IOAPIC

From OSDev Wiki

Contents

- 1 Basic Info
- 2 Detecting I/O APIC
- 3 Programming the I/O APIC
- 4 IOAPICID
- 5 IOAPICVER
- 6 IOAPICARB
- 7 IOREDTBL
  - 7.1 IOREGSEL and IOWIN
- 8 External Links
  - 8.1 MP Tables
  - 8.2 I/O APIC

Basic Info

The Intel I/O Advanced Programmable Interrupt Controller is used to distribute external interrupts in a more advanced manner than that of the standard 8259 PIC. With the I/O APIC, interrupts can be distributed to physical or logical (clusters) of processors and can be prioritized. Each I/O APIC typically handles 24 external interrupts.

Detecting I/O APIC

In order to detect the existence of an I/O APIC (or multiple ones), the Intel Multi-Processor or ACPI tables (specifically, the MADT) must be parsed. In the MP tables, configuration tables with the entry identification of 0x02 are for I/O APICs. Parsing will tell how many (if any) I/O APICs exist, what are their APIC ID, base MMIO address and first IRQ (or GSI - Global System Interrupt). For more information of parsing the MP tables, see the External MP Tables Links section below. So you can have, say, 2 I/O APICs, the first handling IRQs 0 - 23 and the second 24 - 47.

Programming the I/O APIC

Each I/O APIC has a set of 2 or 3 (depending on version) 32-bit registers and up to many 64-bit registers (one per IRQ). The 64-bit registers have actually to be accessed as two 32-bit reads/writes. All registers are memory indexed. It means that you actually have only two 32-bit registers in memory, called IOREGSEL and IOREGWIN. You put the register index in IOREGSEL, and then you can read/write in IOREGWIN. The first three registers contain general informations about this I/O APIC, the remaing contain the specific configuration for each of the IRQs.

IOAPICID

This register has index 0 (you write 0 to IOREGSEL and then read from IOREGWIN). It's a Read Only registers with almost all bits reserved. The only interesting field is in bits 24 - 27: the APIC ID for this device (each peripheral which is interfaced with the APIC Bus needs an APIC ID, not only CPUs). You shall find this ID in ACPI/MP Tables as well.

IOAPICVER

This register (index 1) contains the I/O APIC Version in bits 0 - 8, and the **Max Redirection Entry** which is "how many IRQs can this I/O APIC handle - 1". It is encoded in bits 16 - 23.

IOAPICARB

This register (index 2) contains in bits 24 - 27 the APIC Arbitration ID. **TODO**

IOREDTBL

Following there are two 32-bit register for each IRQ. The first IRQ has indexes 0x10 and 0x11, the second 0x12 and 0x13, the third 0x14 and 0x15, and so on. So the *Redirection Entry* register for IRQ *n* is 0x10 + n * 2 (+ 1). In the first of the two registers you access to the LOW dword / bits 31:0, and the second for the high dword / 63:32. Each redirection entry is made of the following fields:

Field	Bits	Description
Vector	0 - 7	The Interrupt vector that will be raised on the specified CPU(s).
Delivery Mode	8 - 10	How the interrupt will be sent to the CPU(s). It can be 000 (Fixed), 001 (Lowest Priority), 010 (SMI), 100 (NMI), 101 (INIT) and 111 (ExtINT). Most of the cases you want Fixed mode, or Lowest Priority if you don't want to suspend a high priority task on some important Processor/Core /Thread.
Destination Mode	11	Specify how the Destination field shall be interpreted. 0: Physical Destination, 1: Logical Destination
Delivery Status	12	If 0, the IRQ is just relaxed and waiting for something to happen (or it has fired and already processed by Local APIC(s)). If 1, it means that the IRQ has been sent to the Local APICs but it's still waiting to be delivered.
Pin Polarity	13	0: Active high, 1: Active low. For ISA IRQs assume Active High unless otherwise specified in Interrupt Source Override descriptors of the MADT or in the MP Tables.
Remote IRR	14	**TODO**
Trigger Mode	15	0: Edge, 1: Level. For ISA IRQs assume Edge unless otherwise specified in Interrupt Source Override descriptors of the MADT or in the MP Tables.
Mask	16	Just like in the old PIC, you can temporary disable this IRQ by setting this bit, and reenable it by clearing the bit.
Destination	56 - 63	This field is interpreted according to the Destination Format bit. If Physical destination is choosen, then this field is limited to bits 56 - 59 (only 16 CPUs addressable). You put here the APIC ID of the CPU that you want to receive the interrupt. **TODO**: Logical destination format...

IOREGSEL and IOWIN

The register IOREGSEL is an MMIO register select register that is used to access all the other I/O APIC registers. The IOWIN register is the 'data' register. Once the IOREGSEL register has been set, the IOWIN register can be used to write or read the register in the IOREGSEL. The actual position in memory of the two registers is specified in the ACPI MADT Table and/or in the MP table. The IOREGSEL is at the address specified, and IOREGWIN is at the same address + 0x10.

```
void write_ioapic_register(const uintptr_t apic_base, const uint8_
{
    /* tell IOREGSEL where we want to write to */
    *(uint32_t*)(apic_base) = offset;
    /* write the value to IOWIN */
    *(uint32_t*)(apic_base + 0x10) = val;
}

uint32_t read_ioapic_register(const uintptr_t apic_base, const uir
{
    /* tell IOREGSEL where we want to read from */
    *(uint32_t*)(apic_base) = offset;
    /* return the data from IOWIN */
    return *(uint32_t*)(apic_base + 0x10);
}
```

Note: 'uintptr_t' is the size of the default memory reference. If a 32-bit Protected Mode setup, it's equivalent to uint32_t, and uint64_t in 64-bit Long Mode environments. 'apic_base' is the memory base address for a selected IOAPIC, these can be found by enumerating them from the MP or ACPI Tables.

External Links

MP Tables

- Intel MultiProcessor Specification (http://download.intel.com/design/pentium /datashts/24201606.pdf)

I/O APIC

- Intel 82093AA I/O APIC (http://download.intel.com/design/chipsets/datashts /29056601.pdf)
- Intel SDM 3A (see ch. 9.9) (http://download.intel.com/design/processor /manuals/253668.pdf)

Retrieved from "http://wiki.osdev.org/index.php?title=IOAPIC&oldid=16641"
Categories: Interrupts | Multiprocessing

- This page was last modified on 28 August 2014, at 06:11.
- This page has been accessed 28,884 times.