Intel Graphics Virtualization on KVM

Aug-16-2011

allen.m.kay@intel.com

KVM Forum 2011

Rev. 3

Agenda

- Background on IO Virtualization
  - Device Operation on Native Platform
  - QEMU IO Virtualization
  - Device Direct Assignment
- Intel Graphics Direct Assignment
- Status

Background: Native Device Operation

- Device driver finds matching device with vendor/device ID
  - Reads device’s PCI configuration space
  - Writes bus/device/func/offset into IO port 0xCF8
  - Reads data from IO port 0xCFC
- Device HW responds to IO PORT accesses
- MMIO is mapped un-cached with CPU page tables
  - Driver can directly access device MMIO memory
- Driver programs physical address to device to do DMA
  - No virtual-to-physical translation from IO side
- Device HW generates interrupts when DMA completes

Background: QEMU Software IO Virtualization

- QEMU provides virtual platform for HVM guests
- Virtual PCI devices hang off on a virtual PCI bus
- QEMU device model emulates real device HW behavior
  - PCIConfig, MMIO, IO port, DMA, interrupts operations are trapped by the hypervisor and forwarded to QEMU device models to handle
- QEMU emulates 0xCF8/0xCFC IO port accesses
  - QEMU device model responds to bus/dev/func/offset PCI config access
- PCI vendor/device ID's in PCI configuration space are hardcoded to match real HW
  - i.e. Realtek 8139 NIC has vendor_id = 0x10ec, device_id = 0x8139
  - Native driver running in the guest initializes base on vendor/device ID's
  - Native driver thinks it is talking to a real HW device

Background: Device Direct Assignment

- PCI Configuration Access
  - Configuration space values are read off from real HW – not hard coded
  - Native device driver in the guest can initialize base on real vendor/device ID's
  - PCI BAR register accesses are emulated as in QEMU model case
    - To avoid guest OS changing device's BAR register values on the host platform
  - Other registers such as COMMAND register are forwarded to real HW
- GPA-to-HPA translation is setup for MMIO
  - Host maps MMIO BAR in KVM user space via sysfs
  - Host creates a new memory slot for the MMIO BAR of the assigned device
  - When the guest accesses MMIO region, page fault is resolved according to the new MMIO memory slot
- IO port accesses
  - Use VMCS bit maps to allow the guest to access IO port directly without causing VM exits

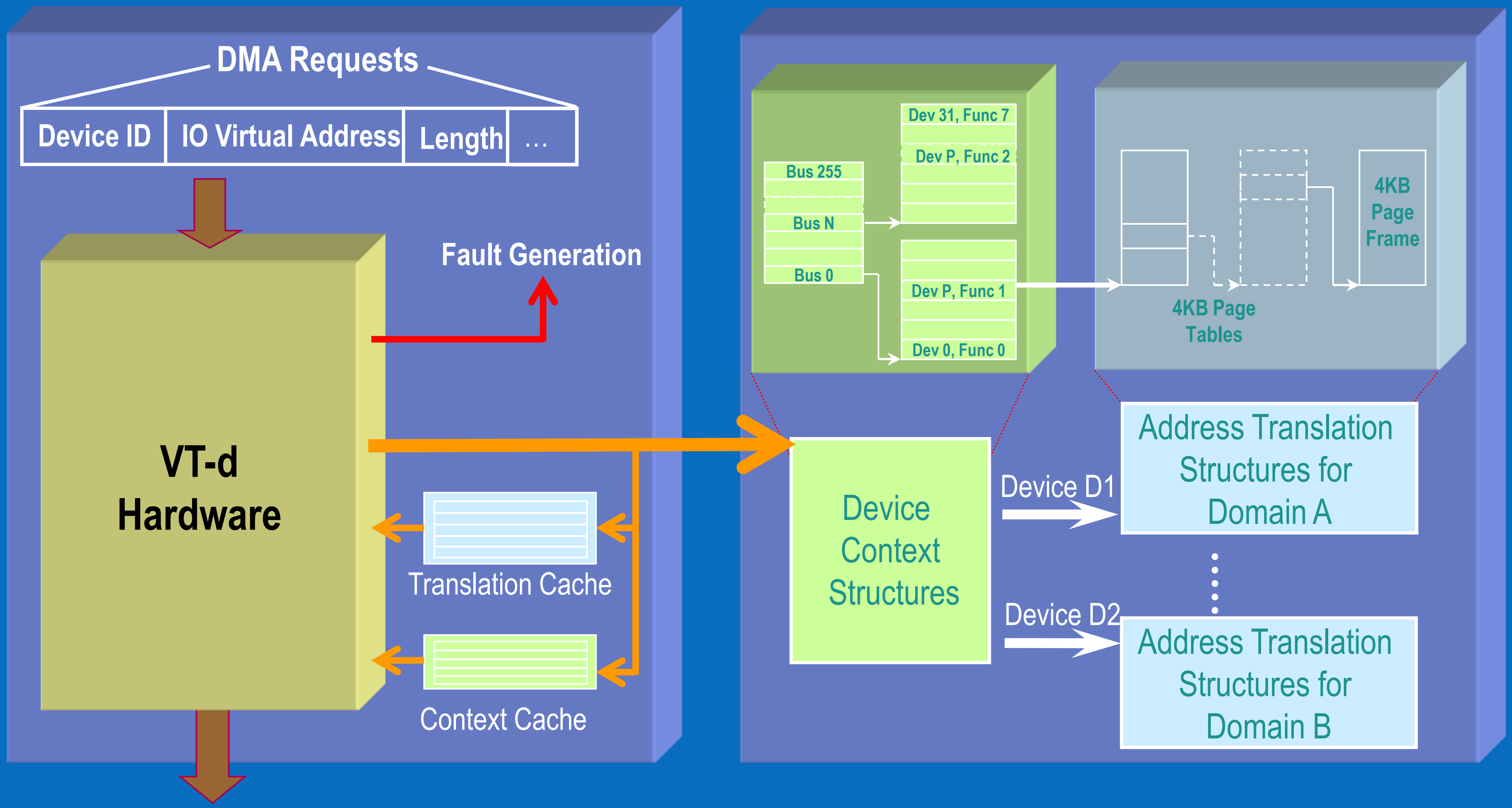
Background: Device Assignment (Interrupt)

- Interrupts
  - Host registers a pass-through interrupt handler for IRQ on behalf of the guest
  - Interrupt from pass-through device received by the host and injected into guest
    - Interrupt is raised in virtual IO-APIC
    - Virtual IO-APIC propagates the interrupts to virtual local APIC
    - During VMENTER, KVM sets external interrupt in VMCS if there are interrupt pending in virtual local APIC
  - Guest APIC accesses are emulated by virtual LAPIC and IO-APIC models in KVM
    - Guest writes to EOI register causes VMEXIT
    - Virtual local APIC and IO-APIC registers are changed the same way as in real hardware
  - To avoid complexity, devices with shared level interrupts cannot be assigned
    - Handling sharing is very complicated with many corner cases
    - If not handle correctly, a failure in the guest can affect host functionality
      - If a guest fails to issue EOI , it can block interrupts to the device sharing the interrupt
      - If the device is SATA on the host, then system will hang
    - Not a big issue as there are now many MSI capable devices available and MSI interrupts are edge triggered
    - Exception: USB controllers – which can complicate graphics assignment use case

Background: Device Assignment (DMA)

- Driver program physical address to DMA engines
- In guest environment, guest physical address (GPA) != host physical address (HPA)
  - DMA would fail since GPA does not point to real memory
- IOMMU HW such as VT-d was created to auto translate GPA to HPA
- No need to change device drivers
  - Continue to program GPA to device DMA HW
  - VT-d HW automatically GPA references from devices to HPA

Background: VT-d Hardware Overview



Memory Access with Host Physical Address

Memory-resident IO Partitioning & Translation Structures

Background: Device Assignment Example

- Pass-through PCI device at BDF 01:00.0
- QEMU reads PCI configuration space register using sysfs
  - /sys/bus/pci/devices/0000:01:00.0/config
- QEMU constructs a virtual PCI configuration space with same content as 01:00.0 on real platform
- Guest PCI configuration accesses to device 01:00.0 is handled as pass-through device
  - Some registers are emulated: i.e. BAR registers
  - Other registers are passed through to HW: command register
- Access to any non pass-through devices are still emulated
  - Example: host bridge device 00:00.0

Graphics Direct Assignment

- Most source code changes are in QEMU (pt-graphics.c)
- Hypervisor change to support 1:1 graphics memory mapping
  - Legacy VGA 0xa0000 – 0xc0000
  - Intel graphics op-region: not reported in PCI BAR register
  - Op-region is used by driver to get info setup by the BIOS
- PCI config read/write of device 0
  - Need to pass-through certain device 0 PCI config registers
  - Windows driver accesses some PCI config registers in device 0
- Need to create a second PCH device 0x1f
  - Pass-through vendor_id, device_id, revision_id of real HW
  - Windows driver hard codes references to this device
  - Some display HW is located in PCH

Graphics Direct Assignment (2)

- Working closely with graphics group in resolving virtualization issues encountered
- FLR complication
  - FLR clears MMIO registers that was setup by system BIOS
  - Example: graphics C-state were disabled
  - Need to save/restore certain MMIO registers across FLR's

QEMU and KVM Changes

- How to invoke VGA assignment
  - Add “-vga passthru -device pci-assign,host=00:02.0” to QEMU command
- Added functions to read/write PCI config space base on bus/dev/func/offset
  - Existing functions only work for assigned devices
- Graphics related hooks in QEMU
  - In assigned_initfn(): copy option ROM content to guest’s 0xc0000 memory
  - In assigned_dev_register_regions(): setup VGA compatible MMIO and IO port resources
- Hypervisor enhancement
  - Added a new hypercall to identity map multiple pages
  - For 1:1 map VGA and Intel graphics op_region memory

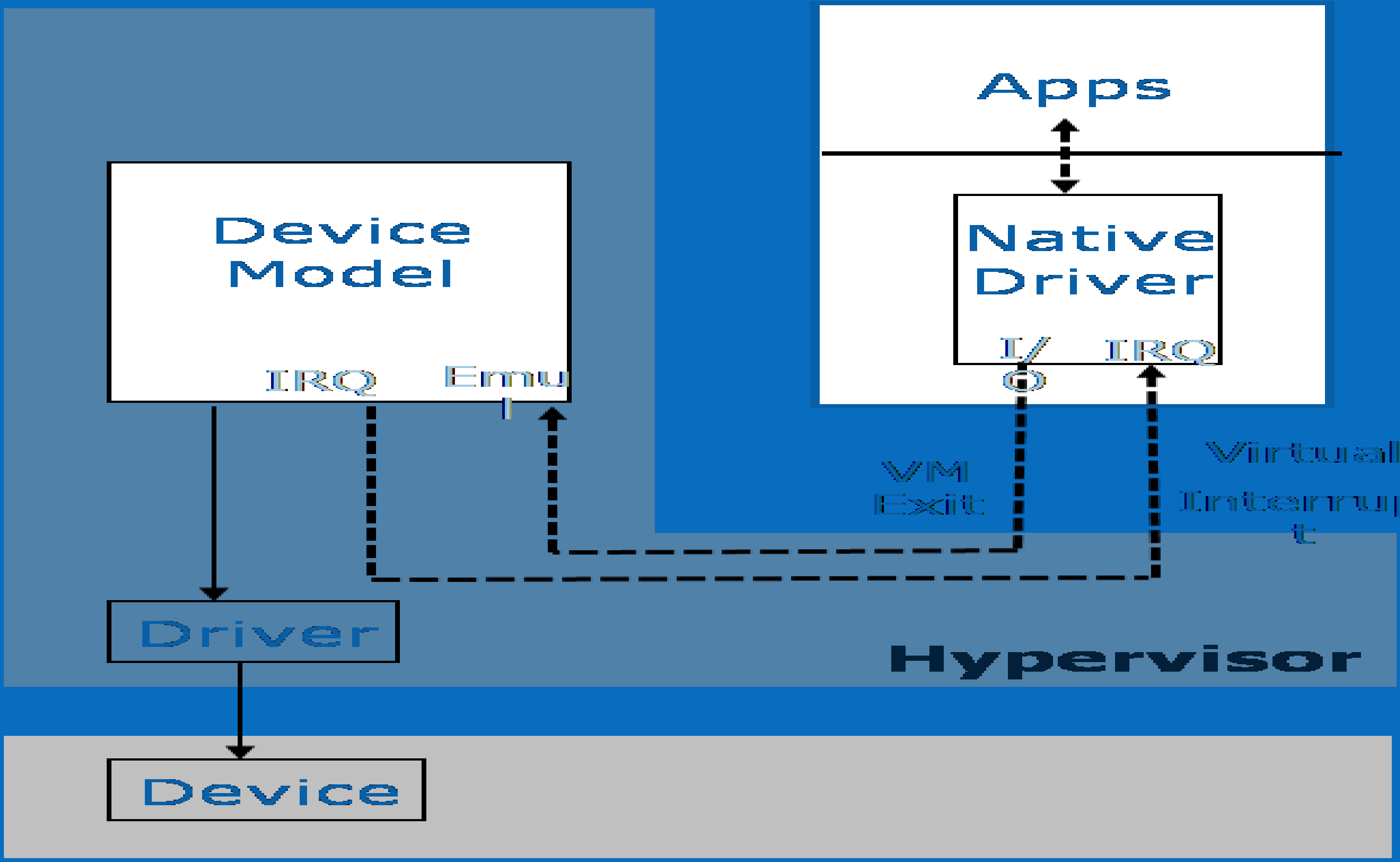
Status

- Finished code changes and enhancements
- In process of bring-up and debug
  - Graphics driver complains about not able to find Video Bios Table (VBT)
- Working with graphics group in resolving virtualization issues encountered

Questions?

Backup

Background: QEMU IO Virtualization



Legal Information

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.
- Intel may make changes to specifications, product descriptions, and plans at any time, without notice.
- All dates provided are subject to change without notice.
- Intel is a trademark of Intel Corporation in the U.S. and other countries.
- *Other names and brands may be claimed as the property of others.
-

Leap ahead™