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

<table>
<thead>
<tr>
<th>DWord 3</th>
<th>DWord 2</th>
<th>DWord 1</th>
<th>DWord 0</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Control</td>
<td>Message Data</td>
<td>Msg Upper Addr</td>
<td>Message Address</td>
<td>0</td>
</tr>
<tr>
<td>Vector Control</td>
<td>Message Data</td>
<td>Msg Upper Addr</td>
<td>Message Address</td>
<td>1</td>
</tr>
<tr>
<td>Vector Control</td>
<td>Message Data</td>
<td>Msg Upper Addr</td>
<td>Message Address</td>
<td>n</td>
</tr>
</tbody>
</table>

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

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

  ```c
  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
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
DMAR: DMA Remapping
INTR: Interrupt Remapping

Memory

APIC

CPU0

CPU1

DMAR

INTR

IOMMU

DEV

APIC

IO APIC

Send to memory controller

Find the RX entry in MSI-X table

Send pin-based INT

Write to APIC according to IRTE

IDT vector

lookup device
Page Table

Send DMA R/W transaction

Find IRTE

Compute interrupt_index

lookup INT remapping table

DMAR: DMA Remapping

INTR: Interrupt Remapping

Send to memory controller

Find the RX entry in MSI-X table

DMAR: DMA Remapping

INTR: Interrupt Remapping
Example of RX interrupt

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

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

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
Requirements

- Assumption:
  - The device, could be VF or legacy PCI device, belongs to the MH’s PCI hierarchy

- Requirement 1:
  - The device is able to directly send interrupts to its assigned host behind NTB.

- Requirement 2:
  - The device assigned to a host and further directly pass-through to a VM can directly send interrupts to the VM’s kernel

- Requirement 3:
  - 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

MH's PCI Domain
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).
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
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 0xFE00598)
- 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 0xFE00598, a real interrupt message at CH3
Summary of Interrupt Delivery

Case 1. Forward interrupt to MH
- Livelock avoidance
- Interrupt coalescing

Case 2. Interrupt Delivery to VMM

Case 3. Direct Interrupt Delivery to VM

Case 4. Disable Interrupt, Enable polling
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
To determine the interrupt_index:

```c
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 illustrates the programming of MSI/MSI-X address and data register remapping of the message signalled interrupt.

Figure 5-14. MSI-X Programming
INTR: IRTE

Destination ID determines which CPU to INT, Vector determines the handler in IDT

Figure 9-28. Interrupt Remapping Table Entry Format
MSI-X Address

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

- **Reserved** (bits 63 to 32)
- **Reserved** (bits 31 to 16)
- **Vector** (bits 15 to 0)

**Trigger Mode**
- 0 - Edge
- 1 - Level

**Level for Trigger Mode**
- 0 - Deassert
- 1 - Assert

**Delivery Mode**
- 000 - Fixed
- 001 - Lowest Priority
- 010 - SMI
- 011 - Reserved
- 100 - NMI
- 101 - INIT
- 110 - Reserved
- 111 - ExtiNT
<table>
<thead>
<tr>
<th>Section</th>
<th>Byte Offset</th>
<th>Byte 3</th>
<th>Byte 2</th>
<th>Byte 1</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCI Configuration Space</td>
<td>0x0</td>
<td>Device ID</td>
<td>Vendor ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4</td>
<td>Status Register</td>
<td>Control Register</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x8</td>
<td>Class Code (0x020000/0x010000)</td>
<td>Revision ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0xC</td>
<td>Reserved</td>
<td>Header Type (0x0/0x80)</td>
<td>Latency Timer</td>
<td>Cache Line Size (0x10)</td>
</tr>
<tr>
<td>Mandatory PCI</td>
<td>0x10</td>
<td></td>
<td>Base Address Register 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>0x14</td>
<td></td>
<td>Base Address Register 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x18</td>
<td></td>
<td>Base Address Register 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x1C</td>
<td></td>
<td>Base Address Register 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x20</td>
<td></td>
<td>Base Address Register 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x24</td>
<td></td>
<td>Base Address Register 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x28</td>
<td></td>
<td>CardBus CIS pointer (0x0000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x2C</td>
<td></td>
<td>Subsystem ID</td>
<td>Subsystem Vendor ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x30</td>
<td></td>
<td>Expansion ROM Base Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x34</td>
<td></td>
<td>Reserved</td>
<td>Cap Ptr (0x40)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x38</td>
<td></td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x3C</td>
<td>Max Latency (0x00)</td>
<td>Min Grant (0x00)</td>
<td>Interrupt Pin (0x01...0x04)</td>
<td>Interrupt Line (0x00)</td>
</tr>
<tr>
<td>PCI / PCIe</td>
<td>0x40...0x47</td>
<td>Power management capability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capabilities</td>
<td>0x50...0x67</td>
<td>MSI Capability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x70...0x7B</td>
<td>MSI-X Capability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0xA0...0xDB</td>
<td>PCIe Capability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0xE0...0xE7</td>
<td>VPD Capability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extended PCIe</td>
<td>0x100...0x12B</td>
<td>AER Capability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>0x140...0x14B</td>
<td>Serial ID Capability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x150...0x157</td>
<td>ARI Capability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x160...0x19C</td>
<td>SR-IOV Capability</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
System Address Map

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