Cheng-Chun Tu

Interrupts Delivery in a Multi-host Environment

V1. Sep 24, 2012

V2. Sep 25, 2012

Intro: Problems of Pin-based Interrupt

- Shared interrupt
  - Usually shared between multiple devices, depends on kernel interrupt handler to associate the destined device
  - Poor performance
- Out-of-order
  - When device writes to memory and rises an interrupt, it's possible that the interrupt arrives before all data has arrived in memory
- Single interrupt per function
  - You have only one pin

http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/PCI/MSI-HOWTO.txt;hb=HEAD

Message Signaled Interrupt

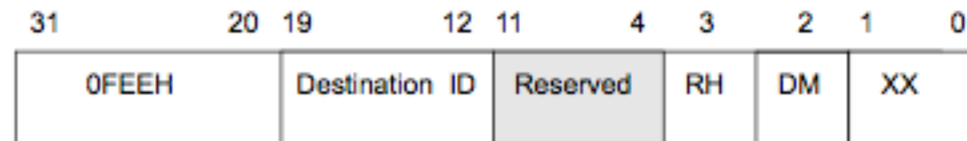
- MSIs are never shared among devices
  - Don't bother the kernel to check and multiplex
- Ordered delivery
  - PCI transaction ordering: Interrupt generating write comes after the data writes
- Multiple interrupt per device
  - Each interrupt is specialized to different purposes
  - A VF has three interrupt: RX / TX / MSG
- MSI v.s MSI-X
  - MSI: address + data (multiple data value creates multiple int)
  - MSI-X: multiple address + data (up to 2048 INTs per device)

Everything becomes read/write

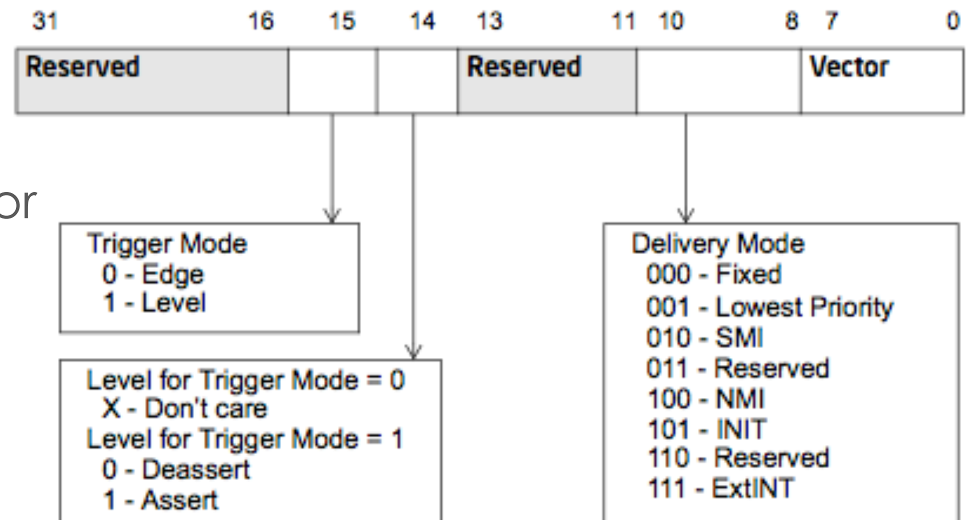
- With MSI/MSI-X, everything in PCIe boils down to PCIe read/write
- A device
  - Signals interrupt to its host using MSI address (write from the bus to the MSI area, interpreted by the chipset.)
  - DMA read/write data to host's memory
- A host
  - Read/write its memory
  - Configure its devices using memory-mapped IO

MSI/MSI-X Format

- Address:
  - Address recognized by chipset, start with 0xFEE (Local APIC)
  - Contains fields:
    - destination CPU ID
    - redirection info.



- Data:
  - Contains field:
    - Vector: interrupt vector associated with the message
    - Delivery mode
    - Trigger mode



MSI/MSI-X in Device and Kernel

- A PCI device keeps an MSI-X table in its HW's register

DWord 3	DWord 2	DWord 1	DWord 0	Entry
Vector Control	Message Data	Msg Upper Addr	Message Address	0
Vector Control	Message Data	Msg Upper Addr	Message Address	1
Vector Control	Message Data	Msg Upper Addr	Message Address	n

Vector Control is only for mask bit (enabled/disabled)

- Device driver registers INT to OS, an 1:1 vector-to-entry mapping is constructed
  - struct msix_entry {

    u16 vector; /* kernel uses to write allocate vector */

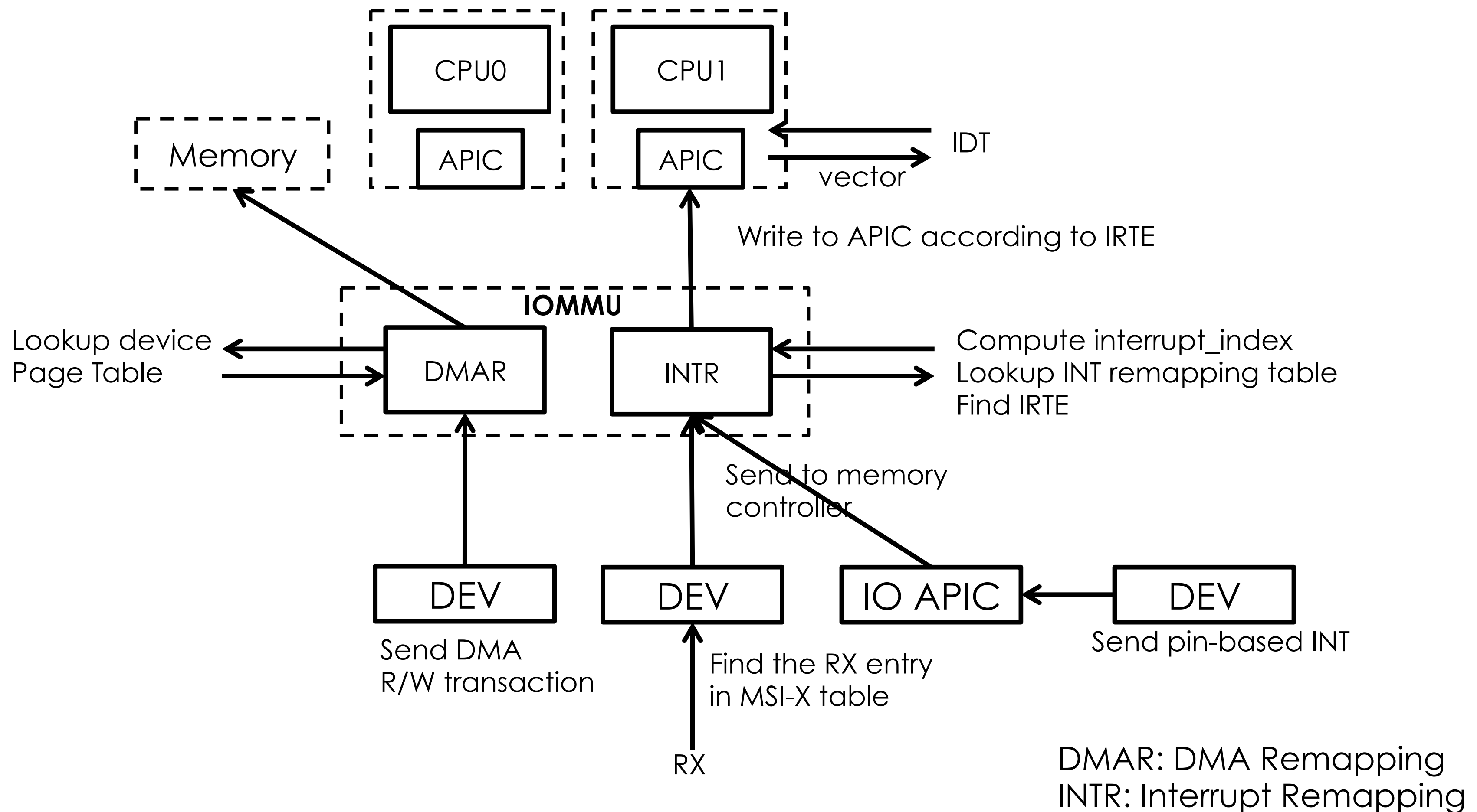
    u16 entry; /* device driver uses to specify entry in HW */ }
  - Driver specifies entry number in its HW MSI-X table
  - Kernel assigns vector number

http://www.mjmwired.net/kernel/Documentation/MSI-HOWTO.txt

IOMMU Interrupt Remapping

- Interrupt-remapping enables system software to control and censor external interrupt generated by
  - Interrupt controllers (I/OxAPICs),
  - MSI/MSI-X capable devices including endpoints,
- INT remapping requests
  - From MSI addr/data sent from devices and I/O APIC, compute the interrupt_index (slide 20, 21)
  - Lookup the IRTE in the remapping table using interrupt_index
- IRTE (Interrupt Remapping Table Entry) (Slide 22)
  - Destination ID: specify interrupt's target processor(s)
  - Vector: interrupt vector number
  - Other fields see spec

Put together



Example of RX interrupt

At IOMMU, IRTE at index 40 contains
Destination ID = CPU0, Vector = 65
Write to CPU0's APIC, CPU finds RX handler at index 40 in its IDT

An incoming packet triggers RX
Device write (0xfee00518, 0)
Compute Interrupt index = addr.handle + data.subhandle = 40

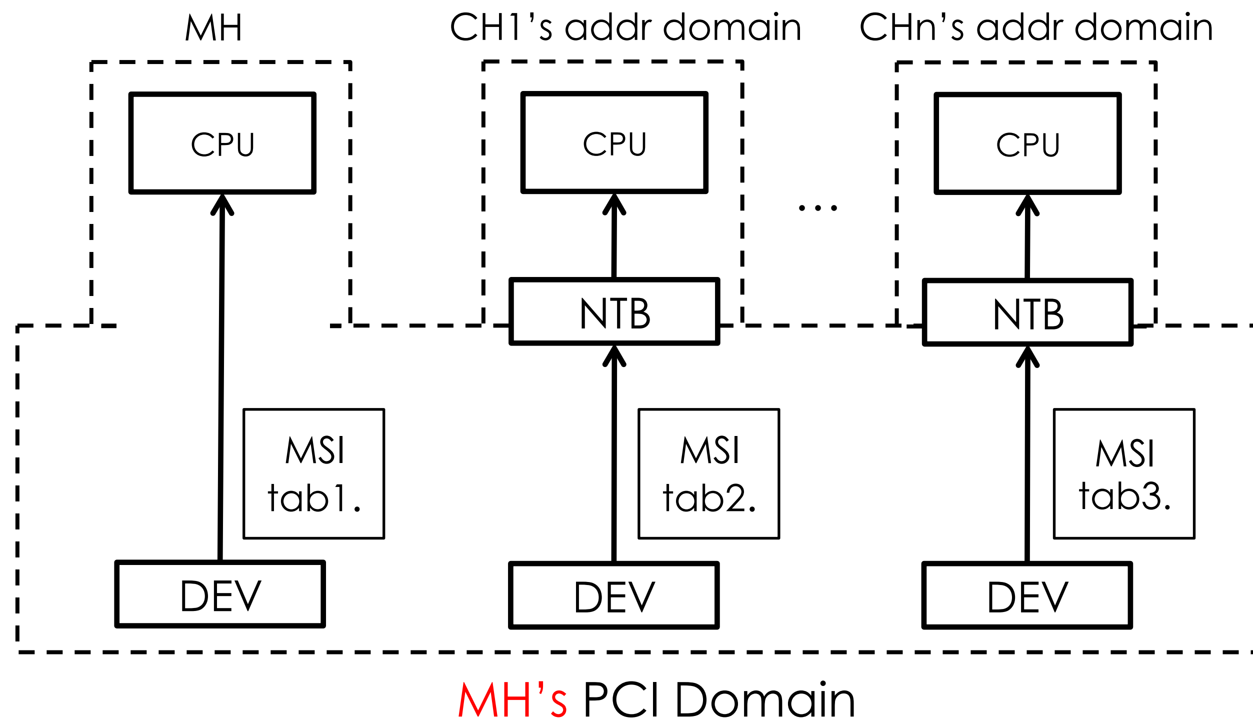
Kernel creates 1:1 vector-to-entry mapping
(65, 0), (66, 1), (67, 2), where vector table contains INT handler's code
65: RX handler, 66: TX handler, 67: Msg handler

DeviceA has 3 MSI-X interrupt (RX/TX/MSG):
MSI-X table (addr, data) has three entries
(0xfee00518, 0), (0xfee00518, 1), (0xfee00518, 2) at entry 0,1,2

Requirements

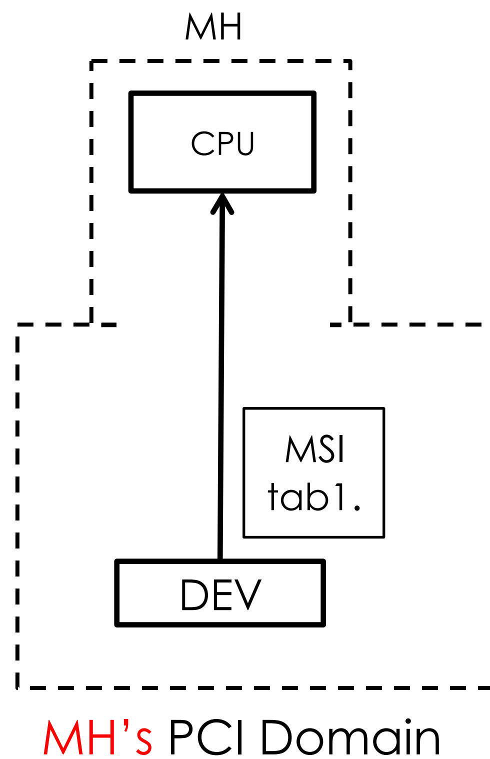
- Assumption:
  - The device, could be VF or legacy PCI device, belongs to the MH's PCI hierarchy
- Requirement1:
  - The device is able to directly send interrupts to its assigned host behind NTB.
- Requirement2:
  - The device assigned to a host and further directly pass-through toa VM can directly send interrupts to the VM's kernel
- Requirement3:
  - The device's INT could be forwarded to the MH for optimization

R1: Cross-domain Interrupt delivery



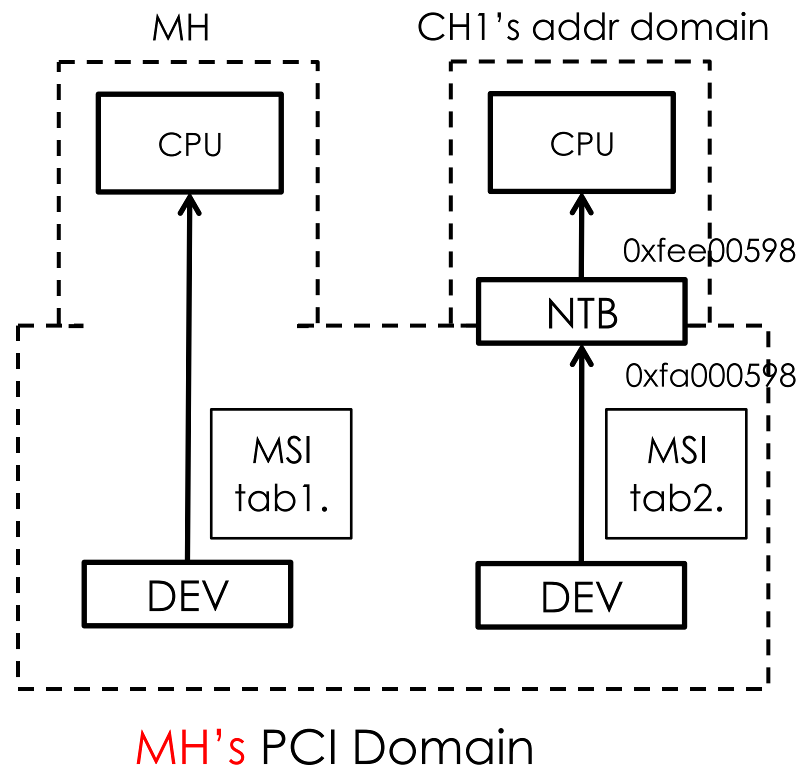
For each pair, MH sets-up the NTB mapping and address in MSI-X table.

Example 0: single host



- MSI table1:
  - RX:
    - Address: 0xfee00518
    - Data: 0x0
  - TX:
    - Address: 0xfee00598
    - Data: 0x1

Example: Multi-host



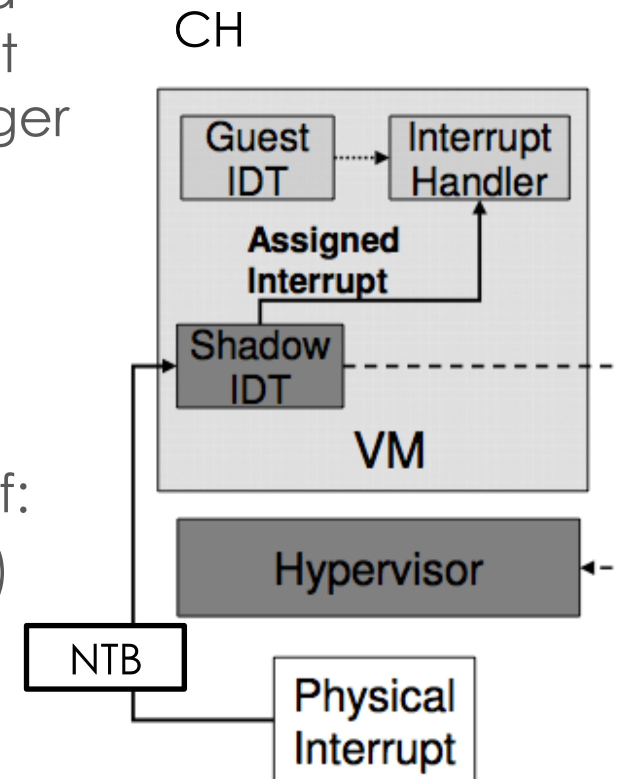
- MH setup NTB mapping and MSI-X tab2
- MSI table2:
  - RX:
    - Address: 0xfa000518
    - Data: 0x0
  - TX:
    - Address: 0xfa000598
    - Data: 0x1
- NTB mapping:
  - 0xfa000000 -> 0xfee00000

R2: Direct delivery to VM

- When a VM is scheduled to run on a core, the direct-assigned device's interrupt goes to the core directly
- If the VM is not running, send the device's interrupt to VMM (Fall back to standard operation)

Implementation

- When a VM X is scheduled on a core, a shadow IDT is set up for the core so that an interrupt not meant for X would trigger a VMexit and invoke the hypervisor
  - Different VMs on a PM are assigned different interrupt numbers
- If a VM is scheduled on a core Y, setup the VM device's INT to coreY, by one of:
  - Configure the MSI-X address field (dst ID)
  - Configure the INT remapping table in IOMMU



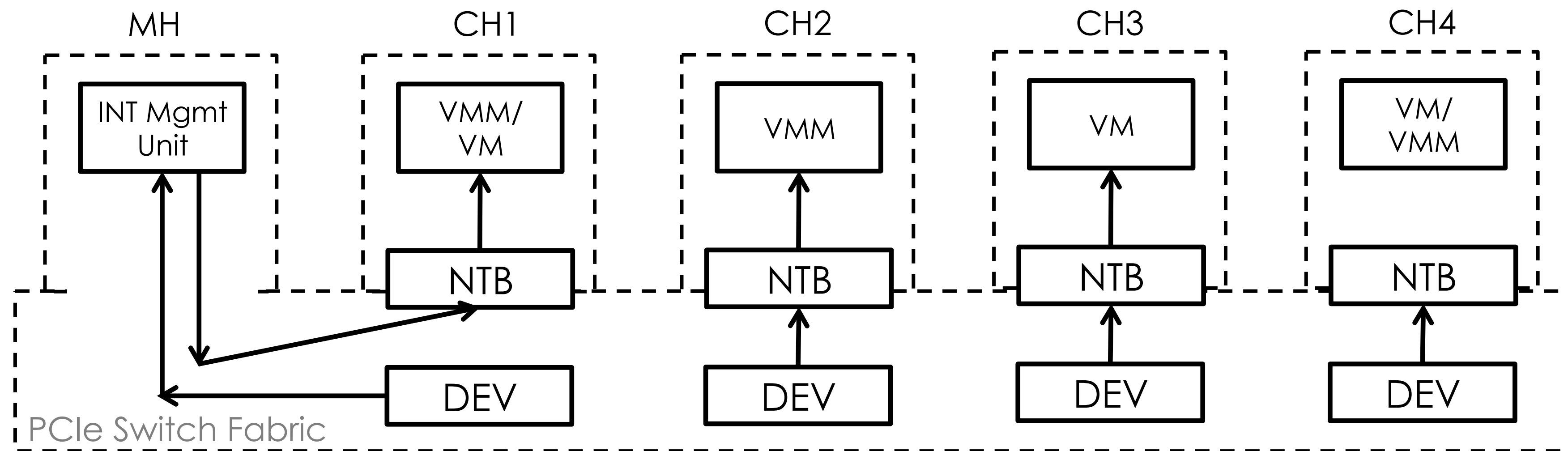
Example

- Suppose VM1 is meant to handle INT3, VM2 is meant to handle INT7 and both are supposed to run on Core2
- Shadow IDT:
  - Only the entry that guest directly handles has Present bit = 1
  - The rests are 0, causing VMexit
- Case1: VM1 runs on core2
  - INT3 entry present = 1, the rests are 0
- Case2: VM2 runs on core2
  - INT7 entry present = 1, the rests are 0
- Case3: VM3 runs on core2
  - All entries' present bit are 0

R3: Interrupt Management

- Let MH centrally schedules all or parts of the interrupts
  - Prevent livelock, reduce CH's loading
  - coalesce bunch of Interrupts and deliver once
- MH is able to interrupt CH's kernel or VM on CH
  - By writing the NTB mapped address and data at Link side
  - The virtual side translates to legitimate MSI-X
- Example:
  - Let DEV3 for CH3 write its MSI to MH (with msiaddr 0xFEE00598)
  - INT handler at MH's core1 receives it and keeps a counter
  - If counter > threshold, MH sends a write to 0xFA000598, which maps to CH3's 0xFEE00598, a real interrupt message at CH3

Summary of Interrupt Delivery



Case1.
Forward interrupt to MH
- Livelock avoidance
- Interrupt coalescing

Case2.
Interrupt Delivery to VMM

Case3.
Direct Interrupt Delivery to VM

Case4.
Disable Interrupt, Enable polling

End

Example

- DeviceX has 3 MSI-X interrupt (RX/TX/MSG):
  - In device A's MSI-X table (addr, data), it has three entries
  - (0xfee00518, 0), (0xfee00518, 1), (0xfee00518, 2) at entry 0,1,2
- Kernel creates 1:1 vector-to-entry mapping
  - (65, 0), (66, 1), (67, 2), where vector table contains INT handler's code
  - 65: RX handler, 66: TX handler, 67: Msg handler
- An incoming packet triggers RX
  - Device write (0xfee00518, 0)
  - addr.handle: [19:5], data.subhandle: [15:0] bit
  - Computed Interrupt index = addr.handle + data.subhandle = 40
- At IOMMU, IRTE at index 40 contains
  - Destination ID = CPU0
  - Vector = 65
  - Write to CPU0's APIC, CPU finds RX handler at index 40 in its IDT

INTR: Interrupt remapping request fmt

To determine the interrupt_index:

```
if (address.SHV == 0) {
        interrupt_index = address.handle;
} else {
        interrupt_index = (address.handle + data.subhandle);
}
```

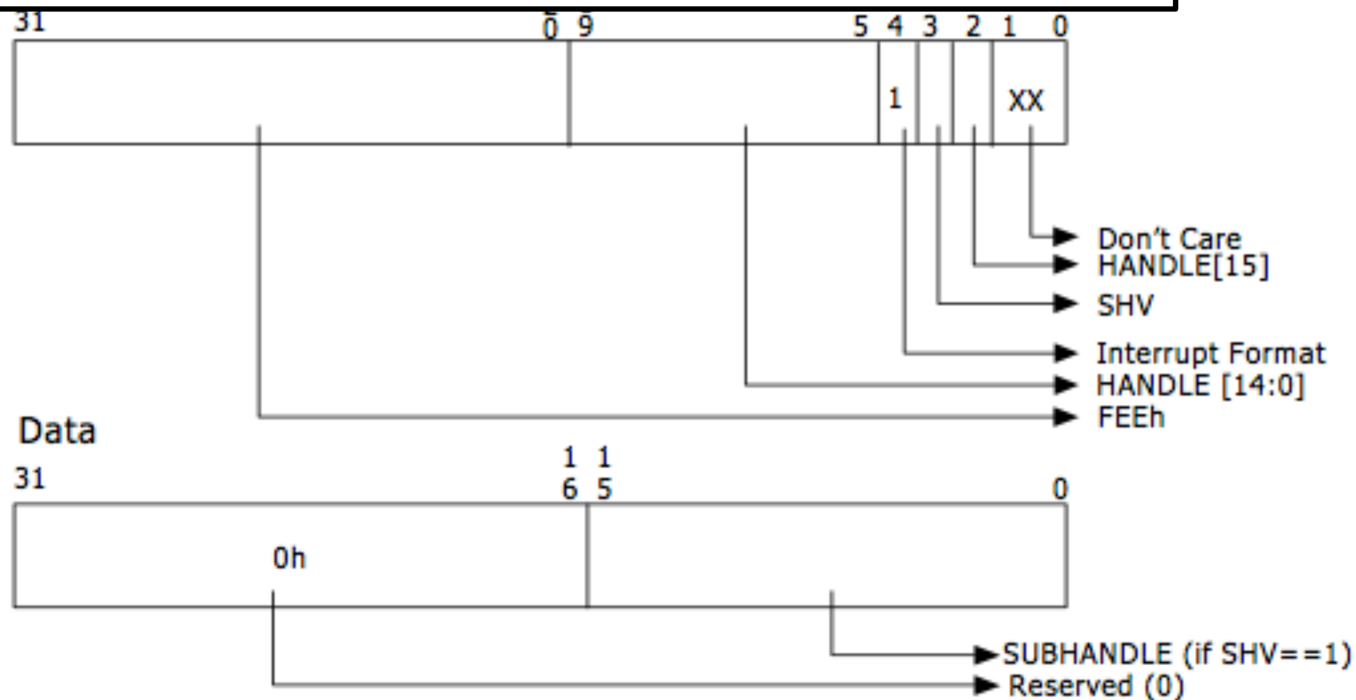


Figure 5-11. Remappable Format Interrupt Request

MSI-X addr. supports remapping

The field Interrupt_Index [14:0] indicate the IRTE

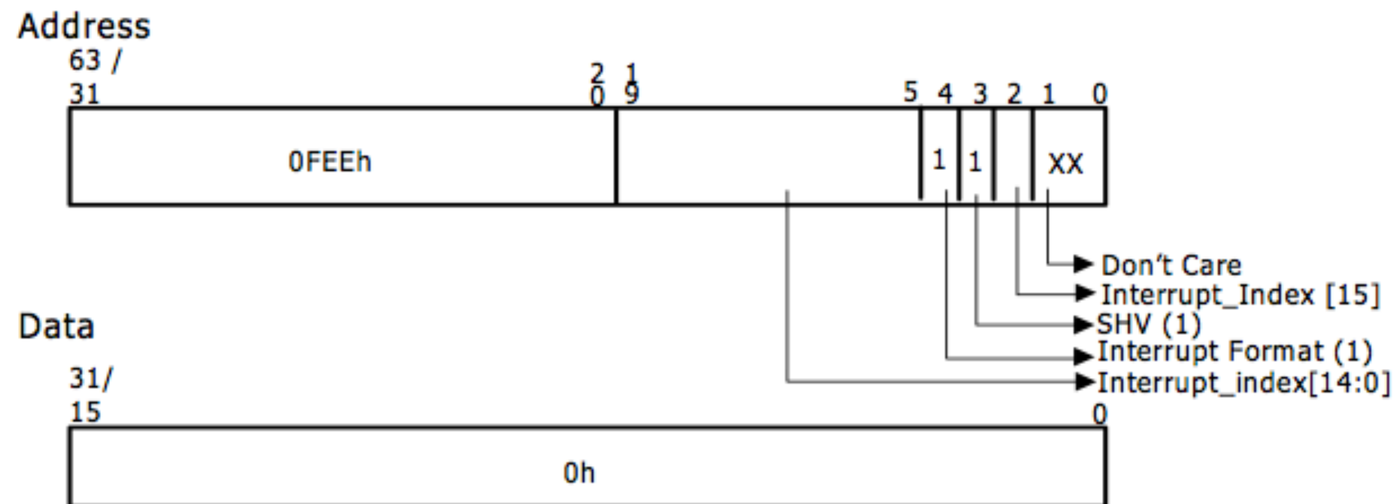


Figure 5-14. MSI-X Programming

INTR: IRTE

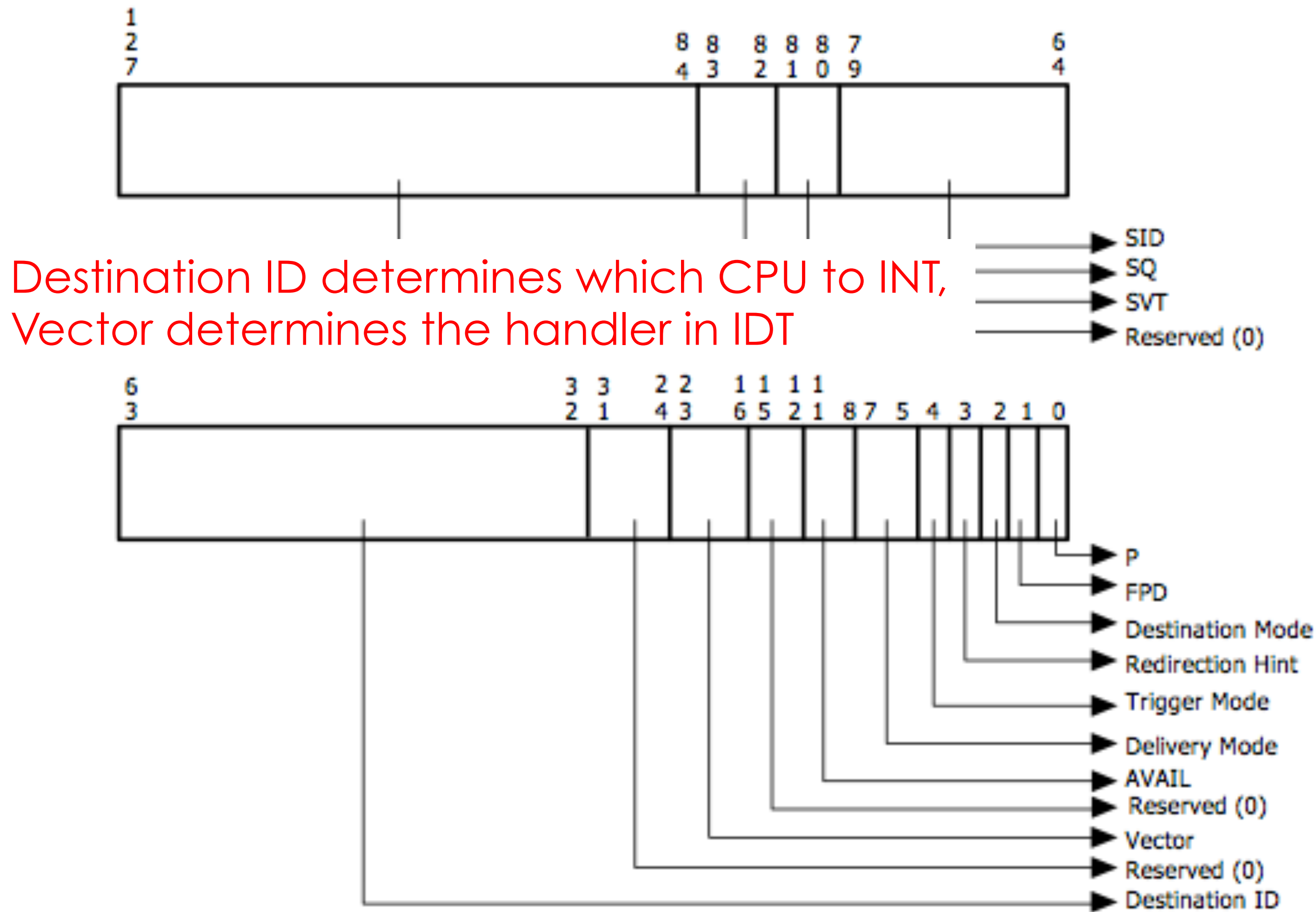


Figure 9-28. Interrupt Remapping Table Entry Format

MSI-X Address

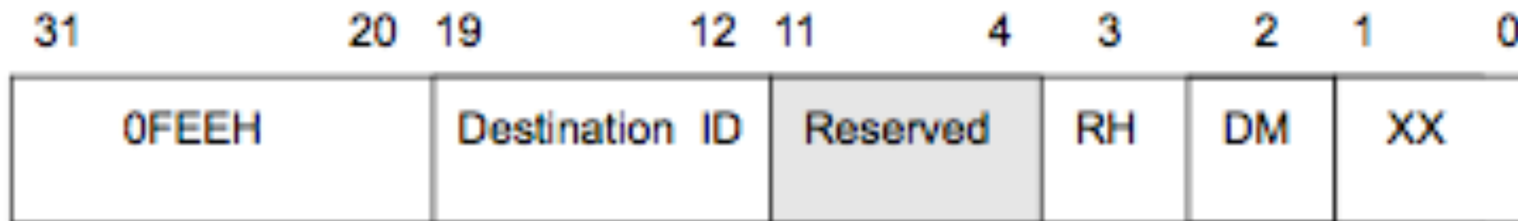


Figure 10-24. Layout of the MSI Message Address Register

MSI-X data

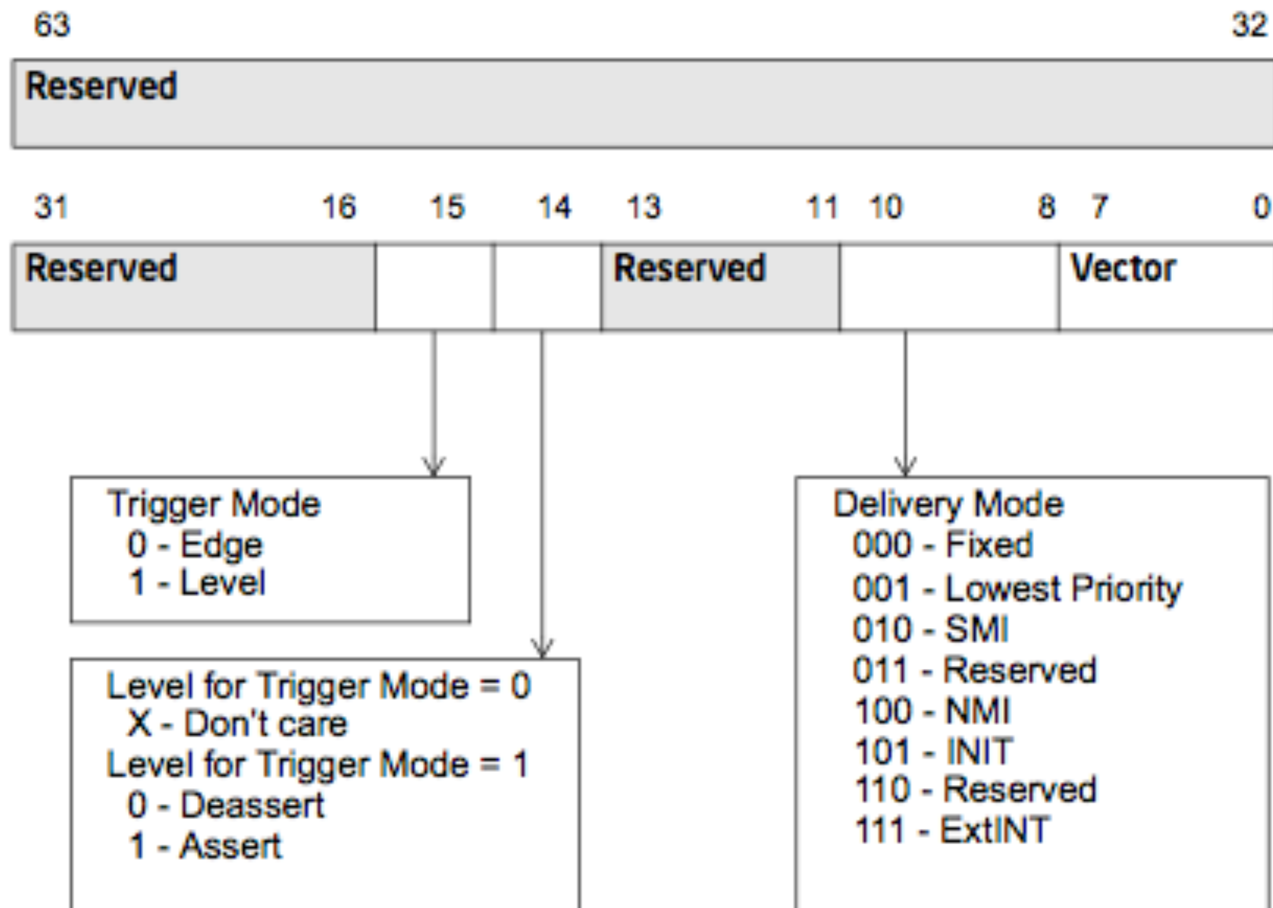


Table 9.1 PCI Configuration Space

Cheng-Chun Tu

Section	Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0
Mandatory PCI Register	0x0	Device ID		Vendor ID	
	0x4	Status Register		Control Register	
	0x8	Class Code (0x020000/0x010000)			Revision ID
	0xC	Reserved	Header Type (0x0/ 0x80)	Latency Timer	Cache Line Size (0x10)
	0x10	Base Address Register 0			
	0x14	Base Address Register 1			
	0x18	Base Address Register 2			
	0x1C	Base Address Register 3			
	0x20	Base Address Register 4			
	0x24	Base Address Register 5			
	0x28	CardBus CIS pointer (0x0000)			
	0x2C	Subsystem ID		Subsystem Vendor ID	
	0x30	Expansion ROM Base Address			
	0x34	Reserved			Cap Ptr (0x40)
	0x38	Reserved			
	0x3C	Max Latency (0x00)	Min Grant (0x00)	Interrupt Pin (0x01...0x04)	Interrupt Line (0x00)
PCI / PCIe Capabilities	0x40...0x47	Power management capability			
	0x50...0x67	MSI Capability			
	0x70...0x7B	MSI-X Capability			
	0xA0...0xDB	PCIe Capability			
	0xE0...0xE7	VPD Capability			
Extended PCIe Configuration	0x100...0x12B	AER Capability			
	0z140...0x14B	Serial ID Capability			
	0x150...0x157	ARI Capability			
	0x160...0x19C	SR-IOV Capability			

System Address Map

1_0000_0000
High BIOS, Optional extended SMRAM
FF00_0000
Hub Interface_A (always)
FEF0_0000
Local APIC Space
FEE0_0000
Hub Interface_A (always)
FED0_0000
Hub Interface_B–D, I/O APIC Space
FEC8_0000
Hub interface_A, I/O APIC Space
FEC0_0000

Example

- DeviceX has 3 MSI-X interrupt (RX/TX/MSG):
  - In device A’s MSI-X table (addr, data), it has three entries
  - (0xfee00518, 0), (0xfee00518, 1), (0xfee00518, 2) at entry 0,1,2
- Kernel creates 1:1 vector-to-entry mapping
  - (65, 0), (66, 1), (67, 2), where vector table contains INT handler’s code
  - 65: RX handler, 66: TX handler, 67: Msg handler
- An incoming packet triggers RX
  - Device write (0xfee00518, 0)
  - addr.handle: [19:5], data.subhandle: [15:0] bit
  - Computed Interrupt index = addr.handle + data.subhandle = 40
- At IOMMU, IRTE at index 40 contains
  - Destination ID = CPU0
  - Vector = 65
  - Write to CPU0’s APIC, CPU finds RX handler at index 40 in its IDT

References

- http://forum.osdev.org/viewtopic.php?f=1&t=24813&p=204113&hilit=William#p204113