How closely do we model real hardware in QEMU?

Anthony Liguori <anthony@codemonkey.ws>
Why?

- QEMU is a **functional simulator**
- Learn from the past, avoid repeating mistakes
- Informed decisions about deviating
- Anticipate future emulation requirements
Overview: -M pc

- ->machine() function creates a tangle of things
  - Initial memory layout
  - IRQ routing tables
  - i440fx (piix gets automagically created)
  - ISA bus and assortment of devices
  - PIIX3 IDE and USB functions
  - Default devices

- The world is flat after this point
Overview:

- **I440fx consists of PMC and PHB**
- **PIIX4 is the Super I/O chip**
- **Integrated IDE controller**
- **PIIX4 adds USB controller**

Comparison to modern hardware:

- Northbridge external to processors
- Southbridge separate from northbridge
- Local APIC could be external
- Did support SMP
- Had very limited support for RAM (1GB)
Flows: RAM read from CPU0

**Observations**

- We get this right!

**read(0x0100 0000)**

- Request is sent to ICB destined for I440FX
- PMC checks against PCI window and PAM window and then dispatches to RAM
**Flows: Write to E1000 bar0**

- Request goes to PCI bus
- Device uses Base Address Registers to determine if it handles the request
- Device asserts #DEVSEL to indicate that it handles request

**Observations**
- We can't check BARs in parallel
- We maintain a dispatch table
- The PHB has many opportunities to alter request
Aside: device endianness

Endianness does not exist in hardware!

So why do we have device_endianness?
Bugs, bugs, everywhere

- X86 has two address spaces: memory and IO
- Most architectures just have one: memory
- PCI has two address spaces
- Non-x86 PCI Host Controllers reserve a range of memory space for PCI IO memory
if (in_range(addr, io_window, io_window_size)) {
    send_pci_req(io, addr, size);
} else {
    send_pci_req(mem, addr, size);
}
Linear dispatch

- Register a memory region for I/O window
- Call cpu_outb/inb
- And byte swap ← BUG

- Each level of endian swapping cancels a previous

- All users need auditing and device_endianness should die
Flows: Opt. ROM read from CPU0

- Request is sent to ICB destined for I440FX
- PMC checks against PAM table
  - Separate bits for read vs. Write
  - Either redirect to RAM or ROM via ISA bus

Observations
- This would be very slow to emulate correctly under QEMU
- Recent kernels allow us to partially emulate this with KVM
Flows: LAPIC from CPU0

Modern CPUs simply implement LAPIC functionality as part of the core.

Observations

- Nothing other than CPU0 can read or write to CPU0's local APIC
- Devices cannot DMA to local APIC

read(0xFEC0 00xx)

- Modern CPUs simply implement LAPIC functionality as part of the core
Observations

- We treat all PCI slots as equal
- We create separate a separate i440fx device that lives as a child of the i440fx-pcihost device
- This lives in piix.c oddly enough
Flows: Read IDE PCI config

- PCI configuration requests are decoded in the i440fx
- Request goes to PCI bus
- PIIX4 responds on behalf of embedded IDE controller

Observations
- We don't model functions vs. slots
- This works okay for simple and mostly discrete functions
- This fails for sophisticated devices that provide virtual functions
Flows: DMA from E1000

DMA to 0x0010 0000
- Request goes to PCI bus
- PCI bus routes through the PMC

Observations
- PHB can remap requests
- More complex topologies are possible
Flows: Read to RTC

- Request goes to PCI bus
- PIIX4 claims request
- RTC handles and responds

Observations
- Request never enters ISA bus
- Very special treatment of PIIX4 devices

outb 0x70
**Flows: Write to serial port**

- **outb 0x3f8**
  - Request goes to PCI bus
  - If no device claims the request, it is directed to the first PCI-to-ISA bridge
    - Subtractive decoding
  - Request is placed on ISA bus
  - Any device can handle it (or not)

**Observations**
- ISA flows through PCI!
- ISA != Super I/O devices
- ISA really isn't a useful bus
Conclusions

- I/O flows are hierarchical
  - We emulate them with flat dispatch

- We get endianness very wrong

- We will eventually need to fix these things
Questions?