Linux IOMMU Support

The architecture spec can be obtained from the below location.

http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf

This guide gives a quick cheat sheet for some basic understanding.

Some Keywords

- DMAR - DMA remapping
- DRHD - DMA Remapping Hardware Unit Definition
- RMRR - Reserved memory Region Reporting Structure
- ZLR - Zero length reads from PCI devices
- IOVA - IO Virtual address.

Basic stuff

ACPI enumerates and lists the different DMA engines in the platform, and device scope relationships between PCI devices and which DMA engine controls them.

What is RMRR?

There are some devices the BIOS controls, for e.g USB devices to perform PS2 emulation. The regions of memory used for these devices are marked reserved in the e820 map. When we turn on DMA translation, DMA to those regions will fail. Hence BIOS uses RMRR to specify these regions along with devices that need to access these regions. OS is expected to setup unity mappings for these regions for these devices to access these regions.

How is IOVA generated?

Well behaved drivers call pci_map_*() calls before sending command to device that needs to perform DMA. Once DMA is completed and mapping is no longer required, device performs a pci_unmap_*() calls to unmap the region.

The Intel IOMMU driver allocates a virtual address per domain. Each PCIE device has its own domain (hence protection). Devices under p2p bridges share the virtual address with all devices under the p2p bridge due to transaction id aliasing for p2p bridges.

IOVA generation is pretty generic. We used the same technique as vmalloc() but these are not global address spaces, but separate for each domain. Different DMA engines may support different number of domains.

We also allocate guard pages with each mapping, so we can attempt to catch any overflow that might happen.

Graphics Problems?

If you encounter issues with graphics devices, you can try adding option intel_iommu=igfx_off to turn off the integrated graphics engine. If this fixes anything, please ensure you file a bug reporting the problem.

Some exceptions to IOVA

Interrupt ranges are not address translated, (0xfee00000 - 0xfeeffffff). The same is true for peer to peer transactions. Hence we reserve the address from PCI MMIO ranges so they are not allocated for IOVA addresses.

Fault reporting

When errors are reported, the DMA engine signals via an interrupt. The fault reason and device that caused it with fault reason is printed on console.

See below for sample.

Boot Message Sample

Something like this gets printed indicating presence of DMAR tables in ACPI.

ACPI: DMAR (v001 A M I OEMDMAR 0x00000001 MSFT 0x00000097) @ 0x000000007f5b5ef0

When DMAR is being processed and initialized by ACPI, prints DMAR locations and any RMRR's processed:

```
ACPI DMAR:Host address width 36
ACPI DMAR:DRHD (flags: 0x00000000)base: 0x00000000fed90000
ACPI DMAR:DRHD (flags: 0x00000000)base: 0x00000000fed91000
```

```
ACPI DMAR:DRHD (flags: 0x00000001)base: 0x00000000fed93000
ACPI DMAR:RMRR base: 0x00000000000ed000 end: 0x00000000000effff
ACPI DMAR:RMRR base: 0x000000007f600000 end: 0x000000007fffffff
```

When DMAR is enabled for use, you will notice..

PCI-DMA: Using DMAR IOMMU

Fault reporting

```
DMAR:[DMA Write] Request device [00:02.0] fault addr 6df084000
DMAR:[fault reason 05] PTE Write access is not set
DMAR:[DMA Write] Request device [00:02.0] fault addr 6df084000
DMAR:[fault reason 05] PTE Write access is not set
```

TBD

- For compatibility testing, could use unity map domain for all devices, just provide a 1-1 for all useful memory under a single domain for all devices.
- API for paravirt ops for abstracting functionality for VMM folks.