

PATCH: Add support for IT8212 IDE controllers

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-08/0007.html>

From: Alan Cox (alan_at_redhat.com)

Date: 08/01/04

Date: Sat, 31 Jul 2004 19:22:27 -0400
To: linux-kernel@vger.kernel.org, linux-ide@vger.kernel.org

There is a messy scsi faking vendor driver for this card but this instead is a standard Linux IDE layer driver.

```
diff -u --new-file --recursive --exclude-from /usr/src/exclude linux.vanilla-2.6.8-rc2/drivers/ide/Kconfig
linux-2.6.8-rc2/drivers/ide/Kconfig
--- linux.vanilla-2.6.8-rc2/drivers/ide/Kconfig 2004-07-27 19:22:42.000000000 +0100
+++ linux-2.6.8-rc2/drivers/ide/Kconfig 2004-07-31 17:15:35.000000000 +0100
@@ -621,6 +621,12 @@
     <http://www.ite.com.tw/ia/brief\_it8172bsp.htm>; picture of the
     board at <http://www.mvista.com/partners/semiconductor/ite.html>.
```

```
+config BLK_DEV_IT8212
+ tristate "IT8212 IDE support (Experimental)"
+ help
+ This driver adds support for the ITE 8212 IDE RAID controller in
+ both RAID and pass-through mode.
+
```

```
config BLK_DEV_NS87415
    tristate "NS87415 chipset support"
    help
```

```
diff -u --new-file --recursive --exclude-from /usr/src/exclude
linux.vanilla-2.6.8-rc2/drivers/ide/pci/it8212.c linux-2.6.8-rc2/drivers/ide/pci/it8212.c
--- linux.vanilla-2.6.8-rc2/drivers/ide/pci/it8212.c 1970-01-01 01:00:00.000000000 +0100
+++ linux-2.6.8-rc2/drivers/ide/pci/it8212.c 2004-07-31 22:18:14.000000000 +0100
@@ -0,0 +1,664 @@
```

```
+/ *
+ * linux/drivers/ide/pci/it8212.c Version 0.01 July 2004
+ *
+ * Copyright (C) 2004 Red Hat <alan@redhat.com>
+ *
+ * May be copied or modified under the terms of the GNU General Public License
+ *
+ * Documentation available from
+ * http://www.ite.com.tw/pc/IT8212F\_V04.pdf
+ *
+ * The ITE8212 isn't exactly a standard IDE controller. It has two
+ * modes. In pass through mode then it is an IDE controller. In its smart
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+ * mode its actually quite a capable hardware raid controller disguised
+ * as an IDE controller.
+ *
+ * Errata:
+ * Rev 0x10 also requires master/slave use the same UDMA timing and
+ * cannot do ATAPI DMA, while the other revisions can do ATAPI UDMA
+ * but not MWDMA.
+ *
+ * This has a few impacts on the driver
+ * - In pass through mode we do all the work you would expect
+ * - In smart mode the clocking set up is done by the controller generally
+ * - There are a few extra vendor commands that actually talk to the
+ * controller but only work PIO with no IRQ.
+ *
+ * Vendor areas of the identify block in smart mode are used for the
+ * timing and policy set up. Each HDD in raid mode also has a serial
+ * block on the disk. The hardware extra commands are get/set chip status,
+ * rebuild, get rebuild status.
+ *
+ * In Linux the driver supports pass through mode as if the device was
+ * just another IDE controller. If the smart mode is running then
+ * volumes are managed by the controller firmware and each IDE "disk"
+ * is a raid volume. Even more cute - the controller can do automated
+ * hotplug and rebuild.
+ *
+ * The pass through controller itself is a little demented. It has a
+ * flaw that it has a single set of PIO/MWDMA timings per channel so
+ * non UDMA devices restrict each others performance. It also has a
+ * single clock source per channel so mixed UDMA100/133 performance
+ * isn't perfect and we have to pick a clock. Thankfully none of this
+ * matters in smart mode. ATAPI DMA is not supported.
+ *
+ * TODO
+ * - Rev 0x10 in pass through mode needs UDMA clock whacking
+ * to work around h/w issues
+ * - Is rev 0x10 out anywhere - test it if so
+ * - More testing
+ * - Find an Innovision 8401D or R board somewhere to test that
+ * - See if we can kick the cheapo board into smart mode
+ * ourselves 8)
+ * - ATAPI UDMA is ok but not MWDMA it seems
+ * - Better clock strategy
+ * - RAID configuration ioctls
+ */
+
+#include <linux/config.h>
+#include <linux/types.h>
+#include <linux/module.h>
+#include <linux/pci.h>
+#include <linux/delay.h>
+#include <linux/hdreg.h>
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+#include <linux/ide.h>
+#include <linux/init.h>
+
+#include <asm/io.h>
+
+struct it8212_dev
+{
+ int smart:1, /* Are we in smart raid mode */
+ timing10:1; /* Rev 0x10 */
+ u8 clock_mode; /* 0, 50 or 66 */
+ u8 want[2][2]; /* Mode/Pri log for master slave */
+
+ /* We need these for switching the clock when DMA goes on/off */
+ u8 pio[2]; /* Cached PIO values */
+ u8 mwdma[2]; /* Cached MWDMA values */
+};
+
+/**
+ * it8212_clock_strategy
+ * @hwif: hardware interface
+ *
+ * Select between the 50 and 66Mhz base clocks to get the best
+ * results for this interface. Right now our approach is stupidity
+ * first until we have the driver stable
+ */
+
+static void it8212_clock_strategy(ide_drive_t *drive)
+{
+ u8 v;
+ int sel = 0;
+ ide_hwif_t *hwif = HWIF(drive);
+ int clock;
+ struct it8212_dev *itdev = ide_get_hwifdata(hwif);
+ /* FIXME: Just to simplify we don't switch clocks once set */
+ if(itdev->clock_mode == 0)
+ {
+ if(itdev->want[0][0] > itdev->want[1][0])
+ clock = itdev->want[0][1];
+ else
+ clock = itdev->want[1][1];
+ /* Load this into the controller ? */
+ if(clock == 66)
+ itdev->clock_mode = 66;
+ else
+ {
+ itdev->clock_mode = 50;
+ sel = 1;
+ }
+ pci_read_config_byte(hwif->pci_dev, 0x50, &v);
+ v &= ~(1 << (1 + hwif->channel));
+ v |= sel << (1 + hwif->channel);
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+ pci_write_config_byte(hwif->pci_dev, 0x50, v);
+
+ printk(KERN_INFO "it8212: selected %dMHz clock.\n", itdev->clock_mode);
+ }
+ }
+
+/**
+ * it8212_ratemask - Compute available modes
+ * @drive: IDE drive
+ *
+ * Compute the available speeds for the devices on the interface. This
+ * is all modes to ATA133 clipped by drive cable setup.
+ */
+
+static byte it8212_ratemask (ide_drive_t *drive)
+{
+ u8 mode = 4;
+ if (!eighty_ninty_three(drive))
+ mode = min(mode, (u8)1);
+ return mode;
+}
+
+/**
+ * it8212_tuneproc - tune a drive
+ * @drive: drive to tune
+ * @mode_wanted: the target operating mode
+ *
+ * Load the timing settings for this device mode into the
+ * controller. By the time we are called the mode has been
+ * modified as necessary to handle the absence of separate
+ * master/slave timers for MWDMA/PIO.
+ *
+ * This code is only used in pass through mode.
+ */
+
+static void it8212_tuneproc (ide_drive_t *drive, byte mode_wanted)
+{
+ ide_hwif_t *hwif = HWIF(drive);
+ struct it8212_dev *itdev = ide_get_hwifdata(hwif);
+ int unit = drive->select.b.unit;
+ int channel = hwif->channel;
+ u8 conf;
+
+ /* Spec says 89 ref driver uses 88 */
+ static u8 pio_50[] = { 0x88, 0x82, 0x81, 0x32, 0x21 };
+ static u8 pio_66[] = { 0xAA, 0xA3, 0xA1, 0x33, 0x31 };
+
+ static u8 pio_want[] = { 66, 66, 66, 66, 0 };
+
+ /* We prefer 66Mhz clock for PIO 0-3, don't care for PIO4 */
+ itdev->want[unit][0] = pio_want[mode_wanted];
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+ itdev->want[unit][1] = 1; /* PIO is lowest priority */
+
+ it8212_clock_strategy(drive);
+
+ /* Program PIO/MWDMA timing bits */
+ pci_read_config_byte(hwif->pci_dev, 0x54 + 4 * channel, &conf);
+ if(itdev->clock_mode == 66)
+ conf = pio_66[mode_wanted];
+ else
+ conf = pio_50[mode_wanted];
+ pci_write_config_byte(hwif->pci_dev, 0x54 + 4 * channel, conf);
+ itdev->pio[unit] = conf ;
+ }
+
+/**
+ * it8212_tune_mwdma - tune a channel for MWDMA
+ * @drive: drive to set up
+ * @mode_wanted: the target operating mode
+ *
+ * Load the timing settings for this device mode into the
+ * controller when doing MWDMA in pass through mode. The caller
+ * must manage the whole lack of per device MWDMA/PIO timings and
+ * the shared MWDMA/PIO timing register.
+ *
+ * FIXME: before we can enable MWDMA we need to cache PIO/MWDMA
+ * timings and reload them on ide_dma_on/ide_dma_off_* because they
+ * occupy the same register.
+ */
+
+static void it8212_tune_mwdma (ide_drive_t *drive, byte mode_wanted)
+{
+ ide_hwif_t *hwif = HWIF(drive);
+ struct it8212_dev *itdev = (void *)ide_get_hwifdata(hwif);
+ int unit = drive->select.b.unit;
+ int channel = hwif->channel;
+ u8 conf;
+
+ static u8 dma_50[] = { 0x66, 0x22, 0x21 };
+ static u8 dma_66[] = { 0x88, 0x32, 0x31 };
+
+ static u8 mwdma_want[] = { 0, 66, 0 };
+
+ itdev->want[unit][0] = mwdma_want[mode_wanted];
+ itdev->want[unit][1] = 2; /* MWDMA is low priority */
+
+ it8212_clock_strategy(drive);
+
+ /* UDMA bits off */
+ pci_read_config_byte(hwif->pci_dev, 0x50, &conf);
+ conf |= 1 << (3 + 2 * channel + unit);
+ pci_write_config_byte(hwif->pci_dev, 0x50, conf);
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+
+ /* Program PIO/MWDMA timing bits */
+ pci_read_config_byte(hwif->pci_dev, 0x54 + 4 * channel, &conf);
+ if(itdev->clock_mode == 66)
+ conf = dma_66[mode_wanted];
+ else
+ conf = dma_50[mode_wanted];
+ pci_write_config_byte(hwif->pci_dev, 0x54 + 4 * channel, conf);
+ itdev->mwdma[unit] = conf;
+ }
+
+/**
+ * it8212_tune_udma - tune a channel for UDMA
+ * @drive: drive to set up
+ * @mode_wanted: the target operating mode
+ *
+ * Load the timing settings for this device mode into the
+ * controller when doing UDMA modes in pass through.
+ */
+
+static void it8212_tune_udma (ide_drive_t *drive, byte mode_wanted)
+{
+ ide_hwif_t *hwif = HWIF(drive);
+ struct it8212_dev *itdev = ide_get_hwifdata(hwif);
+ int unit = drive->select.b.unit;
+ int channel = hwif->channel;
+ u8 conf;
+
+ static u8 udma_50[] = { 0x33, 0x31, 0x21, 0x21, 0x11, 0x11, 0x11 };
+ static u8 udma_66[] = { 0x44, 0x42, 0x31, 0x21, 0x11, 0x22, 0x11 };
+
+ static u8 udma_want[] = { 0, 50, 0, 66, 66, 50, 66 };
+
+ itdev->want[unit][0] = udma_want[mode_wanted];
+ itdev->want[unit][1] = 3; /* UDMA is high priority */
+
+ it8212_clock_strategy(drive);
+
+ /* UDMA on */
+ pci_read_config_byte(hwif->pci_dev, 0x50, &conf);
+ conf &= ~ (1 << (3 + 2 * channel + unit));
+ pci_write_config_byte(hwif->pci_dev, 0x50, conf);
+
+ /* Program UDMA timing bits */
+ pci_read_config_byte(hwif->pci_dev, 0x56 + 4 * channel + unit, &conf);
+ if(itdev->clock_mode == 66)
+ conf = udma_66[mode_wanted];
+ else
+ conf = udma_50[mode_wanted];
+
+ if(mode_wanted >= 5)
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+ conf |= 0x80; /* UDMA 5/6 select on */
+
+ pci_write_config_byte(hwif->pci_dev, 0x56 + 4 * channel + unit, conf);
+ itdev->mwdma[unit] = 0;
+}
+
+/**
+ * config_it8212_chipset_for_pio - set drive timings
+ * @drive: drive to tune
+ * @speed we want
+ *
+ * Compute the best pio mode we can for a given device. We must
+ * pick a speed that does not cause problems with the other device
+ * on the cable.
+ */
+
+static void config_it8212_chipset_for_pio (ide_drive_t *drive, byte set_speed)
+{
+ u8 unit = drive->select.b.unit;
+ ide_hwif_t *hwif = HWIF(drive);
+ ide_drive_t *pair = &hwif->drives[1-unit];
+ u8 speed = 0, set_pio = ide_get_best_pio_mode(drive, 4, 5, NULL);
+ u8 pair_pio;
+
+ /* We have to deal with this mess in pairs */
+ if(pair != NULL)
+ {
+ pair_pio = ide_get_best_pio_mode(pair, 4, 5, NULL);
+ /* Trim PIO to the slowest of the master/slave */
+ if(pair_pio < set_pio)
+ set_pio = pair_pio;
+ }
+ it8212_tuneproc(drive, set_pio);
+ speed = XFER_PIO_0 + set_pio;
+ /* XXX - We trim to the lowest of the pair so the other drive
+ will always be fine at this point until we do hotplug passthru */
+
+ if (set_speed)
+ (void) ide_config_drive_speed(drive, speed);
+}
+
+static void config_chipset_for_pio (ide_drive_t *drive, byte set_speed)
+{
+ config_it8212_chipset_for_pio(drive, set_speed);
+}
+
+/**
+ * it8212_dma_read - DMA hook
+ * @drive: drive for DMA
+ *
+ * The IT8212 has a single timing register for MWDMA and for PIO
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+ * operations. As we flip back and forth we have to reload the
+ * clock.
+ *
+ * FIXME: for 0x10 model we should reprogram UDMA here
+ */
+
+static int it8212_dma_begin(ide_drive_t *drive)
+{
+ ide_hwif_t *hwif = HWIF(drive);
+ struct it8212_dev *itdev = ide_get_hwifdata(hwif);
+ int unit = drive->select.b.unit;
+ if(itdev->mwdma[unit])
+ pci_write_config_byte(hwif->pci_dev, 0x54 + 4 * hwif->channel, itdev->mwdma[drive->select.b.unit]);
+ return __ide_dma_begin(drive);
+}
+
+/**
+ * it8212_dma_write - DMA hook
+ * @drive: drive for DMA stop
+ *
+ * The IT8212 has a single timing register for MWDMA and for PIO
+ * operations. As we flip back and forth we have to reload the
+ * clock.
+ *
+ * FIXME: for 0x10 model we should reprogram UDMA here
+ */
+
+static int it8212_dma_end(ide_drive_t *drive)
+{
+ ide_hwif_t *hwif = HWIF(drive);
+ int unit = drive->select.b.unit;
+ struct it8212_dev *itdev = ide_get_hwifdata(hwif);
+ int ret = __ide_dma_end(drive);
+
+ if(itdev->mwdma[unit])
+ pci_write_config_byte(hwif->pci_dev, 0x54 + 4 * hwif->channel, itdev->pio[drive->select.b.unit]);
+ return ret;
+}
+
+/**
+ * it8212_tune_chipset - set controller timings
+ * @drive: Drive to set up
+ * @xferspeed: speed we want to achieve
+ *
+ * Tune the ITE chipset for the desired mode. If we can't achieve
+ * the desired mode then tune for a lower one, but ultimately
+ * make the thing work.
+ */
+
+static int it8212_tune_chipset (ide_drive_t *drive, byte xferspeed)
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+{
+
+ ide_hwif_t *hwif = HWIF(drive);
+ struct it8212_dev *itdev = ide_get_hwifdata(hwif);
+ u8 speed = ide_rate_filter(it8212_ratemask(drive), xferspeed);
+
+ switch(speed) {
+ case XFER_PIO_4:
+ case XFER_PIO_3:
+ case XFER_PIO_2:
+ case XFER_PIO_1:
+ case XFER_PIO_0:
+ if(!itdev->smart)
+ it8212_tuneproc(drive, (speed - XFER_PIO_0));
+ break;
+ /* MWDMA tuning is really hard because our MWDMA and PIO
+ timings are kept in the same place. We can switch in the
+ host dma on/off callbacks */
+ case XFER_MW_DMA_2:
+ case XFER_MW_DMA_1:
+ case XFER_MW_DMA_0:
+ if(!itdev->smart)
+ it8212_tune_mwdma(drive, (speed - XFER_MW_DMA_0));
+ break;
+ case XFER_UDMA_6:
+ case XFER_UDMA_5:
+ case XFER_UDMA_4:
+ case XFER_UDMA_3:
+ case XFER_UDMA_2:
+ case XFER_UDMA_1:
+ case XFER_UDMA_0:
+ if(!itdev->smart)
+ it8212_tune_udma(drive, (speed - XFER_UDMA_0));
+ break;
+ default:
+ return 1;
+ }
+ /*
+ * In smart mode the clocking is done by the host controller
+ * snooping the mode we picked. The rest of it is not our problem
+ */
+ return (ide_config_drive_speed(drive, speed));
+}
+
+/**
+ * config_chipset_for_dma - configure for DMA
+ * @drive: drive to configure
+ *
+ * Called by the IDE layer when it wants the timings set up.
+ */
+
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+static int config_chipset_for_dma (ide_drive_t *drive)
+{
+ u8 speed = ide_dma_speed(drive, it8212_ratemask(drive));
+
+ config_chipset_for_pio(drive, !speed);
+
+ if (!speed)
+ return 0;
+
+ if (ide_set_xfer_rate(drive, speed))
+ return 0;
+
+ if (!drive->init_speed)
+ drive->init_speed = speed;
+
+ return ide_dma_enable(drive);
+}
+
+/**
+ * it8212_configure_drive_for_dma – set up for DMA transfers
+ * @drive: drive we are going to set up
+ *
+ * Set up the drive for DMA, tune the controller and drive as
+ * required. If the drive isn't suitable for DMA or we hit
+ * other problems then we will drop down to PIO and set up
+ * PIO appropriately
+ */
+static int it8212_config_drive_for_dma (ide_drive_t *drive)
+{
+ ide_hwif_t *hwif = HWIF(drive);
+ struct hd_driveid *id = drive->id;
+
+ if ((id->capability & 1) != 0 && drive->autodma) {
+ /* Consult the list of known "bad" drives */
+ if (__ide_dma_bad_drive(drive))
+ goto fast_ata_pio;
+
+ if ((id->field_valid & 4) && it8212_ratemask(drive)) {
+ if (id->dma_ultra & hwif->ultra_mask) {
+ /* Force if Capable UltraDMA */
+ int dma = config_chipset_for_dma(drive);
+ if ((id->field_valid & 2) && !dma)
+ goto try_dma_modes;
+ }
+ } else if (id->field_valid & 2) {
+ try_dma_modes:
+ if ((id->dma_mword & hwif->mwdma_mask) ||
+ (id->dma_1word & hwif->swdma_mask)) {
+ /* Force if Capable regular DMA modes */
+ if (!config_chipset_for_dma(drive))
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+ goto no_dma_set;
+ }
+ } else if (__ide_dma_good_drive(drive) &&
+ (id->eide_dma_time < 150)) {
+ /* Consult the list of known "good" drives */
+ if (!config_chipset_for_dma(drive))
+ goto no_dma_set;
+ } else {
+ goto fast_ata_pio;
+ }
+ return hwif->ide_dma_on(drive);
+ } else if ((id->capability & 8) || (id->field_valid & 2)) {
+fast_ata_pio:
+no_dma_set:
+ config_chipset_for_pio(drive, 1);
+ return hwif->ide_dma_off_quietly(drive);
+ }
+ /* IORDY not supported */
+ return 0;
+ }
+
+/**
+ * init_chipset_it8212 - set up an ITE device
+ * @dev: PCI device
+ * @name: device name
+ *
+ */
+
+static unsigned int __devinit init_chipset_it8212(struct pci_dev *dev, const char *name)
+{
+ return 0;
+ }
+
+/**
+ * ata66_it8212 - check for 80 pin cable
+ * @hwif: interface to check
+ *
+ * Check for the presence of an ATA66 capable cable on the
+ * interface. Note that the firmware writes bits 4-7 having done
+ * the drive side sampling. This may impact hotplug.
+ */
+
+static unsigned int __devinit ata66_it8212(ide_hwif_t *hwif)
+{
+ u16 iocfg;
+
+ pci_read_config_word(hwif->pci_dev, 0x40, &iocfg);
+ if(iocfg & (1 << (4 + 2*hwif->channel)))
+ return 1;
+ return 0;
+ }
```


Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```
+ hwif->speedproc = &it8212_tune_chipset;
+ hwif->tuneproc = &it8212_tuneproc;
+
+ /* MWDMA/PIO clock switching for pass through mode */
+ if(!idev->smart)
+ {
+ hwif->ide_dma_begin = &it8212_dma_begin;
+ hwif->ide_dma_end = &it8212_dma_end;
+ }
+
+ if (!hwif->dma_base)
+ goto fallback;
+
+ hwif->ultra_mask = 0x7f;
+ hwif->mwdma_mask = 0x07;
+ hwif->swdma_mask = 0x07;
+
+
+ hwif->ide_dma_check = &it8212_config_drive_for_dma;
+ if (!(hwif->udma_four))
+ hwif->udma_four = ata66_it8212(hwif);
+
+ /*
+ * The BIOS often doesn't set up DMA on this controller
+ * so we always do it.
+ */
+
+ hwif->autodma = 1;
+ hwif->drives[0].autodma = hwif->autodma;
+ hwif->drives[1].autodma = hwif->autodma;
+ return;
+
+fallback:
+ hwif->autodma = 0;
+ hwif->drives[0].autotune = 1;
+ hwif->drives[1].autotune = 1;
+ return;
+}
+
+#define DECLARE_ITE_DEV(name_str) \
+ { \
+ .name = name_str, \
+ .init_chipset = init_chipset_it8212, \
+ .init_hwif = init_hwif_it8212, \
+ .channels = 2, \
+ .autodma = AUTODMA, \
+ .bootable = ON_BOARD, \
+ }
+
+static ide_pci_device_t it8212_chipsets[] __devinitdata = {
+ /* 0 */ DECLARE_ITE_DEV("IT8212"),
```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

```

+};
+
+/**
+ * it8212_init_one - pci layer discovery entry
+ * @dev: PCI device
+ * @id: ident table entry
+ *
+ * Called by the PCI code when it finds an ITE8212 controller.
+ * We then use the IDE PCI generic helper to do most of the work.
+ */
+
+static int __devinit it8212_init_one(struct pci_dev *dev, const struct pci_device_id *id)
+{
+ ide_setup_pci_device(dev, &it8212_chipsets[id->driver_data]);
+ return 0;
+}
+
+static struct pci_device_id it8212_pci_tbl[] = {
+ { PCI_VENDOR_ID_ITE, PCI_DEVICE_ID_ITE_8212, PCI_ANY_ID, PCI_ANY_ID, 0, 0, 0},
+ { 0, },
+};
+
+MODULE_DEVICE_TABLE(pci, it8212_pci_tbl);
+
+static struct pci_driver driver = {
+ .name = "ITE8212 IDE",
+ .id_table = it8212_pci_tbl,
+ .probe = it8212_init_one,
+};
+
+static int it8212_ide_init(void)
+{
+ return ide_pci_register_driver(&driver);
+}
+
+module_init(it8212_ide_init);
+
+MODULE_AUTHOR("Alan Cox");
+MODULE_DESCRIPTION("PCI driver module for the ITE 8212");
+MODULE_LICENSE("GPL");
diff -u --new-file --recursive --exclude-from /usr/src/exclude
linux.vanilla-2.6.8-rc2/drivers/ide/pci/Makefile linux-2.6.8-rc2/drivers/ide/pci/Makefile
--- linux.vanilla-2.6.8-rc2/drivers/ide/pci/Makefile 2004-07-27 19:21:37.000000000 +0100
+++ linux-2.6.8-rc2/drivers/ide/pci/Makefile 2004-07-31 17:16:01.000000000 +0100
@@ -13,6 +13,7 @@
obj-$(CONFIG_BLK_DEV_HPT366) += hpt366.o
#obj-$(CONFIG_BLK_DEV_HPT37X) += hpt37x.o
obj-$(CONFIG_BLK_DEV_IT8172) += it8172.o
+obj-$(CONFIG_BLK_DEV_IT8212) += it8212.o
obj-$(CONFIG_BLK_DEV_NS87415) += ns87415.o
obj-$(CONFIG_BLK_DEV_OPTI621) += opti621.o

```

Linux-Kernel: PATCH: Add support for IT8212 IDE controllers

obj-\$(CONFIG_BLK_DEV_PDC202XX_OLD) += pdc202xx_old.o

----- End forwarded message -----

--
--

"Have you noticed the way people's intelligence capabilities decline
sharply the minute they start waving guns around?"

-- Dr. Who

-

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/>