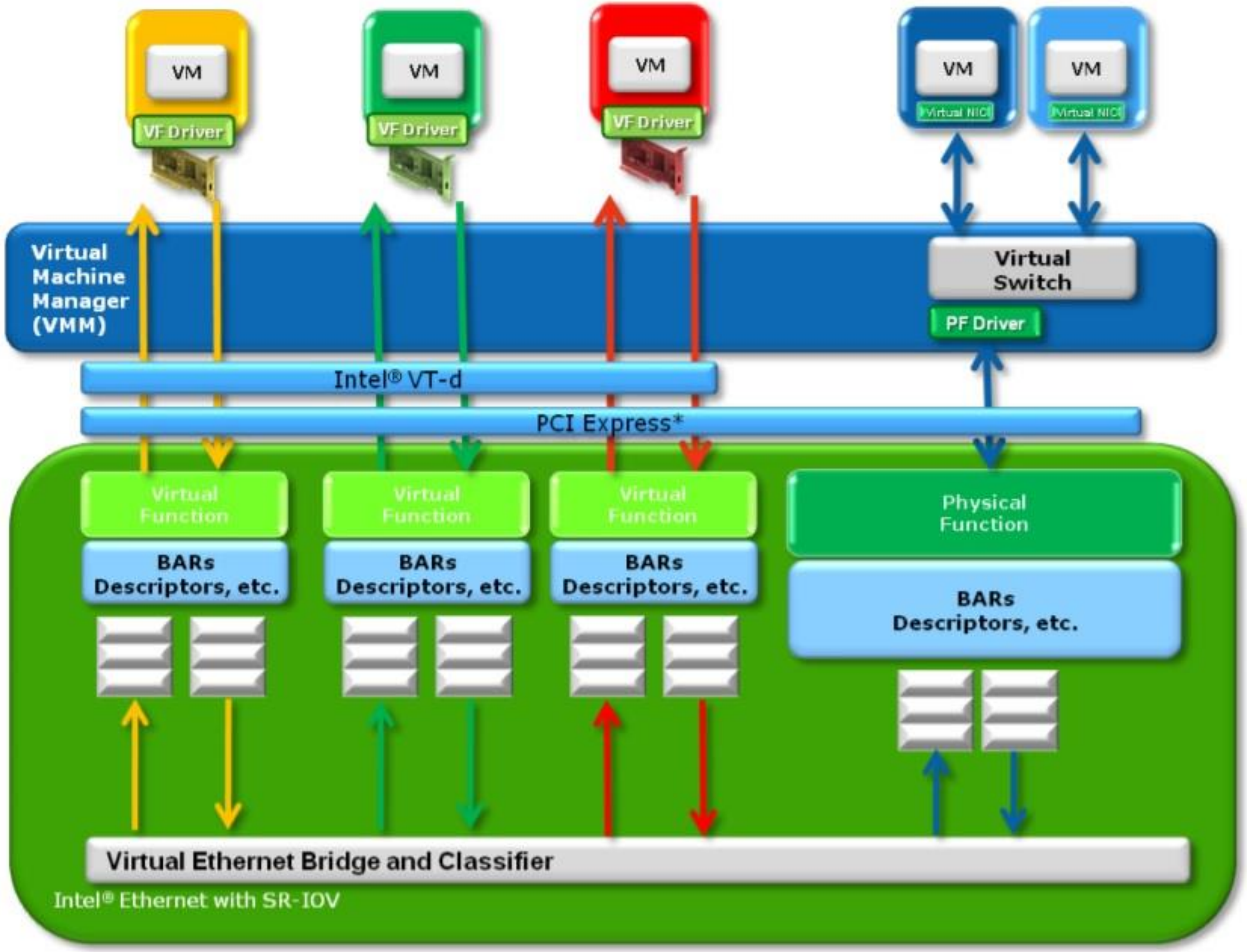


Figure 3. Direct Assignment

I/O Sharing Approaches

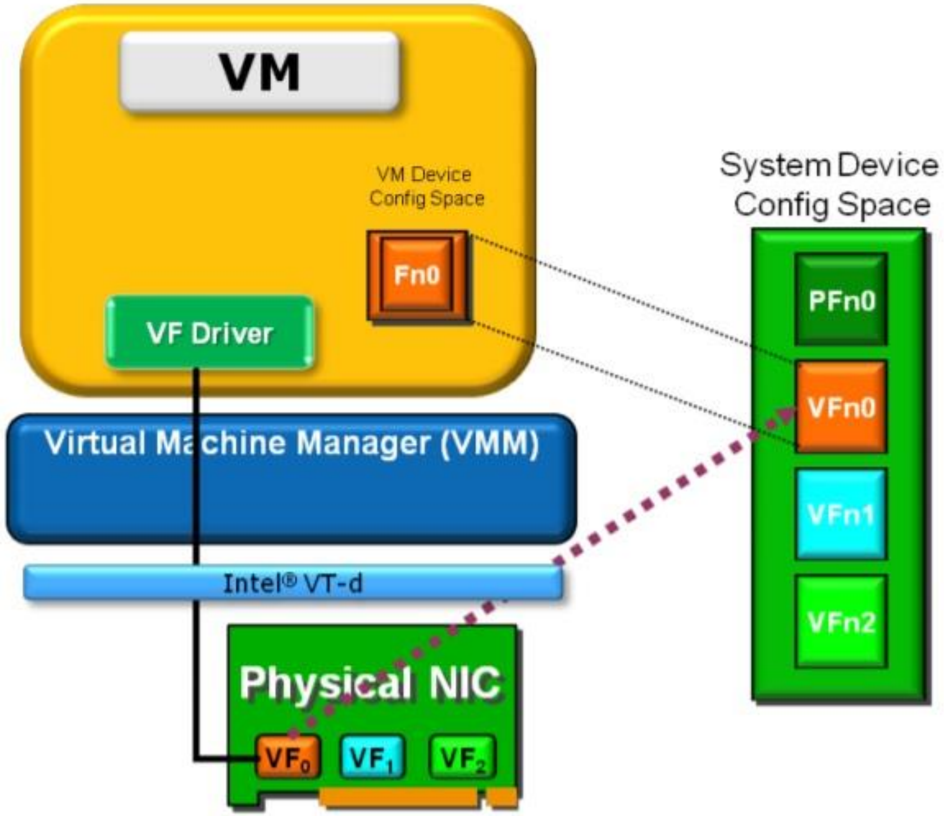
3. SR-IOV:

- Provides a way to have multiple virtual devices which are presented to VMs
- These devices replicate resources necessary for each VM to be directly connected to the I/O device so that the main data movement can occur without VMM involvement.



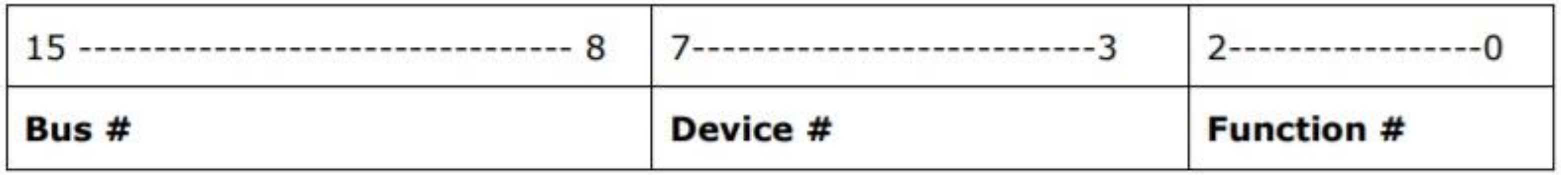
SR-IOV – more details

- SR-IOV is similar to direct assignment, but it adds device sharing support.
- A SR-IOV-capable device can be configured (usually by the VMM) to appear in the PCI configuration space as multiple functions, each with its own configuration space complete with Base Address Registers (BARs).
- The VMM assigns one or more VFs to a VM by mapping the actual configuration space the VFs to the configuration space presented to the virtual machine by the VMM.
- Hardware assisted memory translation (VT-x and VT-d) allows device to DMA directly to/from guest host memory.



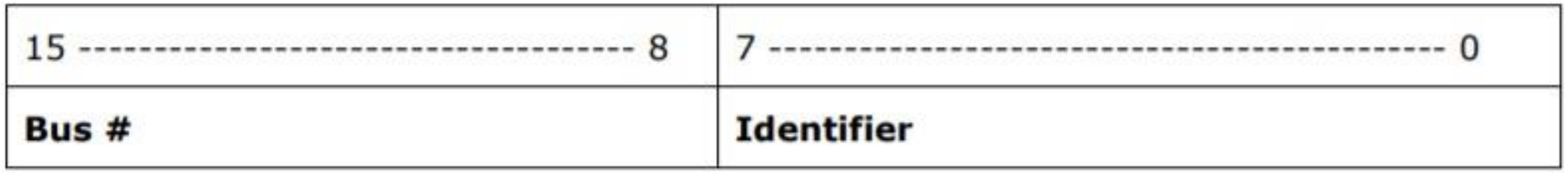
ARI – Alternate Routing-ID Interpretation

- Traditional PCIe addresses uses Bus-Device-Function method
- It can only support up to 8 functions.



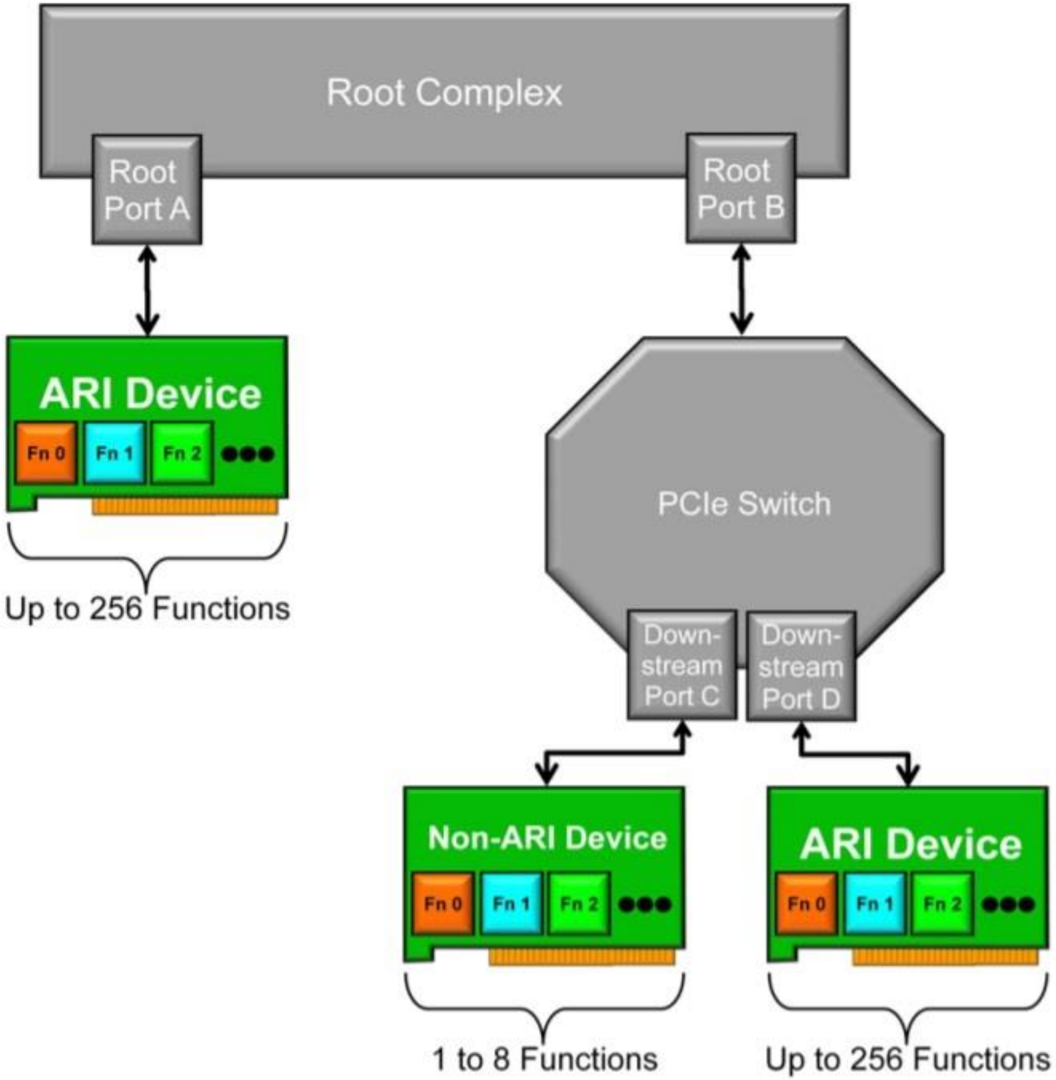
15 ------------------------------- 8	7-------------------------3	2---------------0
Bus #	**Device #**	**Function #**

- In order to support more than 8 VMs, an enhancement is made to the PCIe addressing scheme.
- Device # and Function # fields have been combined to a single field (Identifier).
- The device number in BDF addressing is implied as 0. This is due to the fact that each PCIe bus will have only one device.
- The new extension gives up to 256 functions per device.

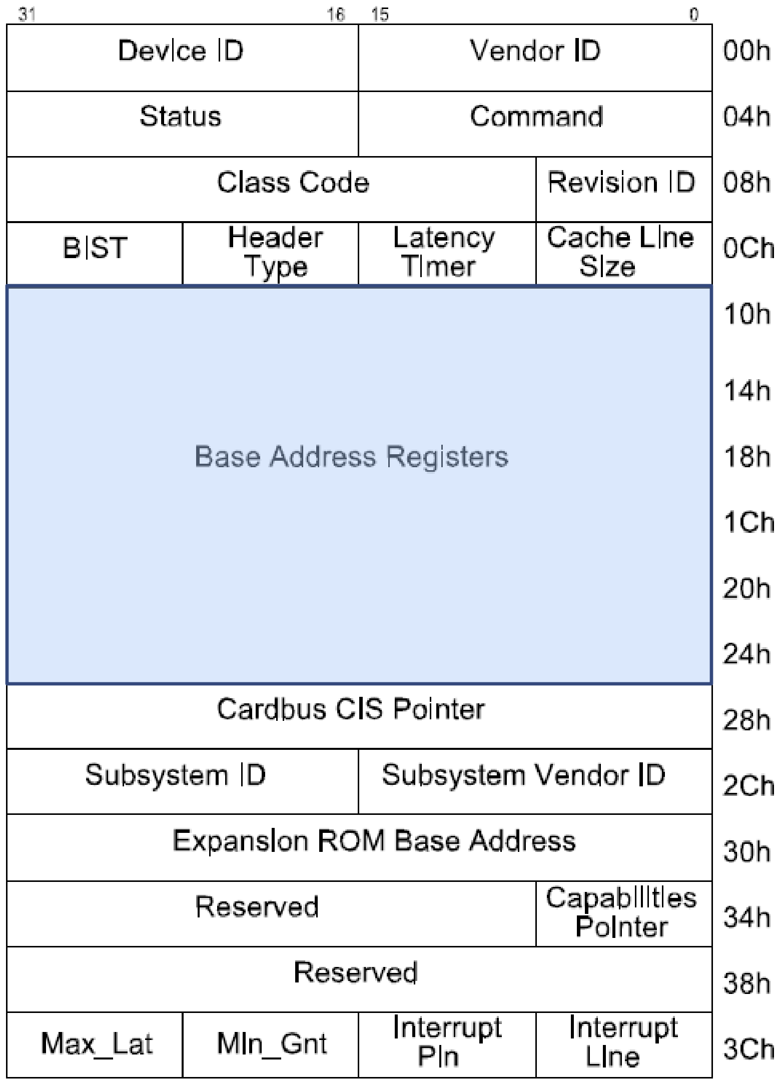


15 ------------------------------------- 8	7 --- 0
Bus #	**Identifier**

ARI – Alternate Routing-ID Interpretation



Base Address Registers (BARs)



31	16	15	0	
Devlce ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency TImer	Cache LIne Slze	0Ch
Base Address Registers				10h
				14h
				18h
				1Ch
				20h
				24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expanslon ROM Base Address				30h
Reserved			Capabllltles Polnter	34h
Reserved				38h
Max_Lat	MIn_Gnt	Interrupt Pln	Interrupt LIne	3Ch

- Base Address Registers point to the location in the system address space where the PCI device will be located
  - The device RAM, etc. (anything really, per the vendor)
- BARs are R/W and the BIOS programs them to set up the Memory Map
- BARs also indicate the amount of system memory required by the device
- PCI Configuration Registers provides space for up to 6 BARs (bytes 10h thru 27h)
  - BAR[0-5]
- Each BAR is 32-bits wide to support 32-bit address space locations
- Concatenating two 32-bit BARs provides 64-bit addressing capability

PF (Physical Function) Driver

- Physical Function Driver PF Driver is a specialized driver that manages global functions for SR-IOV devices and is responsible for configuring shared resources.
- Executes in hypervisor and is operates in a more privileged environment than a typical VF driver.
- Device configuration is only possible through the PF.
- PF driver is a regular device driver which is also capable of moving the data in and out of the device.
- PF driver is loaded before any VF driver and unloaded (if necessary) after all VF drivers.
- Device specific operations that globally affect all VMs should only be performed through PF. To achieve this, PF also needs to communicate to each VF:
  - For example, MTU change is done through PF.
  - PF communicates this change to each VF.

VF (Virtual Function) Driver

- The VF is a ‘lightweight’ PCIe function, containing the resources necessary for data movement.
- It is not a full-fledged PCIe device and only provides a mechanism to transfer data in and out.
- The VF driver resides in the VM and is a para-virtualized driver (aware that it is in a virtualized environment).
- In general the VF provides the ability to send and receive data and the ability to perform a reset. This reset only affects the VF itself, not the entire physical device.
- For actions that are beyond the VF reset or sending and receiving data, the VF driver needs to communicate with the PF Driver.
- Each VF has its own dedicated resources within the I/O device. For example, an Ethernet device will have dedicated Tx and Rx queues associated with dedicated BARs descriptors etc.