Introduction to PCIe driver

03.12.2020

Introduction to PCIe driver

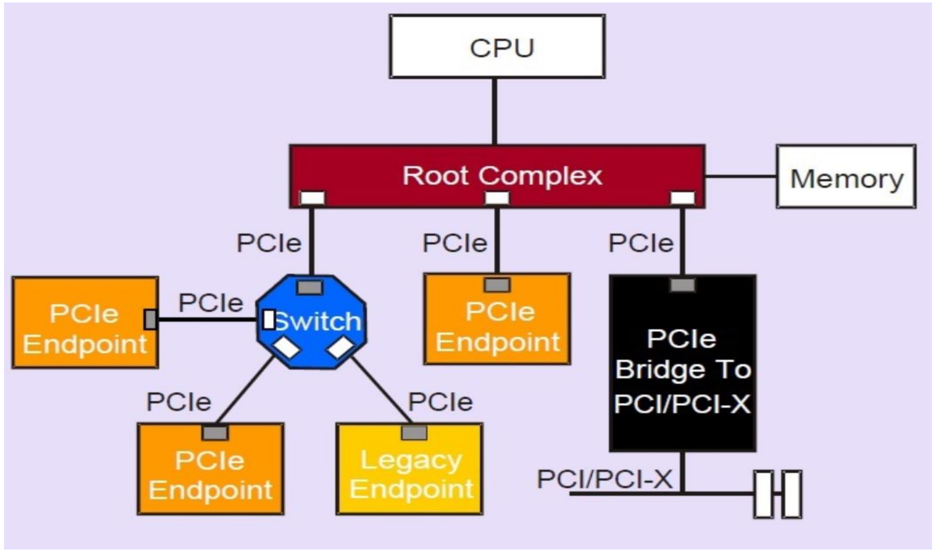
03.12.2020

Agenda

Agenda

PCI Architecture

- PCI Express is based on point-to-point topology with separate serial links connecting every device to the root complex (host).
- PCI Express bus link supports full-duplex communication between any two endpoints.
- The PCI Express link between two devices can vary in size from one to 32 lanes.

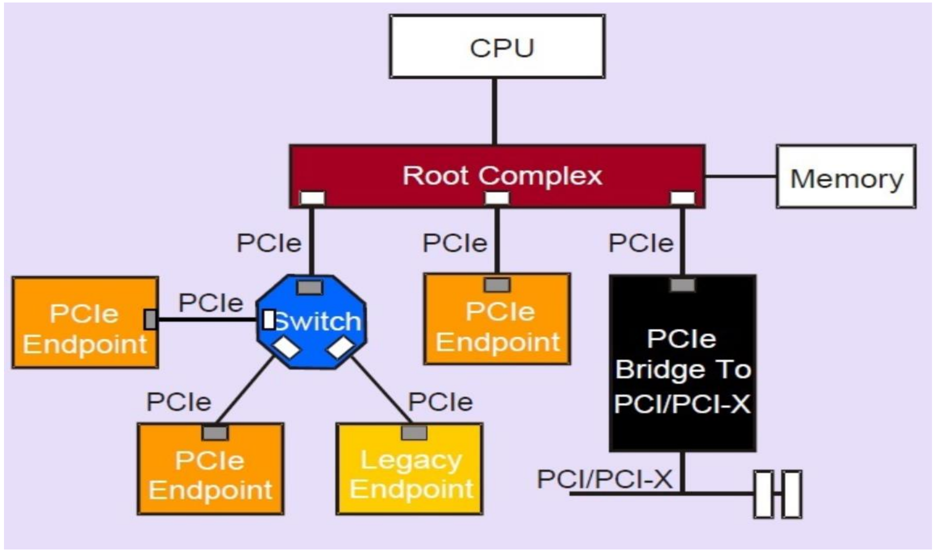


- The PCI Express standard defines link widths of x1, x2, x4, x8, x12, x16 and x32.
- PCI Express preserves backward compatibility with PCI; legacy PCI system software can detect and configure newer PCI Express devices without explicit support for the PCI Express standard.

PCI Architecture & Topology

PCIe Components:

- Root Complex
- End Points
- PCIe to PCI(X) Bridge
- Requester
- Completer
- Port
- Switch



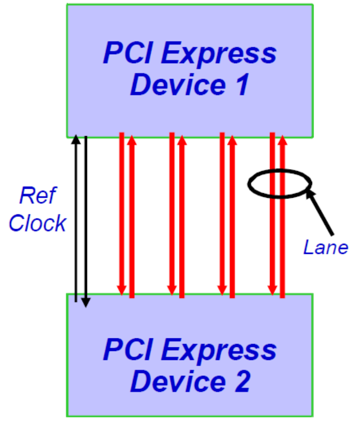
1. Root Complex is interface between the CPU and the PCIe buses may contain several components.
2. End Pointes can be Wire, Wireless cards, FPGAs.
3. Requester is who initiate the commands (read, write).
4. Complete is who accepts the commands and process that.
5. Ports where endpoints/switches connected
6. Switches are used to connect the multiple endpoints

altran Part of Capgemini

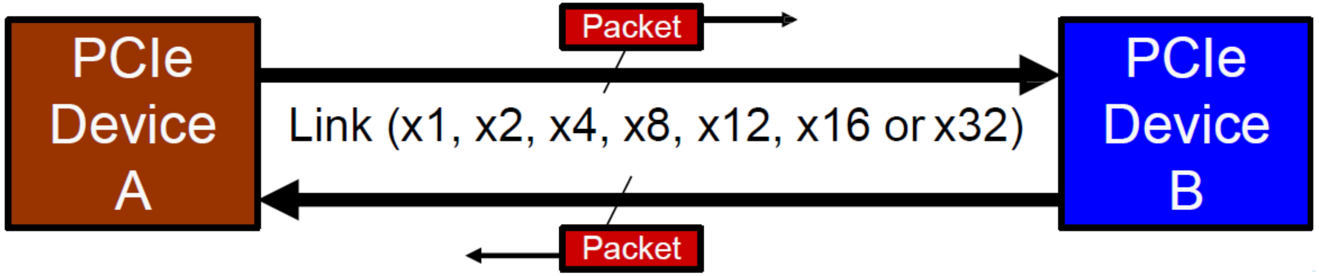
PCIe Basics and Concepts

Dual Simplex Point to Point serial connection

- Serial, point-to-point, Low Voltage diff signals
- Independent transmit and receive sides
- Scalable Link Widths (x1,x2,x4,x8,x12,x16,x32)
- Scalable Link Speeds (2.5 to 32 GT/s [64 GT/s for PCIe 6.0])
- Packet based transaction protocol
- Virtual Channels, Traffic classes
- End to end CRC
- Flow Control



x4 Link Example



PCI Limitation

- Limited host pin-count.
- PCI(X) uses a shared parallel bus architecture, in which the PCI host and all devices share a common set of address, data and control lines, So it is limited to one master at a time.
- Parallel bus required more wires between host and endpoint where PCIe uses Serial bus required less wires.

- Data Rate : PCI
  PCI   133 MB/s (32-bit at 33 MHz)
  PCI   266 MB/s (32-bit at 66 MHz)
  PCI-X 266 MB/s (64-bit at 33 MHz)
  PCI-X 533 MB/s (64-bit at 66 MHz)

PCIe Performance

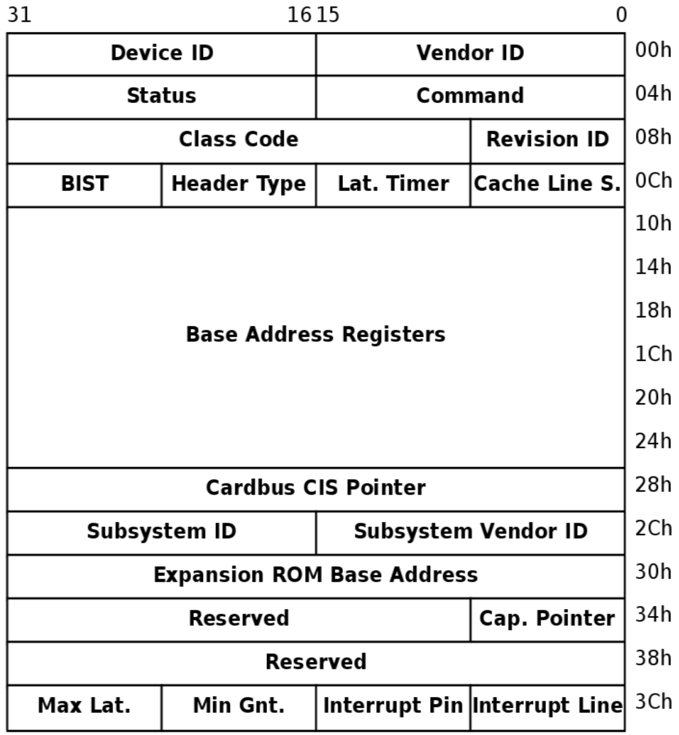
PCIe Bandwidth:

Data Rate (GB/s) aprx	**Link Width**				
	x1	**x2**	**x4**	**x8**	**x16**
PCIe 1.x	0.25	0.5	1	2	4
PCIe 2.x	0.5	1	2	4	8
PCIe 3.0	1	2	4	8	16
PCIe 4.0	2	4	8	16	32
PCIe 5.0	4	8	16	32	64
PCIe 6.0	8	16	32	64	128

PCI Configuration Address Space

- Contains 256 or 4K bytes of basic device information.
- The first 64 bytes (00h –3Fh) make up the standard configuration header, including PCI ID, i.e. vendor ID and device ID registers, to identify the device.
- The remaining 192 bytes (40h –FFh) represent user-definable configuration space.

Vendor ID:
This 16 bit value identifies the manufacturer of the function.
Device ID:
This 16 bit value is assigned by manufacturer to identify the type of the function.



31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Lat. Timer	Cache Line S.	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			Cap. Pointer	34h
Reserved				38h
Max Lat.	Min Gnt.	Interrupt Pin	Interrupt Line	3Ch

PCI Configuration Address Space

Revision ID:

This 8 bit value is assigned by manufacturer to identify the revision of the function.

Class Code:

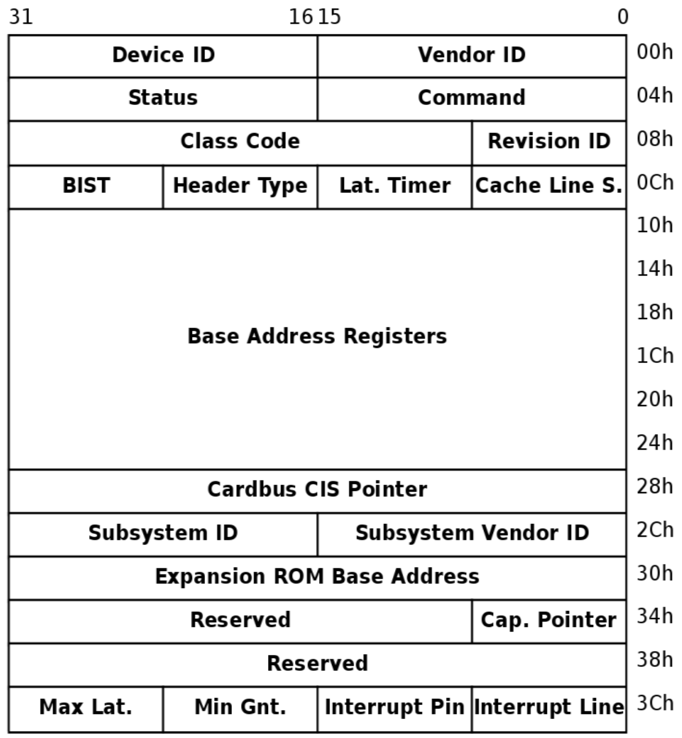
Defines the basic function of the functions

01	Mass storage controller
02	Network controller
03	Display controller (etc)

Base Address Registers:

These registers are used to determine and allocate the type, amount and location of PCI I/O and PCI memory space that the device can use.

The **Subsystem ID** and the **Subsystem Vendor ID** differentiate specific model (such as an add-in card). While the Vendor ID is that of the chipset manufacturer, the Subsystem Vendor ID is that of the card manufacturer.

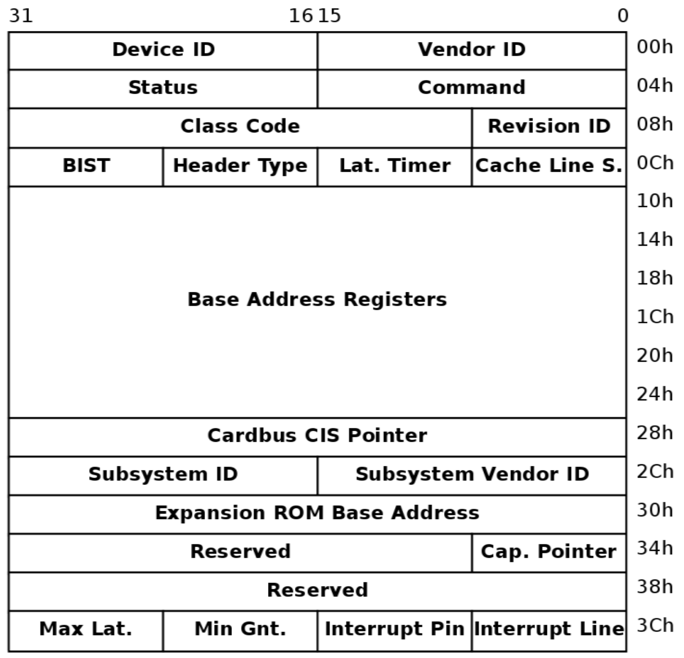


31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Lat. Timer	Cache Line S.	0Ch
Base Address Registers				10h
				14h
				18h
				1Ch
				20h
				24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Cap. Pointer	34h
Reserved				38h
Max Lat.	Min Gnt.	Interrupt Pin	Interrupt Line	3Ch

PCI Configuration Address Space

Header Type:
Type 0 for endpoints
Type 1 for RC, switches and bridges.

- Type 0 configuration requests are sent only to the device for which they are intended.
- Type 1 configuration requests are sent to switches/bridges on the way; the last one before the actual target device will convert it to type 0.
- A device that receives a type 0 request knows that this request is intended for this particular device, so it uses the bus/device number fields in the request to find out what its own bus address is.



31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Lat. Timer	Cache Line S.	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			Cap. Pointer	34h
Reserved				38h
Max Lat.	Min Gnt.	Interrupt Pin	Interrupt Line	3Ch

PCI Config Space Read/Write Functions in Linux

int **pci_read_config_byte**(const struct pci_dev *dev, int addr, u8 *val);

int **pci_read_config_word**(const struct pci_dev *dev, int addr, u16 *val);

int **pci_read_config_dword**(const struct pci_dev *dev, int addr, u32 *val);

int **pci_write_config_byte**(const struct pci_dev *dev, int addr, u8 val);

int **pci_write_config_word**(const struct pci_dev *dev, int addr, u16 val);

int **pci_write_config_dword**(const struct pci_dev *dev, int addr, u32 val);

PCI Config Space Read/write Commands in Linux

Accessing configuration space from user space PCI configuration space access from user space is possible via sysfs.

```
 # lspci | grep VGA
01:00.0 VGA compatible controller: ATI Technologies Inc RV370 [Sapphire X550 Silent]

# hexdump /sys/bus/pci/devices/0000\:00\:00.0/config
0000000 8086 7192 0006 0200 0003 0600 0000 0000
0000010 0000 0000 0000 0000 0000 0000 0000 0000
```

PCI Memory and I/O Spaces

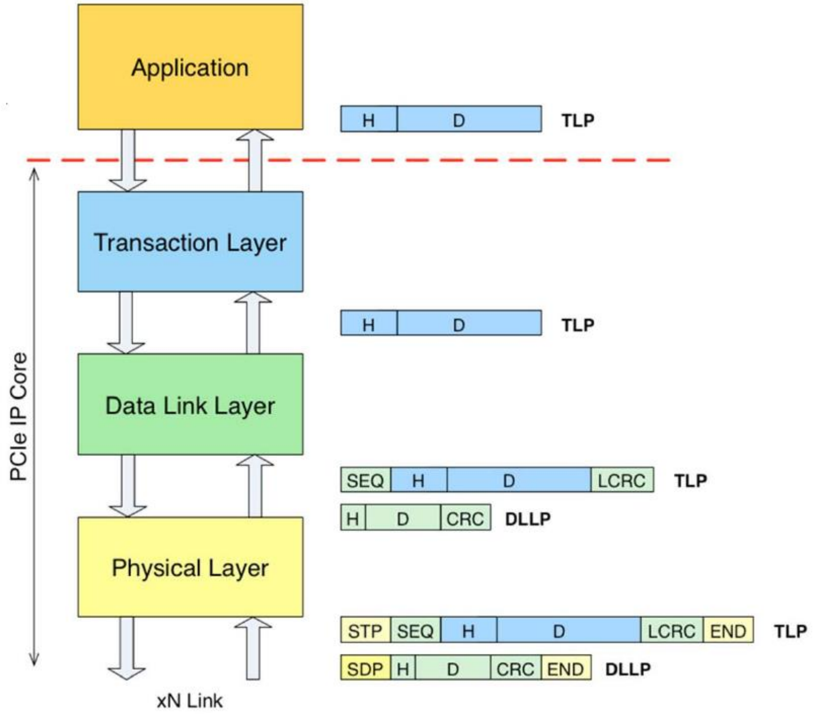
Memory space is used by most everything else – it’s the general-purpose address space

- The PCI spec recommends that a device use memory space, even if it is a peripheral
- An agent can request between 16 bytes and 2GB of memory space. The PCI spec recommends that an agent use at least 4kB of memory space, to reduce the width of the agent’s address decoder
- IO space is where basic PC peripherals (keyboard, serial port, etc.) are mapped
- The PCI spec allows an agent to request 4 bytes to 2GB of I/O space

PCIe Layers

Application Data transferred via packets:
Transaction Layer Packet (TLP)
Data Link Layer Packet (DLLP)

TLP:
Memory Read Unlocked
Memory Read Locked
Memory Write
IO Read/Write
Conf Read
Conf Write
Completion (locked, Unlocked)
Completion with Data (Locked, Unlocked)
Msg Req
Msg Req with Data



DLLP:
TLP ACK/NAK
Power Management
Link Flow Control

altran Part of Capgemini

PCIe Bus Enumeration

- Bus enumeration is performed by attempting to read the vendor ID and device ID register for each combination of bus number and device number at the device's function #0.
- After a new bridge is detected, a new bus number is defined, and device enumeration restarts at device number zero
- If a valid vendor ID is returned from bus 0, device 0, function 0, this indicates that the device is implemented and contains at least one function. Looking for remaining functions (1-7). Proceed to next device.
- If a value of FFFFh were returned as the Vendor ID, this would indicate that function 0 is not implemented in device 0 and the enumeration software (OS/device driver) will proceed for next device.
- OS/device driver software attempts to read the Vendor ID from function 0 in each of the 32 possible devices on bus.
- If the Vendor ID succeeds, Software writes all ones to its BARs and reads back the device's requested memory size and operating system will program the memory-mapped and I/O port addresses into the device's BAR configuration register.

PCIe Interrupts

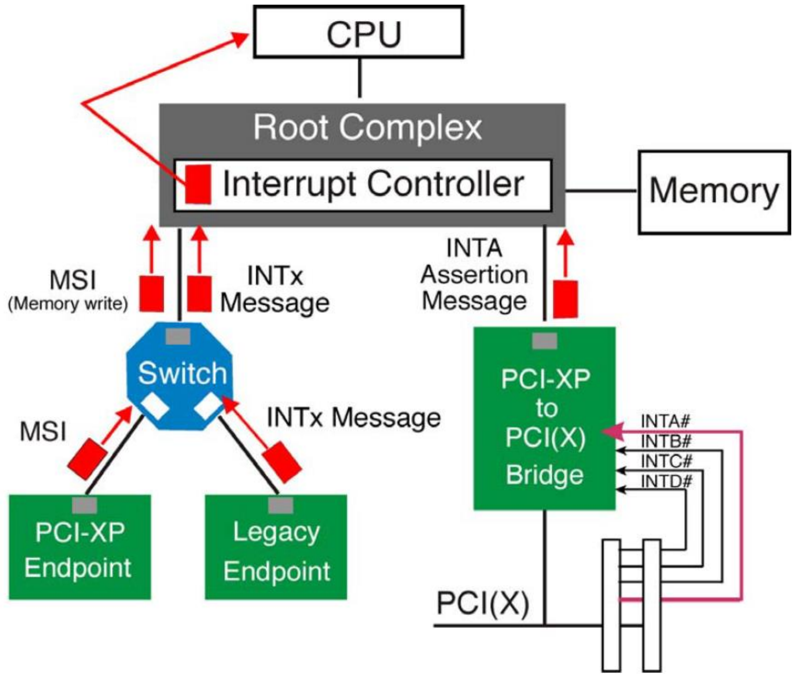
Types of interrupts: Legacy Interrupt, MSI (Message Signaled Interrupts) or MSI-X depending on their design requirements.

Legacy PCI Interrupt Delivery:

- Legacy functions use one of the interrupt lines (INTA#, INTB#, INTC#, INTD#) to signal an interrupt.
- INTx# signal is asserted to request interrupt service.
- PCIe defines in-band messages that act as virtual INTx# wires which target the interrupt controller located typical within the Root Complex.

Limitation of Legacy PCI Interrupts:

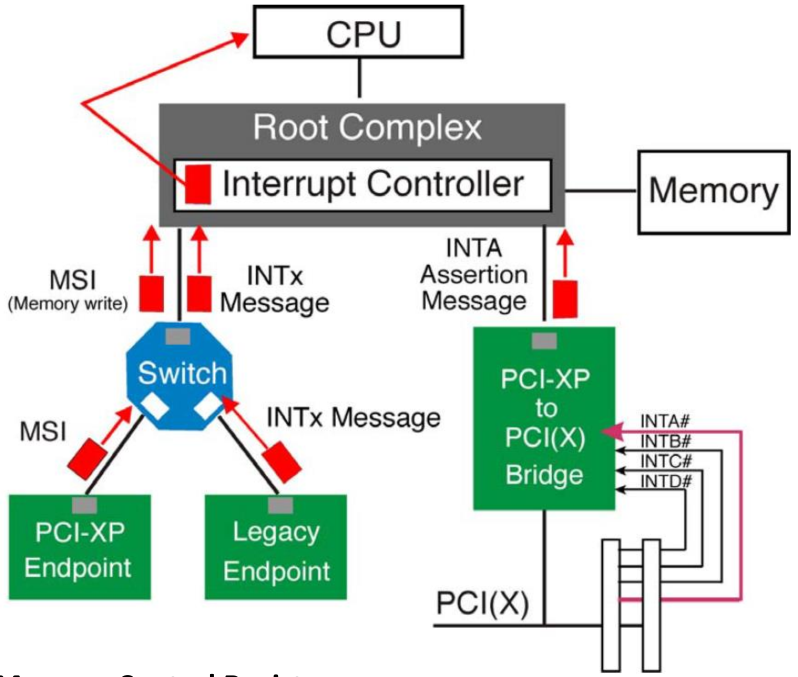
- 4 Interrupt pins needs to shared across all endpoints/bridges/switches. Need to parse the source of the interrupt.



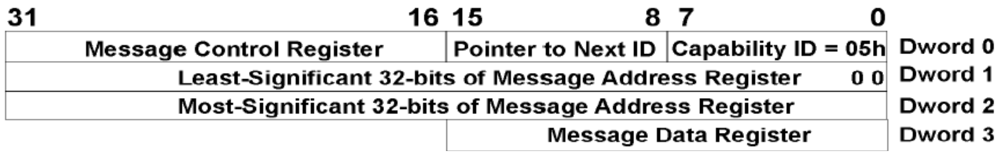
PCIe Interrupts

MSI/MSI-X:

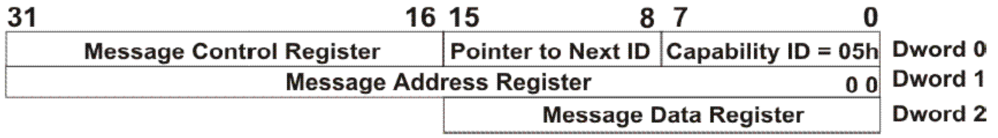
- No need any sideband signals.
- Simple memory Write transactions.
- MSI supports 32 interrupt vectors
- MSI-X supports 2048 interrupt vectors
- MSI can only be distinguished from other memory writes by the address locations they target, which are reserved by the system for interrupt delivery.



MSI Capability Register:

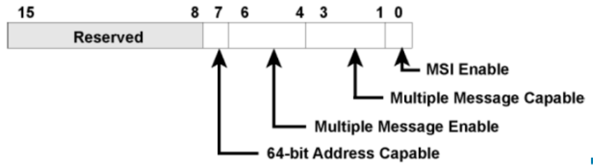


31 … 16	15 … 8	7 … 0	
Message Control Register	Pointer to Next ID	Capability ID = 05h	Dword 0
Least-Significant 32-bits of Message Address Register		0 0	Dword 1
Most-Significant 32-bits of Message Address Register			Dword 2
	Message Data Register		Dword 3



31 … 16	15 … 8	7 … 0	
Message Control Register	Pointer to Next ID	Capability ID = 05h	Dword 0
Message Address Register		0 0	Dword 1
	Message Data Register		Dword 2

Message Control Register



PCI Linux commands

# lspci - display all the device information

- 00:00.0 Host bridge: Intel Corporation 5500 I/O Hub to ESI Port (rev 13)
- 00:01.0 PCI bridge: Intel Corporation 5520/5500/X58 I/O Hub PCI Express Root Port 1 (rev 13)
- 00:09.0 PCI bridge: Intel Corporation 7500/5520/5500/X58 I/O Hub PCI Express Root Port 9 (rev 13)
- 00:14.0 PIC: Intel Corporation 7500/5520/5500/X58 I/O Hub System Management Registers (rev 13)

# lspci -t

```
-[0000:00]-+-00.0
           +-01.0-[01]--+-00.0
           |            \-00.1
           +-03.0-[02]--+-00.0
           |            \-00.1
           +-07.0-[04]
```

PCI Linux commands

```
# lspci -v
03:00.0 RAID bus controller: LSI Logic / Symbios Logic MegaRAID SAS 2108 [Liberator] (rev 05)
        Subsystem: Dell PERC H700 Integrated
        Flags: bus master, fast devsel, latency 0, IRQ 16
        I/O ports at fc00 [size=256]
        Memory at df1bc000 (64-bit, non-prefetchable) [size=16K]
        Memory at df1c0000 (64-bit, non-prefetchable) [size=256K]
        Expansion ROM at df100000 [disabled] [size=256K]
        Capabilities: [50] Power Management version 3
        Capabilities: [68] Express Endpoint, MSI 00
        Capabilities: [d0] Vital Product Data
        Capabilities: [a8] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [c0] MSI-X: Enable+ Count=15 Masked-
        Capabilities: [100] Advanced Error Reporting
        Kernel driver in use: megaraid_sas
        Kernel modules: megaraid_sas
```

PCI Linux commands

# setpci

A utility for querying and configuring PCI devices

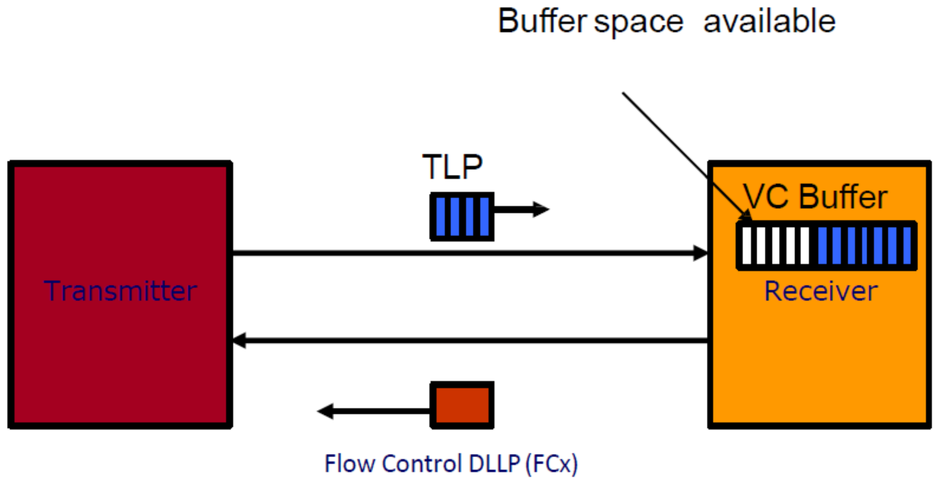
"**setpci -s 12:3.4 3c.l=1,2,3**" writes longword 1 to register 3c, 2 to register 3d and 3 to register 3e of device at bus 12, slot 3, function 4

# setpci --dumpregs

```
cap pos w name
  00 W VENDOR_ID
  02 W DEVICE_ID
  04 W COMMAND
  06 W STATUS
  08 B REVISION (etc)
```

PCIe Flow Control

Flow control guarantees that transmitters will never send Transaction Layer Packets (TLPs) that the receiver can't accept. This prevents receive buffer over-runs and eliminates the need for inefficient disconnects, retries, and wait-states on the link. Receiver sends Flow Control Packets (FCP) which are a type of DLLP (Data Link Layer Packet) to provide the transmitter with credits so that it can transmit packets to the receiver.



Linux PCI Driver Example

```
static struct pci_device_id ids[] =
{
		{ PCI_DEVICE(VENDOR_ID, DEVICE_ID) }
};

static struct pci_driver pci_drv =
{
		.name = DEVICE_NAME,
		.id_table = ids,
		.probe = device_probe,
		.remove = device_remove,
};
```

Linux PCI Driver Example

```
static int __init device_init(void)
{
        int status = 0;

        //Register PCI device. This will match the Device Vendor ID from table and call the probe function
        status = pci_register_driver(&pci_drv);
        if (status) {
                printk("PCI driver failed to register, rval = %d\n", status);
        }
        return status;
}

static void __exit device_exit(void)
{
        pci_unregister_driver(&pci_drv);
        return;
}
```

Linux PCI Driver Example

```
static int device_probe(struct pci_dev *dev, const struct pci_device_id *id)
{
        unsigned int base0;
        unsigned long  memflag = 0;
        unsigned long  memlen = 0;
        unsigned char *bar0_addr;

        //Read PCI BAR0 address (Shared mem address)
        pci_read_config_dword(dev, PCI_BASE_ADDRESS_0, &base0);

        //Returns bus start address for a given PCI region
        memstart = pci_resource_start(dev, BAR0);
        //Returns the byte length of a PCI region
        memlen = pci_resource_len(dev, BAR0);

        //check memlen before requesting the memory region
        if (request_mem_region(memstart, memlen, dev->dev.kobj.name) == NULL)
        {
                return -ENOMEM;
        }
```

Linux PCI Driver Example

```
		//Enable PCI device
		pci_enable_device(dev);

		//Get the virtual address of the BAR0
		bar0_addr = (unsigned char *) ioremap(base0, memlen);

		return 0;
}

static void device_remove(struct pci_dev *dev)
{
		//Disable PCI device
		pci_disable_device(dev);
}
```

PCI Applications

PCI Express operates in consumer, server, and industrial applications, as a motherboard-level interconnect (to link motherboard-mounted peripherals), a passive backplane interconnect and as an expansion card interface for add-in boards.

External GPUs

- 3D Graphic Cards
- Soundcards

Network

- 10G or multigigabit cards
- 802.11ax WiFi cards

Enterprise Storage

- RAID Cards
- SSD drives
- Flash memories

PCIe 6.0

Planned to be released in 2021

- Bandwidth is expected to increase to 64 GT/s, yielding 126 GB/s in each direction and 256GB/s(bidirectional) in a 16-lane configuration
- 4-levelpulse-amplitude modulation(PAM-4) signaling will be used
- Reed—Solomon forward error correction(RS-FEC), a predictive error correction.
- Maintains backward compatibility to previous generations
- PCI-Express 6.0 is also likely to be useful in self-driving systems, the industrial IoT, and any system that juggles multiple sensors to combine input from multiple peripherals into a cohesive whole

REVISION HISTORY

Rev No.	Date	Description of Change	Author	Review and Approved By
1.0	03-12-2020	Initial Draft	Rajeshkumar R	Suresh K

altran

Part of Capgemini