VT-d and FreeBSD

Константин Белоусов
kib@freebsd.org

21 сентября 2013 г.

Revision : 1.11
Example PCI Express Topology – Root & Switch
TLP - Transaction Layer Packets

- **I/O**
  - Host access to device (BARs)
  - Device access to memory (DMA)
  - Peer to peer
    - GPU RDMA over Infiniband
    - Nvidia Optimus
- **Messaging**: Interrupts, Errors
- **Configuration I/O.**
Device DMA engines

Features and Limitations

- Scatter/Gather: number of segments
- DMA engine restrictions
  - Address width
  - Dead bits (alignment)
  - Segment length
- Streaming
- Coherence (Snoop)
- Traffic Prioritization
This document describes the Intel® Virtualization Technology for Directed I/ O ("Intel® VT for Directed I/ O"); specifically, it describes the components supporting I/ O virtualization as it applies to platforms that use Intel® processors and core logic chipsets complying with Intel® platform specifications.

Figure 1-1 illustrates the general platform topology.

The document includes the following topics:

- An overview of I/ O subsystem hardware functions for virtualization support
- A brief overview of expected usages of the generalized hardware functions
- The theory of operation of hardware, including the programming interface

The following topics are not covered (or are covered in a limited context):

- Intel® Virtualization Technology for Intel® 64 Architecture. For more information, refer to the "Intel® 64 Architecture Software Developer’s Manual, Volume 3B: System Programming Guide".
- Intel® Virtualization Technology for Intel® Itanium® Architecture. For more information, refer to the "Intel® Itanium® Architecture software developer’s manuals".

1.1 Audience

This document is aimed at hardware designers developing Intel platforms or core-logic providing hardware support for virtualization. The document is also expected to be used by operating system and virtual machine monitor (VMM) developers utilizing the I/ O virtualization hardware functions.

Figure 1-1. General Platform Topology
DMAR

- Process TLPs from devices accessing memory
- Performs
  - Address Translation and Access Control
  - Snoop Control
  - Prioritization
- Based on the originator of the TLP

Requester Identifier

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8 7 3 2</td>
</tr>
</tbody>
</table>

Bus #  Device #  Function #
Context-entry maps a specific I/O device on a bus to the domain to which it is assigned, and, in turn, to the address translation structures for the domain. The context entries are programmed through the memory-resident context-entry tables. Each root-entry in the root-entry table contains the pointer to the context-entry table for the corresponding bus number. Each context-entry table contains 256 entries, with each entry representing a unique PCI device function on the bus. For a PCI device, the device and function numbers (lower 8-bits) of a source-id are used to index into the context-entry table.

Each context-entry contains the following attributes:

- **Domain Identifier**: The domain identifier is a software-assigned field in a context entry that identifies the domain to which a device with the given source-id is assigned. Hardware may use this field to tag its caching structures. Context entries programmed with the same domain identifier must reference the same address translation structure. Context entries referencing the same address translation structure are recommended to use the same domain identifier for best hardware efficiency.

- **Present Flag**: The present field is used by software to indicate to hardware whether the context-entry is present and initialized. Software may clear the present field for context entries corresponding to device functions that are not present in the platform. If the present field of a context-entry used to process a DMA request is cleared, the DMA request is blocked, resulting in a translation fault.

- **Translation Type**: The translation-type field indicates the type of the address translation structure that must be used to address-translate a DMA request processed through the context-entry.

- **Address Width**: The address-width field indicates the address width of the domain to which the device corresponding to the context-entry is assigned.
Hardware

- Nehalem+ Xeons
- Desktop Core i7 CPUs: not -K, BIOS
- Core2 gen: G45, 5500

Documentation

- Intel® Virtualization Technology for Directed I/O, D51397-005
- External Design Specification (EDS)
- BIOS Write Guide (BWG)
- Chipset erratas
<table>
<thead>
<tr>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMI handlers, USB legacy</td>
</tr>
<tr>
<td>UMA GPU: GTT and VGA framebuffer</td>
</tr>
<tr>
<td>Service processor for BMC (AMT, IPMI, iLO, DRAC etc)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware bugs, Specification Updates</td>
</tr>
<tr>
<td>BIOS bugs</td>
</tr>
</tbody>
</table>
How to detect

acpidump -t

DMAR: Length=368, Revision=1, Checksum=7,
    OEMID=DELL, OEM Table ID=PE_SC3, OEM Revision=0x1,
    Creator ID=DELL, Creator Revision=0x1
    Host Address Width=46
    Flags={INTR_REMAP,X2APIC_OPT_OUT}
VT-d

Other features

- Interrupt remapping
  - MSI, MSI-X: memory write
  - IO-APICs
  - FSB interrupts: HPET
- ATS (Address Translation Service): IO TLB in devices
- Hypervisors PCI pass-through
PCI-era

- Architectures
  - SPARC4u
  - POWER: DART
- coarse domains
DMA in FreeBSD

**Busdma(9) layer**
- FreeBSD KPI abstracting access to DMA implementations from NetBSD

**Busdma(9) overview**
- Tags: device capabilities
- Maps: Accessible memory
- Loads and unloads: maps activation and deactivation
Busdma implementations

**Bounce buffers**

- Allocate memory to satisfy device constraints
  - contigmalloc(9)
  - Low 16MB, low 4GB
- Copy to/from
- Flush cache on non-coherent platforms
Busdma implementations

IOMMU: pro
- Performance: No bouncing
- Stability: No memory corruption
- Privacy: Only sanctioned access to memory
- Driver debugging: Reports of violations

IOMMU: contra
- Performance: Page table setup
- Performance: Translation overhead
Busdma over VT-d

Layers
- Page tables and TLB invalidation
- Fault handler
- Context and domain
- Busdma emulation

Integration
- ACPI: DMAR table parsing
  - DMAR discovery
  - RMRR and BIOS bugs
- newbus: bus_get_dma_tag()
- fallback to bounce, enabling pass-through
Busdma over VT-d

Busdma KPI problems

- Locking
  - BUS_DMA_NOWAIT abuse
  - bus_dmamap_unload(9) cannot sleep
- No I/O direction
- Tag specification of alignment
### Current state

- **Drivers**
  - Storage: ahci(4), mfi(4)
  - USB: uhci(4), ehci(4)
  - Network: em(4), igb(4) (*), bce(4)

- **Platforms**
  - Xeon 5400, 5500 NB
  - Xeon Romely-EP (E5-26XX)
  - Haswell (Core i7 4770)

- **Not supported yet**
  - Intel GPUs

- **Not tested**
  - HDA
  - Discrete GPUs (Radeon, Nvidia)
  - Everything else (HW bugs)