

# Update MSI Patches

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2003-08/4049.html>

---

**From:** long (tlnguyen\_at\_snoqualmie.dp.intel.com)

**Date:** 08/14/03

Date: Thu, 14 Aug 2003 12:14:22 -0700

To: zwane@linuxpower.ca

Thank you for providing us all of your feedback on the MSI patches. The attach is an update of vector base patch to reflect your feedback. We appreciate for more comments ...

```
diff -X excludes -urN linux-2.6.0-test2/arch/i386/Kconfig
linux-2.6.0-test2-create-vectorbase/arch/i386/Kconfig
---- linux-2.6.0-test2/arch/i386/Kconfig 2003-07-27 12:57:48.000000000 -0400
+++ linux-2.6.0-test2-create-vectorbase/arch/i386/Kconfig 2003-08-05 09:25:54.000000000 -0400
@@ -1072,6 +1072,17 @@
     depends on PCI && ((PCI_GODIRECT || PCI_GOANY) || X86_VISWS)
     default y
```

```
+config PCI_USE_VECTOR
+ bool "PCI_USE_VECTOR"
+ default n
+ help
+ This replaces the current existing IRQ-based index interrupt scheme
+ with the vector-base index scheme. The advantages of vector base over IRQ base are listed below:
+ 1) Support MSI implementation.
+ 2) Support future IOxAPIC hotplug
+
+ If you don't know what to do here, say N.
+
source "drivers/pci/Kconfig"
```

```
config ISA
diff -X excludes -urN linux-2.6.0-test2/arch/i386/kernel/i8259.c
linux-2.6.0-test2-create-vectorbase/arch/i386/kernel/i8259.c
---- linux-2.6.0-test2/arch/i386/kernel/i8259.c 2003-07-27 13:09:30.000000000 -0400
+++ linux-2.6.0-test2-create-vectorbase/arch/i386/kernel/i8259.c 2003-08-05 09:25:54.000000000 -0400
@@ -419,8 +419,10 @@
     * us. (some of these will be overridden and become
     * 'special' SMP interrupts)
     */
- for (i = 0; i < NR_IRQS; i++) {
+ for (i = 0; i < (NR_VECTORS - FIRST_EXTERNAL_VECTOR); i++) {
     int vector = FIRST_EXTERNAL_VECTOR + i;
```

## Linux-Kernel: Update MSI Patches

```
+ if (i >= NR_IRQS)
+ break;
    if (vector != SYSCALL_VECTOR)
        set_intr_gate(vector, interrupt[i]);
    }
diff -X excludes -urN linux-2.6.0-test2/arch/i386/kernel/io_apic.c
linux-2.6.0-test2-create-vectorbase/arch/i386/kernel/io_apic.c
--- linux-2.6.0-test2/arch/i386/kernel/io_apic.c 2003-07-27 13:00:21.000000000 -0400
+++ linux-2.6.0-test2-create-vectorbase/arch/i386/kernel/io_apic.c 2003-08-12 14:07:15.000000000
-0400
@@ -76,6 +76,14 @@
    int apic, pin, next;
} irq_2_pin[PIN_MAP_SIZE];

#ifdef CONFIG_PCI_USE_VECTOR
+int vector_irq[NR_IRQS] = { [0 ... NR_IRQS - 1] = -1 };
+#define vector_to_irq(vector) \
+ (platform_legacy_irq(vector) ? vector : vector_irq[vector])
+#else
+#define vector_to_irq(vector) (vector)
+#endif
+
/*
 * The common case is 1:1 IRQ<->pin mappings. Sometimes there are
 * shared ISA-space IRQs, so we have to support them. We are super
@@ -249,7 +257,7 @@
    clear_IO_APIC_pin(apic, pin);
}

-static void set_ioapic_affinity (unsigned int irq, unsigned long cpu_mask)
+static void set_ioapic_affinity_irq (unsigned int irq, unsigned long cpu_mask)
{
    unsigned long flags;
    int pin;
@@ -667,13 +675,13 @@

__setup("noirqbalance", irqbalance_disable);

-static void set_ioapic_affinity (unsigned int irq, unsigned long mask);
+static void set_ioapic_affinity_irq (unsigned int irq, unsigned long mask);

static inline void move_irq(int irq)
{
    /* note - we hold the desc->lock */
    if (unlikely(pending_irq_balance_cpumask[irq])) {
- set_ioapic_affinity(irq, pending_irq_balance_cpumask[irq]);
+ set_ioapic_affinity_irq(irq, pending_irq_balance_cpumask[irq]);
        pending_irq_balance_cpumask[irq] = 0;
    }
}
@@ -850,7 +858,7 @@
```

## Linux-Kernel: Update MSI Patches

```
        if (irq_entry == -1)
            continue;
        irq = pin_2_irq(irq_entry, ioapic, pin);
- set_ioapic_affinity(irq, mask);
+ set_ioapic_affinity_irq(irq, mask);
    }

}

@@ -1140,8 +1148,10 @@
static int __init assign_irq_vector(int irq)
{
    static int current_vector = FIRST_DEVICE_VECTOR, offset = 0;
- if (IO_APIC_VECTOR(irq) > 0)
- return IO_APIC_VECTOR(irq);
+ int vector;
+
+ if ((vector = IO_APIC_VECTOR(irq)) > 0)
+ return vector;
next:
    current_vector += 8;
    if (current_vector == SYSCALL_VECTOR)
@@ -1152,12 +1162,40 @@
        current_vector = FIRST_DEVICE_VECTOR + offset;
    }

#ifdef CONFIG_PCI_USE_VECTOR
+ vector_irq[current_vector] = irq;
#endif
+
+ IO_APIC_VECTOR(irq) = current_vector;
+
+ return current_vector;
}

-static struct hw_interrupt_type ioapic_level_irq_type;
-static struct hw_interrupt_type ioapic_edge_irq_type;
+static struct hw_interrupt_type ioapic_level_type;
+static struct hw_interrupt_type ioapic_edge_type;
+
+#define IOAPIC_AUTO -1
+#define IOAPIC_EDGE 0
+#define IOAPIC_LEVEL 1
+
+static inline void ioapic_register_intr(int irq, int vector, unsigned long trigger)
+{
+ if (use_pci_vector() && !platform_legacy_irq(irq)) {
+ if ((trigger == IOAPIC_AUTO && IO_APIC_irq_trigger(irq)) ||
+ trigger == IOAPIC_LEVEL)
+ irq_desc[vector].handler = &ioapic_level_type;
+ else
+ irq_desc[vector].handler = &ioapic_edge_type;
```

## Linux–Kernel: Update MSI Patches

```
+ set_intr_gate(vector, interrupt[vector]);
+ } else {
+ if ((trigger == IOAPIC_AUTO && IO_APIC_irq_trigger(irq)) ||
+ trigger == IOAPIC_LEVEL)
+ irq_desc[irq].handler = &ioapic_level_type;
+ else
+ irq_desc[irq].handler = &ioapic_edge_type;
+ set_intr_gate(vector, interrupt[irq]);
+ }
+}

void __init setup_IO_APIC_irqs(void)
{
@@ -1215,13 +1253,7 @@
    if (IO_APIC_IRQ(irq)) {
        vector = assign_irq_vector(irq);
        entry.vector = vector;
-
- if (IO_APIC_irq_trigger(irq))
- irq_desc[irq].handler = &ioapic_level_irq_type;
- else
- irq_desc[irq].handler = &ioapic_edge_irq_type;
-
- set_intr_gate(vector, interrupt[irq]);
+ ioapic_register_intr(irq, vector, IOAPIC_AUTO);

        if (!apic && (irq < 16))
            disable_8259A_irq(irq);
@@ -1268,7 +1300,7 @@
    * The timer IRQ doesn't have to know that behind the
    * scene we have a 8259A–master in AEOI mode ...
    */
- irq_desc[0].handler = &ioapic_edge_irq_type;
+ irq_desc[0].handler = &ioapic_edge_type;

    /*
    * Add it to the IO–APIC irq–routing table:
@@ -1736,6 +1768,7 @@
    return 0;
}

#ifdef CONFIG_PCI_USE_VECTOR
/*
* In the SMP+IOAPIC case it might happen that there are an unspecified
* number of pending IRQ events unhandled. These cases are very rare,
@@ -1748,9 +1781,6 @@
* that was delayed but this is now handled in the device
* independent code.
*/
#define enable_edge_ioapic_irq unmask_IO_APIC_irq
-

```

## Linux–Kernel: Update MSI Patches

```
–static void disable_edge_ioapic_irq (unsigned int irq) { /* nothing */ }

/*
 * Starting up a edge–triggered IO–APIC interrupt is
@@ –1761,7 +1791,175 @@
 * This is not complete – we should be able to fake
 * an edge even if it isn't on the 8259A...
 */
+static unsigned int startup_edge_ioapic_vector(unsigned int vector)
+{
+ int was_pending = 0;
+ int irq = vector_to_irq(vector);
+ unsigned long flags;
+
+ spin_lock_irqsave(&ioapic_lock, flags);
+ if (irq < 16) {
+ disable_8259A_irq(irq);
+ if (i8259A_irq_pending(irq))
+ was_pending = 1;
+ }
+ __unmask_IO_APIC_irq(irq);
+ spin_unlock_irqrestore(&ioapic_lock, flags);
+
+ return was_pending;
+}
+
+/*
+ * Once we have recorded IRQ_PENDING already, we can mask the
+ * interrupt for real. This prevents IRQ storms from unhandled
+ * devices.
+ */
+static void ack_edge_ioapic_vector(unsigned int vector)
+{
+ int irq = vector_to_irq(vector);
+
+ move_irq(irq);
+ if ((irq_desc[irq].status & (IRQ_PENDING | IRQ_DISABLED))
+ == (IRQ_PENDING | IRQ_DISABLED))
+ mask_IO_APIC_irq(irq);
+ ack_APIC_irq();
+}
+
+/*
+ * Level triggered interrupts can just be masked,
+ * and shutting down and starting up the interrupt
+ * is the same as enabling and disabling them — except
+ * with a startup need to return a "was pending" value.
+ *
+ * Level triggered interrupts are special because we
+ * do not touch any IO–APIC register while handling
+ * them. We ack the APIC in the end–IRQ handler, not
```

## Linux–Kernel: Update MSI Patches

```
+ * in the start–IRQ–handler. Protection against reentrance
+ * from the same interrupt is still provided, both by the
+ * generic IRQ layer and by the fact that an unacked local
+ * APIC does not accept IRQs.
+ */
+static unsigned int startup_level_ioapic_vector (unsigned int vector)
+{
+ int irq = vector_to_irq(vector);

+ unmask_IO_APIC_irq(irq);
+
+ return 0; /* don't check for pending */
+}
+
+static void end_level_ioapic_vector (unsigned int vector)
+{
+ unsigned long v;
+ int i;
+ int irq = vector_to_irq(vector);
+
+ move_irq(irq);
+ /*
+ * It appears there is an erratum which affects at least version 0x11
+ * of I/O APIC (that's the 82093AA and cores integrated into various
+ * chipsets). Under certain conditions a level–triggered interrupt is
+ * erroneously delivered as edge–triggered one but the respective IRR
+ * bit gets set nevertheless. As a result the I/O unit expects an EOI
+ * message but it will never arrive and further interrupts are blocked
+ * from the source. The exact reason is so far unknown, but the
+ * phenomenon was observed when two consecutive interrupt requests
+ * from a given source get delivered to the same CPU and the source is
+ * temporarily disabled in between.
+ *
+ * A workaround is to simulate an EOI message manually. We achieve it
+ * by setting the trigger mode to edge and then to level when the edge
+ * trigger mode gets detected in the TMR of a local APIC for a
+ * level–triggered interrupt. We mask the source for the time of the
+ * operation to prevent an edge–triggered interrupt escaping meanwhile.
+ * The idea is from Manfred Spraul. --macro
+ */
+ i = IO_APIC_VECTOR(irq);
+
+ v = apic_read(APIC_TMR + ((i & ~0x1f) >> 1));
+
+ ack_APIC_irq();
+
+ if (!(v & (1 << (i & 0x1f)))) {
+ #ifdef APIC_LOCKUP_DEBUG
+ struct irq_pin_list *entry;
+ #endif
+
+ }
```

```

+ #ifdef APIC_MISMATCH_DEBUG
+ atomic_inc(&irq_mis_count);
+ #endif
+ spin_lock(&ioapic_lock);
+ __mask_and_edge_IO_APIC_irq(irq);
+ #ifdef APIC_LOCKUP_DEBUG
+ for (entry = irq_2_pin + irq;;) {
+ unsigned int reg;
+
+ if (entry->pin == -1)
+ break;
+ reg = io_apic_read(entry->apic, 0x10 + entry->pin * 2);
+ if (reg & 0x00004000)
+ printk(KERN_CRIT "Aiee!!! Remote IRR"
+ " still set after unlock!\n");
+ if (!entry->next)
+ break;
+ entry = irq_2_pin + entry->next;
+ }
+ #endif
+ __unmask_and_level_IO_APIC_irq(irq);
+ spin_unlock(&ioapic_lock);
+ }
+
+ static void mask_IO_APIC_vector (unsigned int vector)
+ {
+ unsigned long flags;
+ int irq = vector_to_irq(vector);
+
+ spin_lock_irqsave(&ioapic_lock, flags);
+ __mask_IO_APIC_irq(irq);
+ spin_unlock_irqrestore(&ioapic_lock, flags);
+ }
+
+ static void unmask_IO_APIC_vector (unsigned int vector)
+ {
+ unsigned long flags;
+ int irq = vector_to_irq(vector);
+
+ spin_lock_irqsave(&ioapic_lock, flags);
+ __unmask_IO_APIC_irq(irq);
+ spin_unlock_irqrestore(&ioapic_lock, flags);
+ }
+
+ static void set_ioapic_affinity_vector (unsigned int vector,
+ unsigned long cpu_mask)
+ {
+ int irq = vector_to_irq(vector);
+
+ set_ioapic_affinity_irq(irq, cpu_mask);

```

## Linux–Kernel: Update MSI Patches

```
+}
+##else
+/*
+ * In the SMP+IOAPIC case it might happen that there are an unspecified
+ * number of pending IRQ events unhandled. These cases are very rare,
+ * so we 'resend' these IRQs via IPIs, to the same CPU. It's much
+ * better to do it this way as thus we do not have to be aware of
+ * 'pending' interrupts in the IRQ path, except at this point.
+ */
+/*
+ * Edge triggered needs to resend any interrupt
+ * that was delayed but this is now handled in the device
+ * independent code.
+ */
+
+/*
+ * Starting up a edge–triggered IO–APIC interrupt is
+ * nasty – we need to make sure that we get the edge.
+ * If it is already asserted for some reason, we need
+ * return 1 to indicate that is was pending.
+ *
+ * This is not complete – we should be able to fake
+ * an edge even if it isn't on the 8259A...
+ */
static unsigned int startup_edge_ioapic_irq(unsigned int irq)
{
    int was_pending = 0;
@@ -1779,8 +1977,6 @@
    return was_pending;
}

–#define shutdown_edge_ioapic_irq disable_edge_ioapic_irq
–
/*
 * Once we have recorded IRQ_PENDING already, we can mask the
 * interrupt for real. This prevents IRQ storms from unhandled
@@ -1795,9 +1991,6 @@
    ack_APIC_irq();
}

–static void end_edge_ioapic_irq (unsigned int i) { /* nothing */ }
–
–
/*
 * Level triggered interrupts can just be masked,
 * and shutting down and starting up the interrupt
@@ -1819,10 +2012,6 @@
    return 0; /* don't check for pending */
}

–#define shutdown_level_ioapic_irq mask_IO_APIC_irq
```

## Linux–Kernel: Update MSI Patches

```
–#define enable_level_ioapic_irq unmask_IO_APIC_irq
–#define disable_level_ioapic_irq mask_IO_APIC_irq
–
static void end_level_ioapic_irq (unsigned int irq)
{
    unsigned long v;
@@ –1849,6 +2038,7 @@
    * The idea is from Manfred Spraul. --macro
    */
    i = IO_APIC_VECTOR(irq);
+
    v = apic_read(APIC_TMR + ((i & ~0x1f) >> 1));

    ack_APIC_irq();
@@ –1882,8 +2072,7 @@
    spin_unlock(&ioapic_lock);
}
}
–
–static void mask_and_ack_level_ioapic_irq (unsigned int irq) { /* nothing */ }
+##endif

/*
 * Level and edge triggered IO–APIC interrupts need different handling,
@@ –1893,26 +2082,25 @@
 * edge–triggered handler, without risking IRQ storms and other ugly
 * races.
 */
–
–static struct hw_interrupt_type ioapic_edge_irq_type = {
+static struct hw_interrupt_type ioapic_edge_type = {
    .typename = "IO–APIC–edge",
– .startup = startup_edge_ioapic_irq,
– .shutdown = shutdown_edge_ioapic_irq,
– .enable = enable_edge_ioapic_irq,
– .disable = disable_edge_ioapic_irq,
– .ack = ack_edge_ioapic_irq,
– .end = end_edge_ioapic_irq,
+ .startup = startup_edge_ioapic,
+ .shutdown = shutdown_edge_ioapic,
+ .enable = enable_edge_ioapic,
+ .disable = disable_edge_ioapic,
+ .ack = ack_edge_ioapic,
+ .end = end_edge_ioapic,
    .set_affinity = set_ioapic_affinity,
};

–static struct hw_interrupt_type ioapic_level_irq_type = {
+static struct hw_interrupt_type ioapic_level_type = {
    .typename = "IO–APIC–level",
– .startup = startup_level_ioapic_irq,
```

## Linux–Kernel: Update MSI Patches

```

- .shutdown = shutdown_level_ioapic_irq,
- .enable = enable_level_ioapic_irq,
- .disable = disable_level_ioapic_irq,
- .ack = mask_and_ack_level_ioapic_irq,
- .end = end_level_ioapic_irq,
+ .startup = startup_level_ioapic,
+ .shutdown = shutdown_level_ioapic,
+ .enable = enable_level_ioapic,
+ .disable = disable_level_ioapic,
+ .ack = mask_and_ack_level_ioapic,
+ .end = end_level_ioapic,
    .set_affinity = set_ioapic_affinity,
};

@@ -1932,7 +2120,13 @@
    * 0x80, because int 0x80 is hm, kind of importantish. ;)
    */
    for (irq = 0; irq < NR_IRQS ; irq++) {
- if (IO_APIC_IRQ(irq) && !IO_APIC_VECTOR(irq)) {
+ int tmp = irq;
+ if (use_pci_vector()) {
+ if (!platform_legacy_irq(tmp))
+ if ((tmp = vector_to_irq(tmp)) == -1)
+ continue;
+ }
+ if (IO_APIC_IRQ(tmp) && !IO_APIC_VECTOR(tmp)) {
    /*
        * Hmm.. We don't have an entry for this,
        * so default to an old-fashioned 8259
@@ -2362,9 +2556,7 @@
    "IRQ %d)\n", ioapic,
    mp_ioapics[ioapic].mpc_apicid, pin, entry.vector, irq);

- irq_desc[irq].handler = &ioapic_level_irq_type;
-
- set_intr_gate(entry.vector, interrupt[irq]);
+ ioapic_register_intr(irq, entry.vector, IOAPIC_LEVEL);

    if (!ioapic && (irq < 16))
        disable_8259A_irq(irq);
diff -X excludes -urN linux-2.6.0-test2/arch/i386/kernel/mpparse.c
linux-2.6.0-test2-create-vectorbase/arch/i386/kernel/mpparse.c
--- linux-2.6.0-test2/arch/i386/kernel/mpparse.c 2003-07-27 12:59:51.000000000 -0400
+++ linux-2.6.0-test2-create-vectorbase/arch/i386/kernel/mpparse.c 2003-08-12 14:39:06.596668760
-0400
@@ -1124,14 +1124,20 @@
    if ((1<<bit) & mp_ioapic_routing[ioapic].pin_programmed[idx]) {
        printk(KERN_DEBUG "Pin %d-%d already programmed\n",
            mp_ioapic_routing[ioapic].apic_id, ioapic_pin);
- entry->irq = irq;
+ if (use_pci_vector() && !platform_legacy_irq(irq))

```

## Linux-Kernel: Update MSI Patches

```

+ entry->irq = IO_APIC_VECTOR(irq);
+ else
+ entry->irq = irq;
      continue;
    }

    mp_ioapic_routing[ioapic].pin_programmed[idx] |= (1<<bit);

    if (!io_apic_set_pci_routing(ioapic, ioapic_pin, irq))
- entry->irq = irq;
+ if (use_pci_vector() && !platform_legacy_irq(irq))
+ entry->irq = IO_APIC_VECTOR(irq);
+ else
+ entry->irq = irq;

    printk(KERN_DEBUG "%02x:%02x:%02x[%c] -> %d-%d -> IRQ %d\n",
            entry->id.segment, entry->id.bus,
diff -X excludes -urN linux-2.6.0-test2/arch/i386/pci/irq.c
linux-2.6.0-test2-create-vectorbase/arch/i386/pci/irq.c
--- linux-2.6.0-test2/arch/i386/pci/irq.c 2003-07-27 13:11:50.000000000 -0400
+++ linux-2.6.0-test2-create-vectorbase/arch/i386/pci/irq.c 2003-08-12 14:38:09.893288984 -0400
@@ -743,6 +743,10 @@
                bridge->bus->number, PCI_SLOT(bridge->devfn), pin, irq);
            }
            if (irq >= 0) {
+ if (use_pci_vector() &&
+ !platform_legacy_irq(irq))
+ irq = IO_APIC_VECTOR(irq);
+
                printk(KERN_INFO "PCI->APIC IRQ transform: (B%d,I%d,P%d) -> %d\n",
                        dev->bus->number, PCI_SLOT(dev->devfn), pin, irq);
                dev->irq = irq;
diff -X excludes -urN linux-2.6.0-test2/include/asm-i386/hw_irq.h
linux-2.6.0-test2-create-vectorbase/include/asm-i386/hw_irq.h
--- linux-2.6.0-test2/include/asm-i386/hw_irq.h 2003-07-27 13:11:11.000000000 -0400
+++ linux-2.6.0-test2-create-vectorbase/include/asm-i386/hw_irq.h 2003-08-05 09:25:54.000000000
-0400
@@ -41,6 +41,7 @@
extern asmlinkage void error_interrupt(void);
extern asmlinkage void spurious_interrupt(void);
extern asmlinkage void thermal_interrupt(struct pt_regs);
+#define platform_legacy_irq(irq) ((irq) < 16)
#endif

extern void mask_irq(unsigned int irq);
diff -X excludes -urN linux-2.6.0-test2/include/asm-i386/io_apic.h
linux-2.6.0-test2-create-vectorbase/include/asm-i386/io_apic.h
--- linux-2.6.0-test2/include/asm-i386/io_apic.h 2003-07-27 13:04:51.000000000 -0400
+++ linux-2.6.0-test2-create-vectorbase/include/asm-i386/io_apic.h 2003-08-12 11:48:10.000000000
-0400
@@ -11,6 +11,46 @@

```

## Linux–Kernel: Update MSI Patches

\* Copyright (C) 1997, 1998, 1999, 2000 Ingo Molnar

\*/

```
+#ifndef CONFIG_PCI_USE_VECTOR
+static inline int use_pci_vector(void) {return 1;}
+static inline void disable_edge_ioapic_vector(unsigned int vector) { }
+static inline void mask_and_ack_level_ioapic_vector(unsigned int vector) { }
+static inline void end_edge_ioapic_vector (unsigned int vector) { }
+#define startup_level_ioapic startup_level_ioapic_vector
+#define shutdown_level_ioapic mask_IO_APIC_vector
+#define enable_level_ioapic unmask_IO_APIC_vector
+#define disable_level_ioapic mask_IO_APIC_vector
+#define mask_and_ack_level_ioapic mask_and_ack_level_ioapic_vector
+#define end_level_ioapic end_level_ioapic_vector
+#define set_ioapic_affinity set_ioapic_affinity_vector
+
+#define startup_edge_ioapic startup_edge_ioapic_vector
+#define shutdown_edge_ioapic disable_edge_ioapic_vector
+#define enable_edge_ioapic unmask_IO_APIC_vector
+#define disable_edge_ioapic disable_edge_ioapic_vector
+#define ack_edge_ioapic ack_edge_ioapic_vector
+#define end_edge_ioapic end_edge_ioapic_vector
+#else
+static inline int use_pci_vector(void) {return 0;}
+static inline void disable_edge_ioapic_irq(unsigned int irq) { }
+static inline void mask_and_ack_level_ioapic_irq(unsigned int irq) { }
+static inline void end_edge_ioapic_irq (unsigned int irq) { }
+#define startup_level_ioapic startup_level_ioapic_irq
+#define shutdown_level_ioapic mask_IO_APIC_irq
+#define enable_level_ioapic unmask_IO_APIC_irq
+#define disable_level_ioapic mask_IO_APIC_irq
+#define mask_and_ack_level_ioapic mask_and_ack_level_ioapic_irq
+#define end_level_ioapic end_level_ioapic_irq
+#define set_ioapic_affinity set_ioapic_affinity_irq
+
+#define startup_edge_ioapic startup_edge_ioapic_irq
+#define shutdown_edge_ioapic disable_edge_ioapic_irq
+#define enable_edge_ioapic unmask_IO_APIC_irq
+#define disable_edge_ioapic disable_edge_ioapic_irq
+#define ack_edge_ioapic ack_edge_ioapic_irq
+#define end_edge_ioapic end_edge_ioapic_irq
+#endif
+
+#ifdef CONFIG_X86_IO_APIC

#define APIC_MISMATCH_DEBUG
diff -X excludes -urN linux-2.6.0-test2/include/asm-i386/mach-default/irq_vectors.h
linux-2.6.0-test2-create-vectorbase/include/asm-i386/mach-default/irq_vectors.h
--- linux-2.6.0-test2/include/asm-i386/mach-default/irq_vectors.h 2003-07-27 12:58:54.000000000
-0400
+++ linux-2.6.0-test2-create-vectorbase/include/asm-i386/mach-default/irq_vectors.h 2003-08-12
```

## Linux-Kernel: Update MSI Patches

11:37:19.000000000 -0400

@@ -76,9 +76,19 @@

\* Since vectors 0x00-0x1f are used/reserved for the CPU,

\* the usable vector space is 0x20-0xff (224 vectors)

\*/

+/\*

+ \* The maximum number of vectors supported by i386 processors

+ \* is limited to 256. For processors other than i386, NR\_VECTORS

+ \* should be changed accordingly.

+ \*/

+#define NR\_VECTORS 256

#ifdef CONFIG\_X86\_IO\_APIC

+#ifndef CONFIG\_PCI\_USE\_VECTOR

#define NR\_IRQS 224

#else

+#define NR\_IRQS FIRST\_SYSTEM\_VECTOR

+#endif

+#else

#define NR\_IRQS 16

#endif

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>