**Chapter-5**

**Peripheral Component Interconnect (PCI)**

## **Introduction to PCI**

A computer bus is used to transfer data from one location or device on the motherboard to the central processing unit where all calculations take place. There are two different parts of a Bus.

* Address bus‐transfers information about where the data should go
* Data bus‐transfers the actual data

CPU interacts with different peripherals through internal and expansion bus. An Internal Bus is a bus that operates only within the internal circuitry of the CPU, communicating among the internal caches of memory that are part of the CPU chip’s design. Microprocessors communicate with its internal and external components through the bus, are mainly divided into two categories:

1. Front-side Bus: Another name for the system bus. The bus that connects the CPU to main memory on the motherboard. The system bus is also called the memory bus, local bus, or host bus.
2. Backside bus: A microprocessor bus that connects the CPU to a Level 2 cache. Typically, a backside bus runs at a faster clock speed than the frontside bus that connects the CPU to main memory.

An expansion bus is a collection of wires and protocols that allows the expansion of a computer by inserting printed circuit boards (expansion boards).

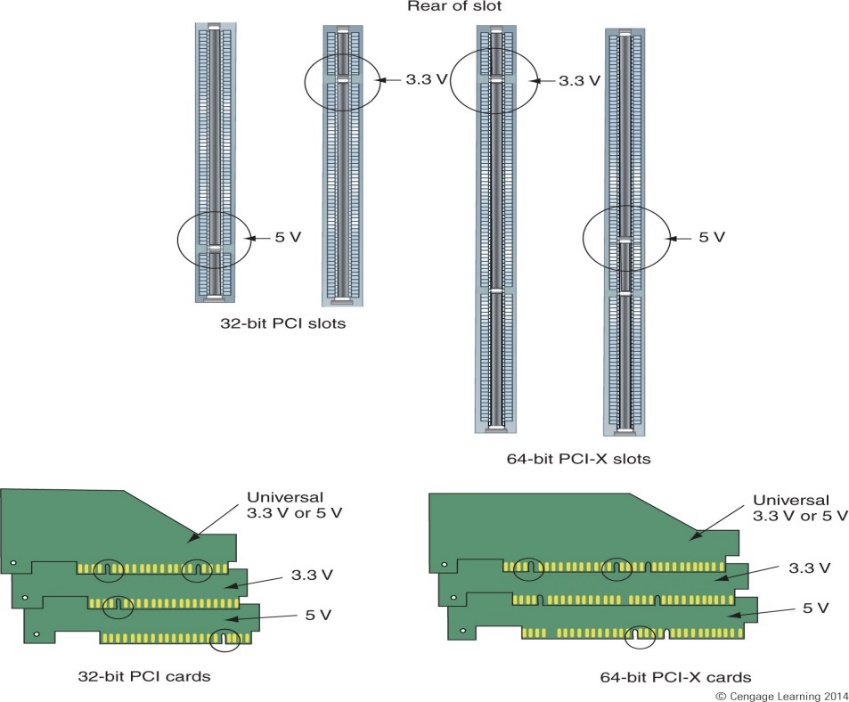
* AT Bus: The AT bus, which runs at 8 megahertz and has a 16- bit data path. The AT bus is sometimes referred to as the ISA bus.
* ISA Bus: Short for Industry Standard Architecture bus. 16 bit bus. ISA began to be replaced by the PCI local bus architecture.
* MCA: IBM introduced the Micro Channel Architecture (MCA) in 1987. 32-bit bus.
* EISA Bus: A more successful alternative to the MCA bus is the Extended Industry Standard Architecture (EISA), a high-speed 32-bit bus architecture developed by a group of IBM's competitors
* VL-bus: Short for VESA Local-Bus, a local bus architecture created by the Video Electronics Standards Association (VESA). Although it was quite popular in PCs made in 1993 and 1994, it has been overshadowed by a competing local bus architecture called PCI.
* PCI Bus: Short for Peripheral Component Interconnect, a local bus standard developed by Intel Corporation. PCI is a 64-bit bus, though it is usually implemented as a 32-bit bus. It can run at clock speeds of 33 or 66 MHz. At 32 bits and 33 MHz, it yields a throughput rate of 133 MBps.
* PCI-X Bus: Short for PCI extended, an enhanced PCI bus. PCI-X is backward-compatible with existing PCI cards. It improves upon the speed of PCI from 133 MBps to as much as 1 GBps.
* NuBus: The expansion bus for versions of the Macintosh computers starting with the Macintosh II and ending with the Performa. Current Macs use the PCI bus.
* SMBus: The System Management Bus (SMBus) is a two-wire interface through which simple power-related chips can communicate with rest of the system. With the SMBus, a device can provide manufacturer information, tell the system what its model or part number is, save its state for a suspend event, report different types of errors, accept control parameters and return its status.

Peripheral Component Interconnect (PCI) also called Compatible PCI. It’s a bus protocol developed by Intel around 1993. The purpose is to attach/interconnect local hardware devices to a computer. PCI is integrated into the chipset, forming a “backbone”. Logically speaking, the Chipset is a PCI System

intended for use as an interconnect mechanism between highly integrated peripheral controller components, peripheral add-in cards, and processor/memory systems. It is a parallel bus, synchronous to a single bus clock. PCI component interface is processor independent

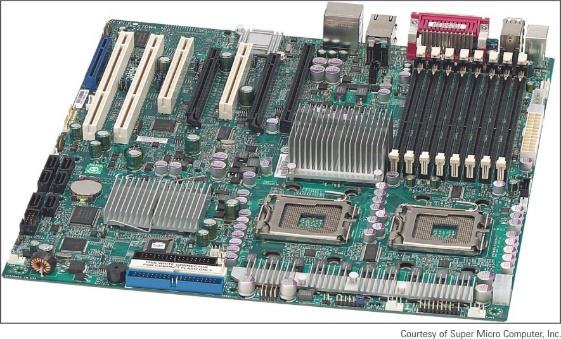
## **2.0 PCI**

The first version of conventional PCI found in consumer desktop computers was a 32-bit bus using a 33 MHz bus clock and 5 V signaling, although the PCI 1.0 standard provided for a 64-bit variant as well. These have one locating notch in the card. Version 2.0 of the PCI standard introduced 3.3 V slots, physically distinguished by a flipped physical connector to prevent accidental insertion of 5 V cards. Universal cards, which can operate on either voltage, have two notches. Version 2.1 of the PCI standard introduced optional 66 MHz operation. Conventional PCI supports 4 types of slots and 6 possible PCI card configurations as shown in Figure 1. Because a card can be damaged if installed in the wrong voltage slot, a notch in a PCI slot distinguishes between a 5 V slot and a 3.3 V slot.



**Figure 1. Conventional PCI add in card and slots**

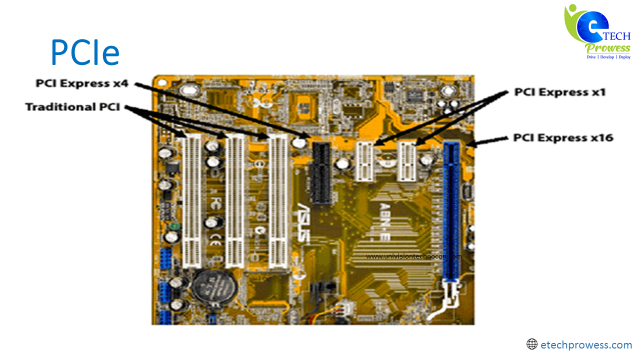
A server-oriented variant of conventional PCI, called PCI-X (PCI Extended) operated at frequencies up to 133 MHz for PCI-X 1.0 and up to 533 MHz for PCI-X 2.0. PCI-X uses a 64- bit data path and all revisions are backward compatible. An internal connector for laptop cards, called Mini PCI, was introduced in version 2.2 of the PCI specification. The PCI bus was also adopted for an external laptop connector standard – the CardBus. The first PCI specification was developed by Intel, but subsequent development of the standard became the responsibility of the PCI Special Interest Group (PCI-SIG).



PCI-X slots

**Figure 2. Motherboard with PCI-X slots**

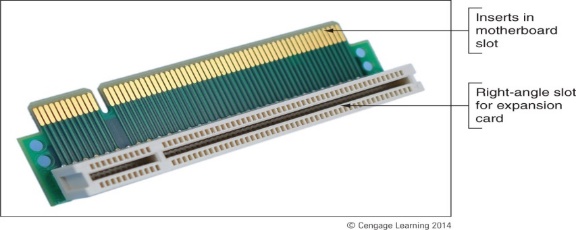
Conventional PCI and PCI-X are sometimes called Parallel PCI in order to distinguish them technologically from their more recent successor PCI Express, which adopted a serial, lane-based architecture. PCI and PCI Express are not compatible; the slots have different configurations and do not fit with the cards. However, most motherboards have a combination of PCI and PCI Express cards as shown in figure 3. PCIe is expected to replace both PCI and PCI-X in the future. PCIe comes in four different slot sizes- PCI Express x1, x4, x8, and x16.



**Figure 3. Motherboard with PCIe slots**

PCI also supports PCI Riser cards which can be use be used as further expansion slot. It installs in a PCI slot and provides another slot at a right angle as shown in figure 4. It is used to fit PCI, PCIe, and PCI-X cards into a low-profile or slimline case. PCI riser card provides a 3.3-V slot or 5-V slot depending on which direction the card is inserted in the PCI slot

Conventional PCI's heyday in the desktop computer market was approximately 1995–2005. PCI and PCI-X have become obsolete for most purposes; however, they are still common on modern desktops for the purposes of backwards compatibility and the low relative cost to produce. Many kinds of devices previously available on PCI expansion cards are now commonly integrated onto motherboards or available in USB and PCI Express versions.



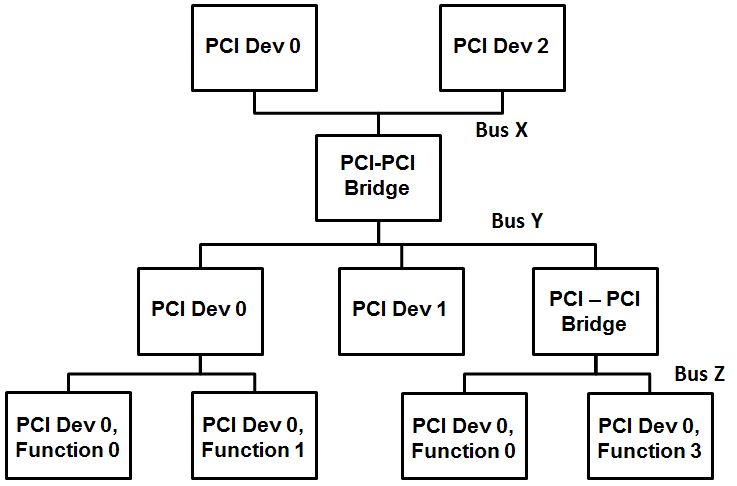
**Figure 4. PCI Riser card**

### **2.1 Features of Conventional PCI**

1. **Singling Environment:** Support both 3.3 and 5 volt signaling environments.
2. **Reliability:**Itoffers the ability to replace modules without disturbing a system’s operation called as hot plug and hot swap.
3. **Speed:**It can transfer up to 132 MB per second.
4. **Configurability:** The ability to configure a system automatically means automatically identify the interfacing systems and assigns new addresses.
5. **Synchronous bus architecture:**PCI is a synchronous bus where data transfer takes place according to a system clock.
6. **32 and 64 bit addressing:**The PCI bus also supports 64 bit addressing with the same 32 bit connector.
7. **Large bandwidth:** It can handle both 32 bit as well as 64 bit data hence the maximum bandwidth will be 132 MB per second.

### **2.2 Generic PCI Topology**

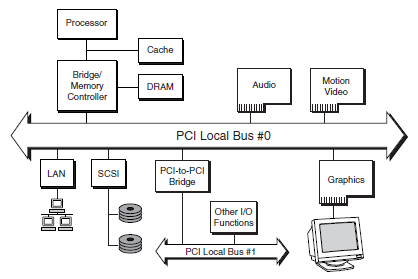
Figure. 5 shows the basic PCI topology. It supports up to 256 Buses and each Bus can have up to 32 devices attached. Each Device on the Bus can extend up to 8 Functions. Buses are interconnected by PCI-PCI Bridges. Multiple bridges can be connected through a bridge interface and are enumerated



**Figure 5. Generic PCI Topology**

### **2.3 PCI Local Bus Overview**

The PCI local Bus is a high performance 32-bit or 64-bit bus with multiplexed address and data lines. The bus is intended for use as an interconnect mechanism between highly integrated peripheral controller components, peripheral add-in boards, and processor/memory systems. Figure. 6 shows a typical PCI local Bus system architecture. In this example the processor/cache/memory subsystem is connected to PCI through a PCI bridge. This bridge provides a low latency path through which the processor may directly access PCI devices mapped anywhere in the memory or I/O address spaces. The bridge may optionally include such functions as data buffering/ posting and PCI central functions (e.g. arbitration).



**Figure 6. PCI local bus system architecture**

## **3.0 PCI Express**

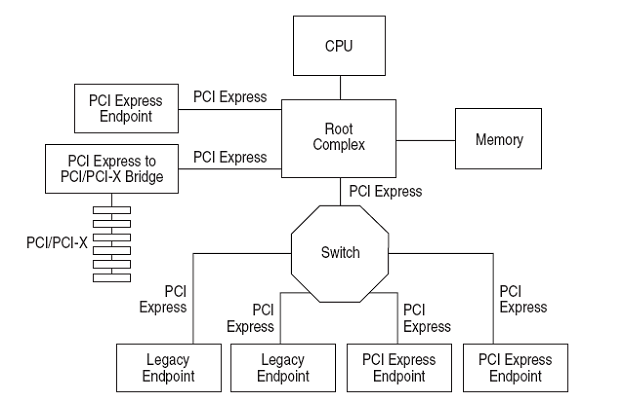
It is a high performance, general purpose I/O interconnect defined for wide range of computing and communication platforms. Along with key features of PCI, usage model, load-store architecture, and software interfaces, its parallel bus implementation is replaced by a highly scalable, fully serial interface.

### **3.1 Features of PCIe**

1. point-to-point interconnects
2. Switch-based technology
3. packetized protocol
4. Power Management
5. Quality Of Service (QoS)
6. Hot-Plug/Hot-Swap support
7. Data Integrity
8. Error Handling

### **3.2 Generic PCIe Topology**

Figure. 7 shows the basic PCIe topology. It supports up to 256 Buses and each Bus can have up to 32 devices attached similar to that of PCI topology. Each Device on the Bus can extend up to 8 Functions. Each Function can implement up to 4 KB of configuration space. The Root Complex connects the processor to the system memory and components



**Figure 7. PCIe Topology**

**Root Complex-** denotes the root of an I/O hierarchy that connects the CPU/memory subsystem to the I/O

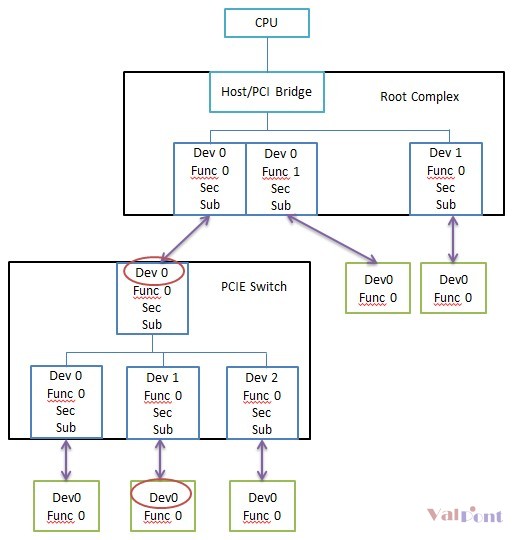
**Endpoints-** classified as either legacy, PCI Express, or Root Complex Integrated Endpoints. Must be a Function with Type 00h Configuration Space header

**Switch-** defined as a logical assembly of multiple virtual PCI-to-PCI Bridge devices

**PCI Express to PCI/PCI-X Bridge**- must be a Function with Type 01h Configuration Space header

### **3.3 PCIe Enumeration**

When system first boots up and PCIE enumeration is not done yet, our example PCIE system looks like below (Figure 8.1).



**Figure 8.1 PCIe Enumeration**

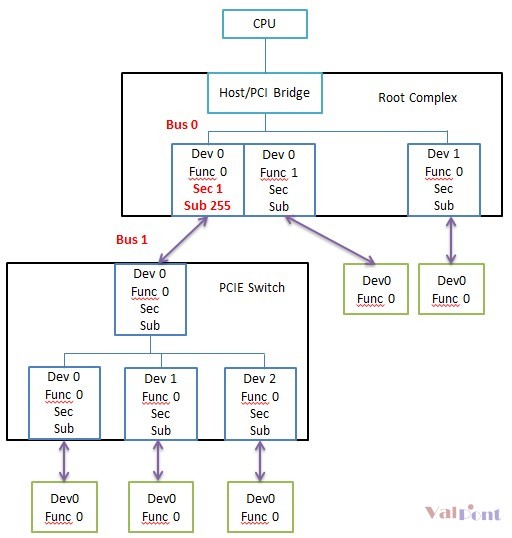
Some observation. We have root complex (RC) which connects to host CPU via host/PCI bridge. Note root complex also resides on host side. Inside root complex, we have two devices, dev 0 and dev 1. Dev 0 is a multi-function device and consists of function 0 and function 1. Dev 1 is single function device. RC dev 0 and dev 1 are not end point devices and they are of bridge type, also called PCIE to PCIE (P2P) virtual bridge.

There is a PCIE switch in the system which is connected to RC dev0 func0 via a PCIE link. This PCIE switch has four P2P bridges and three downstream bridges are connected to a PCIE end point respectively.

Note  
1. As highlighted in diagram, all devices attached to downstream side of a PCIE link must be device 0.  
2. PCIE link is a point to point connection and P2P bridge, either in RC or in switch, is needed to connected multiple PCIE devices. However, the connection among P2P bridges, either inside RC or inside switch, is multi-drop and it is NOT a PCIE link.

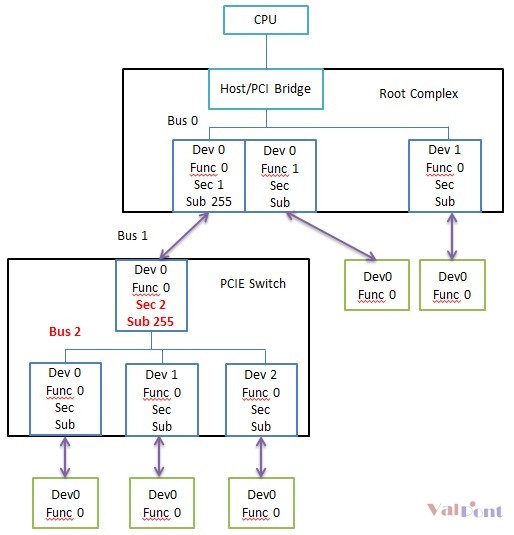
What PCIE enumeration does is to assign PCIE bus number to each PCIE link and P2P bridge connection, and properly fill up secondary bus number and subordinate bus number inside each P2P bridge so that software running in CPU can uniquely identify each PCIE device and P2P bridges combined can properly route the transaction to the correct target PCIE device.

 Now let’s get started. Software first assigns bus 0 to RC connection. Then it reads bus 0 dev0 configuration space and figures this device is a bridge since configuration header is type 1. Then it assigns bus 1 to this device’s downstream PCIE link and updates secondary (sec) bus number to be 1 and subordinate (sub) bus number to be 255. Sec bus number specifies the minimum bus number in the tree under this bridge and sub bus number is the max bus number. Software uses 255 for now since it hasn’t gone through the tree yet so it doesn’t know how many devices down there.



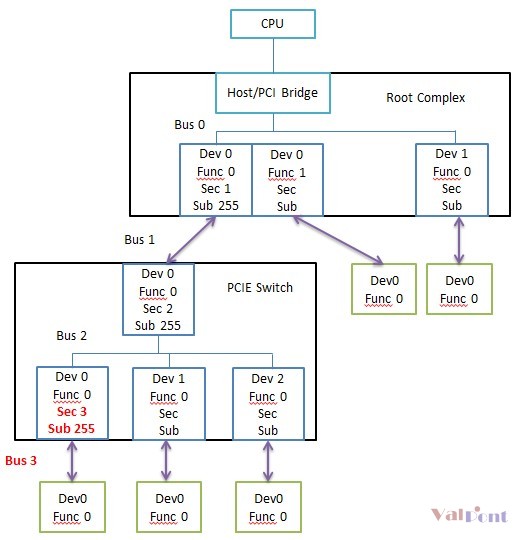
**Figure 8.2 PCIe Enumeration**

Software reads bus 1 dev0 and figures it is a P2P bridge and then assigns bus 2 to its downstream link (note it is not a PCIE link but a connection among P2P bridges) and updates bus 1 dev0 sec bus number to 2 and sub bus number to 255.



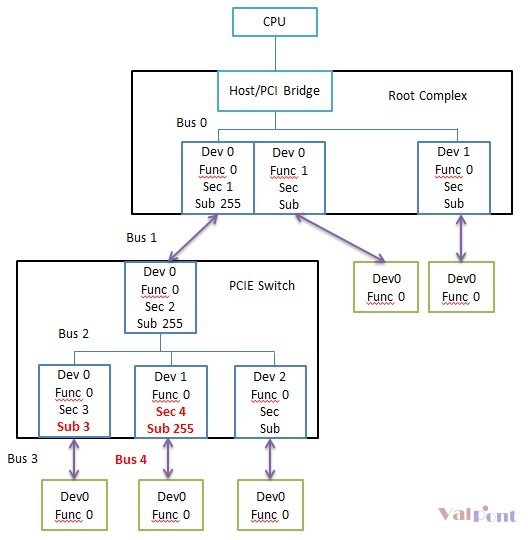
**Figure 8.3 PCIe Enumeration**

Software reads bus 2 dev 0 and figures it is a P2P bridge. It does the same thing as above, assign bus 3 and updates sec bus num to 3 and sub bus num to 255.



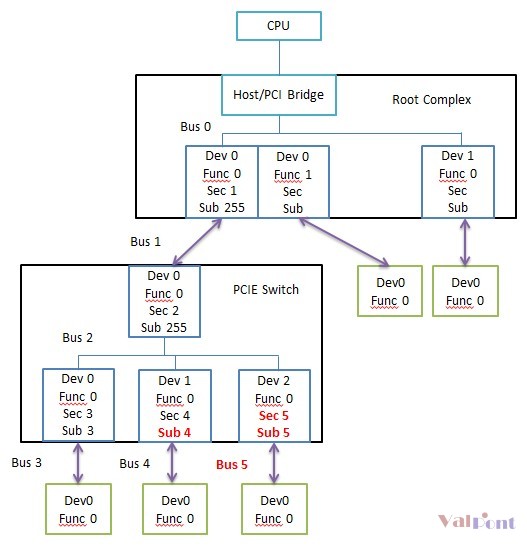
**Figure 8.4 PCIe Enumeration**

Software reads bus 3 dev0 and figures it is a PCIE end point. So it goes back to update bus 2 dev0 sub bus number from 255 to 3. Then it checks if bus 2 has dev 1. Find it. Assign bus 4 and update bus 2 dev 1 sec bus num to 4 and sub bus num to 255.



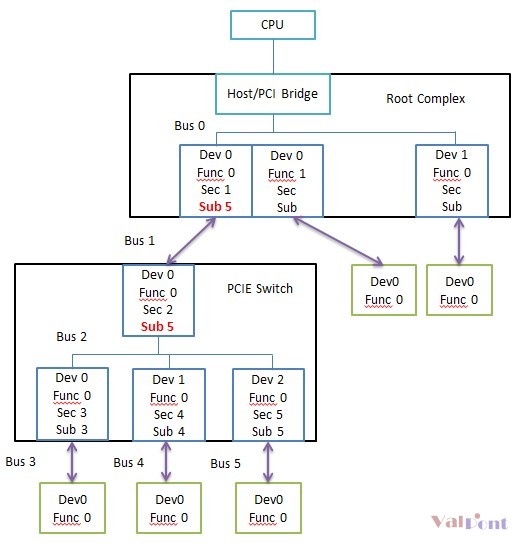
**Figure 8.5 PCIe Enumeration**

Software reads bus 4 dev0 and figures it is a PCIE end point. So it goes back to update bus 2 dev1 sub bus number from 255 to 4. Then it checks if bus 2 has dev 2. Find it. Assign bus 5 and update bus 2 dev 2 sec bus num to 5 and sub bus num to 255. It then reads and figures bus 5 dev 0 is end point so it goes back to update bus 2 dev 2 sub bus num to 5.



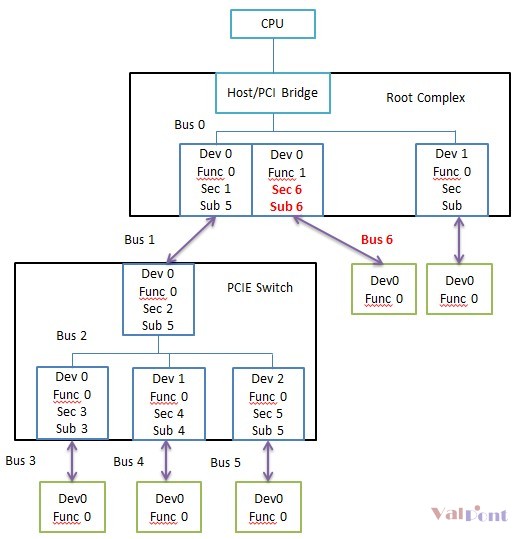
**Figure 8.6 PCIe Enumeration**

Software further checks if bus 2 has dev 3. It goes ahead and read bus dev 3 configuration space. Since there is no dev 3, this transaction eventually times out causing a master abort inside RC and RC returns data of all one’s to software. So software knows no bus 2 dev 3. It then goes back to update bus 1 dev 0 sub bus number from 255 to 5 and further goes back to update bus 0 dev 0 func 0 sub bus number from 255 to 5.



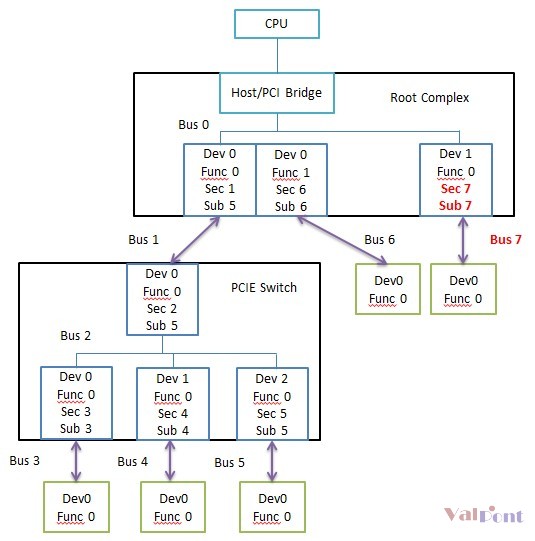
**Figure 8.7 PCIe Enumeration**

Software already know bus 0 dev 0 is a multi-function device in step 1. So it moves to bus 0 dev0 func 1. Assign bus 6 to its downstream link. Reads bus 6 dev 0 and figures it is a end point. Eventually bus 0 dev0 func1 sec bus num is assigned to 6 and sub bus num also to 6.



**Figure 8.8 PCIe Enumeration**

Software checks and sees bus 0 dev 1. Assign bus 7 and update both sec bus number and sub bus number to be 7. Software further checks if there is bus 0 dev 2. No such device. Transaction times out. That’s it. Software gets the whole system figured out and properly configured. PCIE enumeration is done.



**Figure 8.9 PCIe Enumeration**

## **4.0 PCI Programming Model**

### **4.1 PCI Address Spaces**

PCI implements three address spaces:

1. PCI Configuration Space (up to 256 Bytes)
   * Required/standard. Defined in the specifications. Every PCI device has a configuration space.
2. PCI Memory-mapped space
   * Optional. Dependent on whether the device manufacturer needs to map system memory to the PCI device
3. PCI I/O-mapped space
   * Optional. Same as PCI Memory Space

### **4.2 PCI Express (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)

1. PCIe Memory-mapped space
   * Optional. Dependent on whether the device manufacturer needs to map system memory to the PCI device
2. PCIe I/O-mapped space
   * Optional. Same as PCI Memory Space
3. PCIe Message Space
   * For low-level protocol messaging/interrupts. We don’t get into this in this class

### **4.3 PCI Configuration Space**

Configuration Space is intended for configuration, initialization, and catastrophic error handling functions. Its use should be restricted to initialization software and error handling software. All operational software must continue to use I/O and/or Memory Space accesses to manipulate device registers.

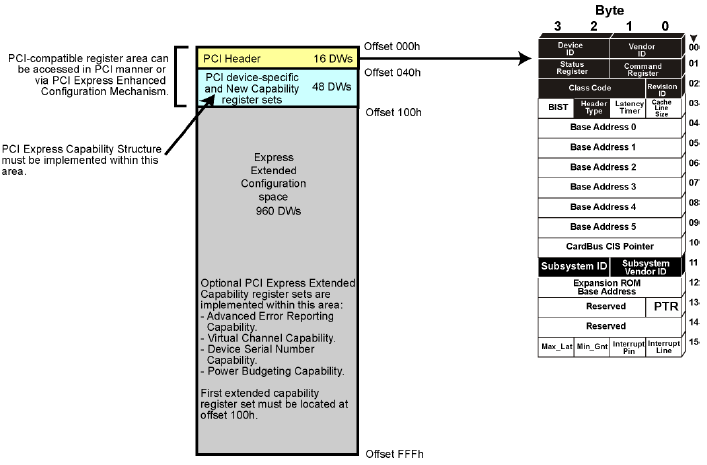
It defines the programming model and usage rules for the configuration register space organization. A device's Configuration Space must be accessible at all times, not just during system boot. It imposes a specific record structure or template on the 256-byte space which is divided into **predefined header region** and **device dependent region**

**predefined header region**

Consists of fields that uniquely identifies the device and allow the device to be generically controlled. It is divided into 2 parts, first 16 bytes are defined same for all devices and the remaining bytes are dependent on the base function of the device

**Device dependent region**

Contains device specific information



**Figure 9. PCIe Address Space**

### **4.4 Accessing Configuration Space**

There are 2 ways to access the compatible PCI configuration space registers (0 to 255)

* Port IO or Memory-mapped IO

And 1 way to access the extended configuration space offered by PCI Express (255 to 4KB)

* Memory-mapped IO

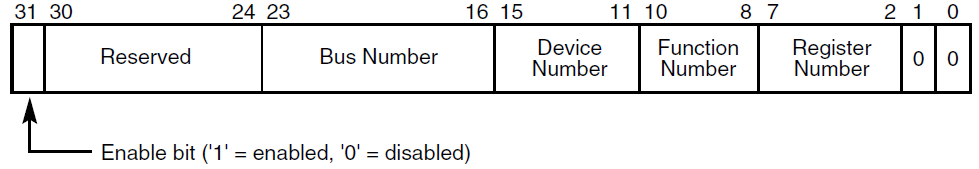
Accessing PCI via Port I/O performed in a couple situations:

* When in Real Mode, when accesses to 32-bit memory space is limited. Real Mode may be reentered even after the system has transitioned to Protected mode which was observed in Legacy BIOS and not in UEFI generally
* Before PCIEXBAR has been configured

#### **4.4.1 Compatible PCI Configuration Space**

This refers to the software generation of PCI configuration transactions those generated by the CPU/BIOS. Compatible PCI provides 256 bytes of Configuration address space to the CPU/BIOS, who in turn programs the registers contained therein to configure the device and system parameters. Compatible PCI is configured using two 32-bit port I/O address/data pair (**CONFIG\_ADDRESS, CONFIG\_DATA**) to generate configuration transactions

###### 4.4.1.1 I/O Port CONFIG\_ADDRESS (CF8h)



**Figure 10. CONFIG\_ADDRESS (CF8h)**

* Bit 31 when set, all reads and writes to CONFIG\_DATA are PCI Configuration transactions
* Bits 30:24 are read-only and must return 0 when read
* Bits 23:16 select a specific Bus in the system (this 8 bit field decides that a PCI system can support up to 256 buses)
* Bits 15:11 specify a Device on the given Bus (this 5 bit field decides that up to 32 devices can be connected to a PCI bus)
* Bits 10:8 Specify the function of a device (this 3 bit field decides that a device can have up to 8 functions )
* Bits 7:0 Select an offset within the Configuration Space (256 bytes max, DWORD-aligned as bits 1:0 are hard-coded 0)
* Addresses are often given in B/D/F, Offset notation

###### **4.4.1.2 I/O Port CONFIG\_DATA (CFCh)**

CONFIG\_DATA can be accessed in DWORD, WORD, or BYTE configurations. Reads and Writes to CONFIG\_DATA with Bit 31 in CONFIG\_ADDRESS set/enabled results in a PCI Configuration transaction to the device specified in CONFIG\_ADDRESS. PCI spec says that if Bit 31 is not enabled, then the transaction is forwarded out as Port I/O

#### **4.4.2 Compatible PCI Configuration Space Header**

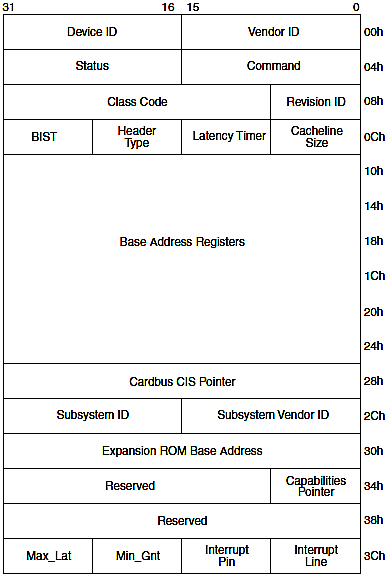
Implemented in of Configuration Space Header in PCIe is same as that in PCI giving it backward compatibility as well. In general supports three header types (0-2)

* Type 0 represents General Endpoint Device
* Type 1 represent PCI-to-PCI Bridge
* Type 2 represent CardBus Bridge which is not used at present

Following Configuration Space header represent an Endpoint Device (Type 00h). It is divided into 2 parts:

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

These registers are used for Device identification, Control, Status and Miscellaneous operations.



**Figure 11. Type 0 Configuration Space Header**

**Vendor ID**

This field identifies the manufacturer of the device. Valid vendor identifiers are allocated by the PCI SIG to ensure uniqueness.0 FFFFh is an invalid value for Vendor ID.

**Device ID**

This field identifies the particular device. This identifier is allocated by the vendor.

**Revision ID**

This register specifies a device specific revision identifier. The value is chosen by the vendor. Zero is an acceptable value. This field should be viewed as a vendor defined extension to the Device ID.

**Header Type**

This byte identifies the layout of the second part of the predefined header (beginning at byte 10h in Configuration Space) and also whether or not the device contains multiple functions. Bit 7 in this register is used to identify a multi-function device. If the bit is 0, then the device is single function. If the bit is 1, then the device has multiple functions. Bits 6 through 0 identify the layout of the second part of the predefined header. The encoding 00h specifies the layout shown in Figure 6-1. The encoding 01h is defined for PCI-to-PCI bridges and is defined in the document PCI to PCI Bridge Architecture Specification. The encoding 02h is defined for a Card Bus bridge and is documented in the PC Card Standard. All other encodings are reserved.

**Class Code**

The Class Code register is read-only and is used to identify the generic function of the device and, in some cases, a specific register level programming interface. The register is broken into three-byte size fields. The upper byte (at offset 0Bh) is a base class code which broadly classifies the type of function the device performs. The middle byte (at offset 0Ah) is a sub-class code which identifies more specifically the function of the device. The lower byte (at offset 09h) identifies a specific register-level programming interface (if any) so that device independent software can interact with the device. Encodings for base class, sub-class, and programming interface are provided in Appendix D. All unspecified encodings are reserved.

**Command Register**

The Command register provides coarse control over a device's ability to generate and respond to PCI cycles. When a 0 is written to this register, the device is logically disconnected from the PCI bus for all accesses except configuration accesses. All devices are required to support this base level of functionality. Individual bits in the Command register may or may not be implemented depending on a device’s functionality. For instance, devices that do not implement an I/O Space will not implement a writable element at bit location 0 of the Command register.

**Status Register**

The Status register is used to record status information for PCI bus related events. Devices may not need to implement all bits, depending on device functionality. For instance, a device that acts as a target, but will never signal Target-Abort, would not implement bit 11. Reserved bits should be read-only and return zero when read.

**CacheLine Size**

This read/write register specifies the system cacheline size in units of DWORDs. This register must be implemented by master devices that can generate the Memory Write and Invalidate command.

**Latency Timer**

This register specifies, in units of PCI bus clocks, the value of the Latency Timer for this PCI bus master

**Built-in Self-Test (BIST)**

This optional register is used for control and status of BIST. Devices that do not support BIST must always return a value of 0

**CardBus CIS Pointer**

This optional register is used by those devices that want to share silicon between CardBus and PCI. The field is used to point to the Card Information Structure (CIS) for the CardBus card.

**Interrupt Line**

The Interrupt Line register is an eight-bit register used to communicate interrupt line routing information. The register is read/write and must be implemented by any device (or device function) that uses an interrupt pin. POST software will write the routing information into this register as it initializes and configures the system

**Interrupt Pin**

The Interrupt Pin register tells which interrupt pin the device (or device function) uses. A value of 1 corresponds to INTA#. A value of 2 corresponds to INTB#. A value of 3 corresponds to INTC#. A value of 4 corresponds to INTD#.

**MIN\_GNT and MAX\_LAT**

These read-only byte registers are used to specify the device’s desired settings for Latency Timer values. For both registers, the value specifies a period of time in units of ¼ microsecond. Values of 0 indicate that the device has no major requirements for the settings of Latency Timers

**Subsystem Vendor ID and Subsystem ID**

These registers are used to uniquely identify the add-in card or subsystem where the PCI device resides. They provide a mechanism for add-in card vendors to distinguish their addin cards from one another even though the add-in cards may have the same PCI controller on them (and, therefore, the same Vendor ID and Device ID).

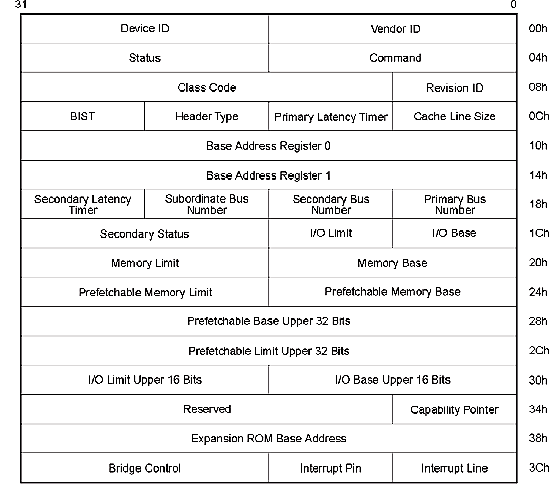
**Capabilities Pointer**

This optional register is used to point to a linked list of new capabilities implemented by this device. This register is only valid if the “Capabilities List” bit in the Status Register is set. If implemented, the bottom two bits are reserved and should be set to 00b. Software should mask these bits off before using this register as a pointer in Configuration Space to the first entry of a linked list of new capabilities.

**Base Address Register**

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 PCI Configuration Registers provides space for up to 6 BARs (bytes 10h through 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

For a Bridge device (Type 01h), initial 16 bytes are same, apart from early mentioned registers, following few registers are given priority.



**Figure 12. Type 1 Configuration Space Header**

**Primary Bus Number**- denotes the bus in which device sits

**Secondary Bus Number**- denotes the bus in which is spawned from the device

**Subordinate Bus Number** - denotes the maximum depth of the bus in the hierarchy