

[RFC][PATCH 2.6.13] Marvell SATA support (PIO mode)

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-08/7675.html>

From: Brett Russ (*russb_at_emc.com*)

Date: 08/30/05

To: Jeff Garzik <jgarzik@pobox.com>

Date: Tue, 30 Aug 2005 14:36:25 -0400 (EDT)

This is the first public release of my libata compatible low level driver for the Marvell SATA family. Currently it successfully runs in PIO mode on a 6081 chip. EDMA support is in the works and should be done shortly. Review, testing (especially on other flavors of Marvell), comments welcome.

Thank you,
BR

Index: linux-2.6.13/drivers/scsi/sata_mv.c

=====

--- /dev/null

+++ linux-2.6.13/drivers/scsi/sata_mv.c

@@ -0,0 +1,825 @@

+/*

+ * sata_mv.c - Marvell SATA support

+ *

+ * Copyright 2005: EMC Corporation, all rights reserved.

+ *

+ * Please ALWAYS copy linux-ide@vger.kernel.org on emails.

+ *

+ * This program is free software; you can redistribute it and/or modify

+ * it under the terms of the GNU General Public License as published by

+ * the Free Software Foundation; version 2 of the License.

+ *

+ * This program is distributed in the hope that it will be useful,

+ * but WITHOUT ANY WARRANTY; without even the implied warranty of

+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

+ * GNU General Public License for more details.

+ *

+ * You should have received a copy of the GNU General Public License

+ * along with this program; if not, write to the Free Software

+ * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

+ *

+ */

+

Linux-Kernel: [RFC][PATCH 2.6.13] Marvell SATA support (PIO mode)

```
+#include <linux/kernel.h>
+#include <linux/module.h>
+#include <linux/pci.h>
+#include <linux/init.h>
+#include <linux/blkdev.h>
+#include <linux/delay.h>
+#include <linux/interrupt.h>
+#include <linux/sched.h>
+#include <linux/dma-mapping.h>
+#include "scsi.h"
+#include <scsi/scsi_host.h>
+#include <linux/libata.h>
+#include <asm/io.h>
+
+#define DRV_NAME "sata_mv"
+#define DRV_VERSION "0.10"
+
+enum {
+ /* BAR's are enumerated in terms of pci_resource_start() terms */
+ MV_PRIMARY_BAR = 0, /* offset 0x10: memory space */
+ MV_IO_BAR = 2, /* offset 0x18: IO space */
+ MV_MISC_BAR = 3, /* offset 0x1c: FLASH, NVRAM, SRAM */
+
+ MV_MAJOR_REG_AREA_SZ = 0x10000, /* 64KB */
+ MV_MINOR_REG_AREA_SZ = 0x2000, /* 8KB */
+
+ MV_PCI_REG_BASE = 0,
+ MV_IRQ_COAL_REG_BASE = 0x18000, /* 6xxx part only */
+ MV_SATAHC0_REG_BASE = 0x20000,
+
+ MV_PCI_REG_SZ = MV_MAJOR_REG_AREA_SZ,
+ MV_SATAHC_REG_SZ = MV_MAJOR_REG_AREA_SZ,
+ MV_SATAHC_ARBTR_REG_SZ = MV_MINOR_REG_AREA_SZ, /* arbiter */
+ MV_PORT_REG_SZ = MV_MINOR_REG_AREA_SZ,
+
+ MV_Q_CT = 32,
+ MV_CRQB_SZ = 32,
+ MV_CRPB_SZ = 8,
+
+ MV_DMA_BOUNDARY = 0xffffffffU,
+ SATAHC_MASK = (~(MV_SATAHC_REG_SZ - 1)),
+
+ MV_PORTS_PER_HC = 4,
+ /* == (port / MV_PORTS_PER_HC) to determine HC from 0-7 port */
+ MV_PORT_HC_SHIFT = 2,
+ /* == (port % MV_PORTS_PER_HC) to determine port from 0-7 port */
+ MV_PORT_MASK = 3,
+
+ MV_FLAG_DUAL_HC = (1 << 30), /* two SATA Host Controllers */
+ MV_FLAG_IRQ_COALESCE = (1 << 29), /* IRQ coalescing capability */
+ MV_FLAG_BDMA = (1 << 28), /* Basic DMA */

```

```

+
+ chip_504x = 0,
+ chip_508x = 1,
+ chip_604x = 2,
+ chip_608x = 3,
+
+ /* PCI interface registers */
+
+ PCI_MAIN_CMD_STS_OFS = 0xd30,
+ STOP_PCI_MASTER = (1 << 2),
+ PCI_MASTER_EMPTY = (1 << 3),
+ GLOB_SFT_RST = (1 << 4),
+
+ PCI_IRQ_CAUSE = 0x1d58,
+ PCI_IRQ_MASK = 0x1d5c,
+ PCI_UNMASK_ALL_IRQS = 0x7ffff, /* bits 22-0 */
+
+ HC_MAIN_IRQ_CAUSE = 0x1d60,
+ PORT0_ERR = (1 << 0), /* shift by port # */
+ PORT0_DONE = (1 << 1), /* shift by port # */
+ HC0_IRQ_PEND = 0x1ff, /* bits 0-8 = HC0's ports */
+ HC_SHIFT = 9, /* bits 9-17 = HC1's ports */
+ PCI_ERR = (1 << 18),
+ TRAN_LO_DONE = (1 << 19), /* 6xxx: IRQ coalescing */
+ TRAN_HI_DONE = (1 << 20), /* 6xxx: IRQ coalescing */
+ PORTS_0_7_COAL_DONE = (1 << 21), /* 6xxx: IRQ coalescing */
+ GPIO_INT = (1 << 22),
+ SELF_INT = (1 << 23),
+ TWSI_INT = (1 << 24),
+ HC_MAIN_RSVD = (0x7f << 25), /* bits 31-25 */
+ HC_MAIN_IRQ_MASK = 0x1d64,
+ HC_MAIN_MASKED_IRQS = (TRAN_LO_DONE | TRAN_HI_DONE |
+ PORTS_0_7_COAL_DONE | GPIO_INT | TWSI_INT |
+ HC_MAIN_RSVD),
+
+ /* SATAHC registers */
+ HC_CFG_OFS = 0,
+
+ HC_IRQ_CAUSE_OFS = 0x14,
+ CRBP_DMA_DONE = (1 << 0), /* shift by port # */
+ HC_IRQ_COAL = (1 << 4), /* IRQ coalescing */
+ DEV_IRQ = (1 << 8), /* shift by port # */
+
+ /* Shadow block registers */
+ SHD_PIO_DATA_OFS = 0x100,
+ SHD_FEA_ERR_OFS = 0x104,
+ SHD_SECT_CNT_OFS = 0x108,
+ SHD_LBA_L_OFS = 0x10c,
+ SHD_LBA_M_OFS = 0x110,
+ SHD_LBA_H_OFS = 0x114,
+ SHD_DEV_HD_OFS = 0x118,

```

Linux-Kernel: [RFC][PATCH 2.6.13] Marvell SATA support (PIO mode)

```
+ SHD_CMD_STA_OFS = 0x11C,
+ SHD_CTL_AST_OFS = 0x120,
+
+ /* SATA registers */
+ SATA_STATUS_OFS = 0x300, /* ctrl, err regs follow status */
+ SATA_ACTIVE_OFS = 0x350,
+
+ /* Port registers */
+ EDMA_CFG_OFS = 0,
+
+ EDMA_ERR_IRQ_CAUSE_OFS = 0x8,
+ EDMA_ERR_IRQ_MASK_OFS = 0xc,
+ EDMA_ERR_D_PAR = (1 << 0),
+ EDMA_ERR_PRD_PAR = (1 << 1),
+ EDMA_ERR_DEV = (1 << 2),
+ EDMA_ERR_DEV_DCON = (1 << 3),
+ EDMA_ERR_DEV_CON = (1 << 4),
+ EDMA_ERR_SERR = (1 << 5),
+ EDMA_ERR_SELF_DIS = (1 << 7),
+ EDMA_ERR_BIST_ASYNC = (1 << 8),
+ EDMA_ERR_CRBQ_PAR = (1 << 9),
+ EDMA_ERR_CRPB_PAR = (1 << 10),
+ EDMA_ERR_INTRL_PAR = (1 << 11),
+ EDMA_ERR_IORDY = (1 << 12),
+ EDMA_ERR_LNK_CTRL_RX = (0xf << 13),
+ EDMA_ERR_LNK_CTRL_RX_2 = (1 << 15),
+ EDMA_ERR_LNK_DATA_RX = (0xf << 17),
+ EDMA_ERR_LNK_CTRL_TX = (0x1f << 21),
+ EDMA_ERR_LNK_DATA_TX = (0x1f << 26),
+ EDMA_ERR_TRANS_PROTO = (1 << 31),
+ EDMA_ERR_FATAL = (EDMA_ERR_D_PAR | EDMA_ERR_PRD_PAR |
+ EDMA_ERR_DEV_DCON | EDMA_ERR_CRBQ_PAR |
+ EDMA_ERR_CRPB_PAR | EDMA_ERR_INTRL_PAR |
+ EDMA_ERR_IORDY | EDMA_ERR_LNK_CTRL_RX_2 |
+ EDMA_ERR_LNK_DATA_RX |
+ EDMA_ERR_LNK_DATA_TX |
+ EDMA_ERR_TRANS_PROTO),
+
+ EDMA_CMD_OFS = 0x28,
+ EDMA_EN = (1 << 0),
+ EDMA_DS = (1 << 1),
+ ATA_RST = (1 << 2),
+
+ /* BDMA is 6xxx part only */
+ BDMA_CMD_OFS = 0x224,
+ BDMA_START = (1 << 0),
+
+ MV_UNDEF = 0,
+};
+
+struct mv_port_priv {
```

```

+
+};
+
+struct mv_host_priv {
+
+};
+
+static void mv_irq_clear(struct ata_port *ap);
+static u32 mv_scr_read(struct ata_port *ap, unsigned int sc_reg_in);
+static void mv_scr_write(struct ata_port *ap, unsigned int sc_reg_in, u32 val);
+static void mv_phy_reset(struct ata_port *ap);
+static int mv_master_reset(void __iomem *mmio_base);
+static irqreturn_t mv_interrupt(int irq, void *dev_instance,
+ struct pt_regs *regs);
+static int mv_init_one(struct pci_dev *pdev, const struct pci_device_id *ent);
+
+static Scsi_Host_Template mv_sht = {
+ .module = THIS_MODULE,
+ .name = DRV_NAME,
+ .ioctl = ata_scsi_ioctl,
+ .queuecommand = ata_scsi_queuecmd,
+ .eh_strategy_handler = ata_scsi_error,
+ .can_queue = ATA_DEF_QUEUE,
+ .this_id = ATA_SHT_THIS_ID,
+ .sg_tablesize = MV_UNDEF,
+ .max_sectors = ATA_MAX_SECTORS,
+ .cmd_per_lun = ATA_SHT_CMD_PER_LUN,
+ .emulated = ATA_SHT_EMULATED,
+ .use_clustering = MV_UNDEF,
+ .proc_name = DRV_NAME,
+ .dma_boundary = MV_DMA_BOUNDARY,
+ .slave_configure = ata_scsi_slave_config,
+ .bios_param = ata_std_bios_param,
+ .ordered_flush = 1,
+};
+
+static struct ata_port_operations mv_ops = {
+ .port_disable = ata_port_disable,
+
+ .tf_load = ata_tf_load,
+ .tf_read = ata_tf_read,
+ .check_status = ata_check_status,
+ .exec_command = ata_exec_command,
+ .dev_select = ata_std_dev_select,
+
+ .phy_reset = mv_phy_reset,
+
+ .qc_prep = ata_qc_prep,
+ .qc_issue = ata_qc_issue_prot,
+
+ .eng_timeout = ata_eng_timeout,

```

```

+
+ .irq_handler = mv_interrupt,
+ .irq_clear = mv_irq_clear,
+
+ .scr_read = mv_scr_read,
+ .scr_write = mv_scr_write,
+
+ .port_start = ata_port_start,
+ .port_stop = ata_port_stop,
+ .host_stop = ata_host_stop,
+};
+
+static struct ata_port_info mv_port_info[] = {
+ { /* chip_504x */
+ .sht = &mv_sht,
+ .host_flags = (ATA_FLAG_SATA | ATA_FLAG_NO_LEGACY |
+ ATA_FLAG_SATA_RESET | ATA_FLAG_MMIO),
+ .pio_mask = 0x1f, /* pio4-0 */
+ .udma_mask = 0, /* 0x7f (udma6-0 disabled for now) */
+ .port_ops = &mv_ops,
+ },
+ { /* chip_508x */
+ .sht = &mv_sht,
+ .host_flags = (ATA_FLAG_SATA | ATA_FLAG_NO_LEGACY |
+ ATA_FLAG_SATA_RESET | ATA_FLAG_MMIO |
+ MV_FLAG_DUAL_HC),
+ .pio_mask = 0x1f, /* pio4-0 */
+ .udma_mask = 0, /* 0x7f (udma6-0 disabled for now) */
+ .port_ops = &mv_ops,
+ },
+ { /* chip_604x */
+ .sht = &mv_sht,
+ .host_flags = (ATA_FLAG_SATA | ATA_FLAG_NO_LEGACY |
+ ATA_FLAG_SATA_RESET | ATA_FLAG_MMIO |
+ MV_FLAG_IRQ_COALESCE | MV_FLAG_BDMA),
+ .pio_mask = 0x1f, /* pio4-0 */
+ .udma_mask = 0, /* 0x7f (udma6-0 disabled for now) */
+ .port_ops = &mv_ops,
+ },
+ { /* chip_608x */
+ .sht = &mv_sht,
+ .host_flags = (ATA_FLAG_SATA | ATA_FLAG_NO_LEGACY |
+ ATA_FLAG_SATA_RESET | ATA_FLAG_MMIO |
+ MV_FLAG_IRQ_COALESCE | MV_FLAG_DUAL_HC |
+ MV_FLAG_BDMA),
+ .pio_mask = 0x1f, /* pio4-0 */
+ .udma_mask = 0, /* 0x7f (udma6-0 disabled for now) */
+ .port_ops = &mv_ops,
+ },
+};
+

```

Linux-Kernel: [RFC][PATCH 2.6.13] Marvell SATA support (PIO mode)

```
+static struct pci_device_id mv_pci_tbl[] = {
+ {PCI_DEVICE(PCI_VENDOR_ID_MARVELL, 0x5040), 0, 0, chip_504x},
+ {PCI_DEVICE(PCI_VENDOR_ID_MARVELL, 0x5041), 0, 0, chip_504x},
+ {PCI_DEVICE(PCI_VENDOR_ID_MARVELL, 0x5080), 0, 0, chip_508x},
+ {PCI_DEVICE(PCI_VENDOR_ID_MARVELL, 0x5081), 0, 0, chip_508x},
+
+ {PCI_DEVICE(PCI_VENDOR_ID_MARVELL, 0x6040), 0, 0, chip_604x},
+ {PCI_DEVICE(PCI_VENDOR_ID_MARVELL, 0x6041), 0, 0, chip_604x},
+ {PCI_DEVICE(PCI_VENDOR_ID_MARVELL, 0x6080), 0, 0, chip_608x},
+ {PCI_DEVICE(PCI_VENDOR_ID_MARVELL, 0x6081), 0, 0, chip_608x},
+ { } /* terminate list */
+};
+
+
+static struct pci_driver mv_pci_driver = {
+ .name = DRV_NAME,
+ .id_table = mv_pci_tbl,
+ .probe = mv_init_one,
+ .remove = ata_pci_remove_one,
+};
+
+/*
+ * Functions
+ */
+
+static inline void writelfl(unsigned long data, void __iomem *addr)
+{
+ writel(data, addr);
+ (void) readl(addr); /* flush */
+}
+
+static inline void __iomem *mv_port_addr_to_hc_base(void __iomem *port_mmio)
+{
+ return ((void __iomem *)((unsigned long)port_mmio &
+ (unsigned long)SATAHC_MASK));
+}
+
+static inline void __iomem *mv_hc_base(void __iomem *base, unsigned int hc)
+{
+ return (base + MV_SATAHC0_REG_BASE + (hc * MV_SATAHC_REG_SZ));
+}
+
+static inline void __iomem *mv_port_base(void __iomem *base, unsigned int port)
+{
+ return (mv_hc_base(base, port >> MV_PORT_HC_SHIFT) +
+ MV_SATAHC_ARBTR_REG_SZ +
+ ((port & MV_PORT_MASK) * MV_PORT_REG_SZ));
+}
+
+static inline void __iomem *mv_ap_base(struct ata_port *ap)
+{
+ return (mv_port_base(ap->host_set->mmio_base, ap->port_no));
+}
```

```

+}
+
+static inline int mv_get_hc_count(unsigned long flags)
+{
+ return ((flags & MV_FLAG_DUAL_HC) ? 2 : 1);
+}
+
+static inline int mv_is_edma_active(struct ata_port *ap)
+{
+ void __iomem *port_mmio = mv_ap_base(ap);
+ return (EDMA_EN & readl(port_mmio + EDMA_CMD_OFS));
+}
+
+static inline int mv_port_bdma_capable(struct ata_port *ap)
+{
+ return (ap->flags & MV_FLAG_BDMA);
+}
+
+static void mv_irq_clear(struct ata_port *ap)
+{
+ return;
+}
+
+static unsigned int mv_scr_offset(unsigned int sc_reg_in)
+{
+ unsigned int ofs;
+
+ switch (sc_reg_in) {
+ case SCR_STATUS:
+ case SCR_CONTROL:
+ case SCR_ERROR:
+ ofs = SATA_STATUS_OFS + (sc_reg_in * sizeof(u32));
+ break;
+ case SCR_ACTIVE:
+ ofs = SATA_ACTIVE_OFS; /* active is not with the others */
+ break;
+ default:
+ ofs = 0xffffffffU;
+ break;
+ }
+ return (ofs);
+}
+
+static u32 mv_scr_read(struct ata_port *ap, unsigned int sc_reg_in)
+{
+ unsigned int ofs = mv_scr_offset(sc_reg_in);
+
+ if (0xffffffffU != ofs) {
+ return (readl(mv_ap_base(ap) + ofs));
+ } else {
+ return ((u32) ofs);
+ }
+}

```

```

+ }
+}
+
+static void mv_scr_write(struct ata_port *ap, unsigned int sc_reg_in, u32 val)
+{
+ unsigned int ofs = mv_scr_offset(sc_reg_in);
+
+ if (0xffffffffU != ofs) {
+ writelfl(val, mv_ap_base(ap) + ofs);
+ }
+}
+
+static int mv_master_reset(void __iomem *mmio_base)
+{
+ void __iomem *reg = mmio_base + PCI_MAIN_CMD_STS_OFS;
+ int i, rc = 0;
+ u32 t;
+
+ VPRINTK("ENTER\n");
+
+ /* Following procedure defined in PCI "main command and status
+ * register" table.
+ */
+ t = readl(reg);
+ writel(t | STOP_PCI_MASTER, reg);
+
+ for (i = 0; i < 100; i++) {
+ msleep(10);
+ t = readl(reg);
+ if (PCI_MASTER_EMPTY & t) {
+ break;
+ }
+ }
+ if (!(PCI_MASTER_EMPTY & t)) {
+ printk(KERN_ERR DRV_NAME "PCI master won't flush\n");
+ rc = 1; /* broken HW? */
+ goto done;
+ }
+
+ /* set reset */
+ i = 5;
+ do {
+ writel(t | GLOB_SFT_RST, reg);
+ t = readl(reg);
+ udelay(1);
+ } while (!(GLOB_SFT_RST & t) && (i-- > 0));
+
+ if (!(GLOB_SFT_RST & t)) {
+ printk(KERN_ERR DRV_NAME "can't set global reset\n");
+ rc = 1; /* broken HW? */
+ goto done;

```

```

+ }
+
+ /* clear reset */
+ i = 5;
+ do {
+ writel(t & ~GLOB_SFT_RST, reg);
+ t = readl(reg);
+ udelay(1);
+ } while ((GLOB_SFT_RST & t) && (i-- > 0));
+
+ if (GLOB_SFT_RST & t) {
+ printk(KERN_ERR DRV_NAME "can't clear global reset\n");
+ rc = 1; /* broken HW? */
+ }
+
+ done:
+ VPRINTK("EXIT, rc = %i\n", rc);
+ return (rc);
+}
+
+static void mv_err_intr(struct ata_port *ap)
+{
+ void __iomem *port_mmio;
+ u32 edma_err_cause, serr = 0;
+
+ /* bug here b/c we got an err int on a port we don't know about,
+ * so there's no way to clear it
+ */
+ BUG_ON(NULL == ap);
+ port_mmio = mv_ap_base(ap);
+
+ edma_err_cause = readl(port_mmio + EDMA_ERR_IRQ_CAUSE_OFS);
+
+ if (EDMA_ERR_SERR & edma_err_cause) {
+ serr = scr_read(ap, SCR_ERROR);
+ scr_write_flush(ap, SCR_ERROR, serr);
+ }
+ DPRINTK("port %u error; EDMA err cause: 0x%08x SERR: 0x%08x\n",
+ ap->port_no, edma_err_cause, serr);
+
+ /* Clear EDMA now that SERR cleanup done */
+ writelfl(0, port_mmio + EDMA_ERR_IRQ_CAUSE_OFS);
+
+ /* check for fatal here and recover if needed */
+ if (EDMA_ERR_FATAL & edma_err_cause) {
+ mv_phy_reset(ap);
+ }
+}
+
+ /* Handle any outstanding interrupts in a single SATAHC
+ */

```

```

+static void mv_host_intr(struct ata_host_set *host_set, u32 relevant,
+ unsigned int hc)
+{
+ void __iomem *mmio = host_set->mmio_base;
+ void __iomem *hc_mmio = mv_hc_base(mmio, hc);
+ struct ata_port *ap;
+ struct ata_queued_cmd *qc;
+ u32 hc_irq_cause;
+ int shift, port, port0, hard_port;
+ u8 ata_status;
+
+ if (hc == 0) {
+ port0 = 0;
+ } else {
+ port0 = MV_PORTS_PER_HC;
+ }
+
+ /* we'll need the HC success int register in most cases */
+ hc_irq_cause = readl(hc_mmio + HC_IRQ_CAUSE_OFS);
+ if (hc_irq_cause) {
+ writelfl(0, hc_mmio + HC_IRQ_CAUSE_OFS);
+ }
+
+ VPRINTK("ENTER, hc%u relevant=0x%08x HC IRQ cause=0x%08x\n",
+ hc, relevant, hc_irq_cause);
+
+ for (port = port0; port < port0 + MV_PORTS_PER_HC; port++) {
+ ap = host_set->ports[port];
+ hard_port = port & MV_PORT_MASK; /* range 0-3 */
+ ata_status = 0xffU;
+
+ if (((CRBP_DMA_DONE | DEV_IRQ) << hard_port) & hc_irq_cause) {
+ BUG_ON(NULL == ap);
+ /* rcv'd new resp, basic DMA complete, or ATA IRQ */
+ /* This is needed to clear the ATA INTRQ.
+ * FIXME: don't read the status reg in EDMA mode!
+ */
+ ata_status = readb((void __iomem *)
+ ap->ioaddr.status_addr);
+ }
+
+ shift = port * 2;
+ if (port >= MV_PORTS_PER_HC) {
+ shift++; /* skip bit 8 in the HC Main IRQ reg */
+ }
+ if ((PORT0_ERR << shift) & relevant) {
+ mv_err_intr(ap);
+ /* FIXME: smart to OR in ATA_ERR? */
+ ata_status = readb((void __iomem *)
+ ap->ioaddr.status_addr) | ATA_ERR;
+ }

```

```

+
+ if (ap &&
+ (NULL != (qc = ata_qc_from_tag(ap, ap->active_tag)))) {
+ VPRINTK("port %u IRQ found for qc, ata_status 0x%x\n",
+ port,ata_status);
+ BUG_ON(0xffU == ata_status);
+ /* mark qc status appropriately */
+ ata_qc_complete(qc, ata_status);
+ }
+ }
+ VPRINTK("EXIT\n");
+ }
+
+static irqreturn_t mv_interrupt(int irq, void *dev_instance,
+ struct pt_regs *regs)
+{
+ struct ata_host_set *host_set = dev_instance;
+ unsigned int hc, handled = 0, n_hcs;
+ void __iomem *mmio;
+ u32 irq_stat;
+
+ mmio = host_set->mmio_base;
+ irq_stat = readl(mmio + HC_MAIN_IRQ_CAUSE);
+ n_hcs = mv_get_hc_count(host_set->ports[0]->flags);
+
+ /* check the cases where we either have nothing pending or have read
+ * a bogus register value which can indicate HW removal or PCI fault
+ */
+ if (!irq_stat || (0xffffffffU == irq_stat)) {
+ return IRQ_NONE;
+ }
+
+ spin_lock(&host_set->lock);
+
+ for (hc = 0; hc < n_hcs; hc++) {
+ u32 relevant = irq_stat & (HC0_IRQ_PEND << (hc * HC_SHIFT));
+ if (relevant) {
+ mv_host_intr(host_set, relevant, hc);
+ handled = 1;
+ }
+ }
+
+ if (PCI_ERR & irq_stat) {
+ /* FIXME: these are all masked by default, but still need
+ * to recover from them properly.
+ */
+ }
+
+ spin_unlock(&host_set->lock);
+
+ return (IRQ_RETVAL(handled));
+ }

```

```

+
+static void mv_phy_reset(struct ata_port *ap)
+{
+ void __iomem *port_mmio = mv_ap_base(ap);
+ struct ata_taskfile tf;
+ struct ata_device *dev = &ap->device[0];
+ u32 edma = 0, bdma;
+
+ VPRINTK("ENTER, port %u, mmio 0x%p\n", ap->port_no, port_mmio);
+
+ edma = readl(port_mmio + EDMA_CMD_OFS);
+ if (EDMA_EN & edma) {
+ /* disable EDMA if active */
+ edma &= ~EDMA_EN;
+ writelfl(edma | EDMA_DS, port_mmio + EDMA_CMD_OFS);
+ udelay(1);
+ } else if (mv_port_bdma_capable(ap) &&
+ (bdma = readl(port_mmio + BDMA_CMD_OFS)) & BDMA_START) {
+ /* disable BDMA if active */
+ writelfl(bdma & ~BDMA_START, port_mmio + BDMA_CMD_OFS);
+ }
+
+ writelfl(edma | ATA_RST, port_mmio + EDMA_CMD_OFS);
+ udelay(25); /* allow reset propagation */
+
+ /* Spec never mentions clearing the bit. Marvell's driver does
+ * clear the bit, however.
+ */
+ writelfl(edma & ~ATA_RST, port_mmio + EDMA_CMD_OFS);
+
+ VPRINTK("Done. Now calling __sata_phy_reset()\n");
+
+ /* proceed to init communications via the scr_control reg */
+ __sata_phy_reset(ap);
+
+ if (ap->flags & ATA_FLAG_PORT_DISABLED) {
+ VPRINTK("Port disabled pre-sig. Exiting.\n");
+ return;
+ }
+
+ tf.lbah = readb((void __iomem *) ap->ioaddr.lbah_addr);
+ tf.lbam = readb((void __iomem *) ap->ioaddr.lbam_addr);
+ tf.lbal = readb((void __iomem *) ap->ioaddr.lbal_addr);
+ tf.nsect = readb((void __iomem *) ap->ioaddr.nsect_addr);
+
+ dev->class = ata_dev_classify(&tf);
+ if (!ata_dev_present(dev)) {
+ VPRINTK("Port disabled post-sig: No device present.\n");
+ ata_port_disable(ap);
+ }
+ VPRINTK("EXIT\n");

```

```

+}
+
+static void mv_port_init(struct ata_ioports *port, unsigned long base)
+{
+ /* PIO related setup */
+ port->data_addr = base + SHD_PIO_DATA_OFS;
+ port->error_addr = port->feature_addr = base + SHD_FEA_ERR_OFS;
+ port->nsect_addr = base + SHD_SECT_CNT_OFS;
+ port->lbal_addr = base + SHD_LBA_L_OFS;
+ port->lbam_addr = base + SHD_LBA_M_OFS;
+ port->lbah_addr = base + SHD_LBA_H_OFS;
+ port->device_addr = base + SHD_DEV_HD_OFS;
+ port->status_addr = port->command_addr = base + SHD_CMD_STA_OFS;
+ port->altstatus_addr = port->ctl_addr = base + SHD_CTL_AST_OFS;
+ /* unused */
+ port->cmd_addr = port->bmdma_addr = port->scr_addr = 0;
+
+ /* unmask all EDMA error interrupts */
+ writel(~0, (void __iomem *)base + EDMA_ERR_IRQ_MASK_OFS);
+
+ VPRINTK("EDMA cfg=0x%08x EDMA IRQ err cause/mask=0x%08x/0x%08x\n",
+ readl((void __iomem *)base+EDMA_CFG_OFS),
+ readl((void __iomem *)base+EDMA_ERR_IRQ_CAUSE_OFS),
+ readl((void __iomem *)base+EDMA_ERR_IRQ_MASK_OFS));
+}
+
+static int mv_host_init(struct ata_probe_ent *probe_ent)
+{
+ int rc = 0, n_hc, port, hc;
+ void __iomem *mmio = probe_ent->mmio_base;
+ void __iomem *port_mmio;
+
+ if (mv_master_reset(probe_ent->mmio_base)) {
+ rc = 1;
+ goto done;
+ }
+
+ n_hc = mv_get_hc_count(probe_ent->host_flags);
+ probe_ent->n_ports = MV_PORTS_PER_HC * n_hc;
+
+ for (port = 0; port < probe_ent->n_ports; port++) {
+ port_mmio = mv_port_base(mmio, port);
+ mv_port_init(&probe_ent->port[port], (unsigned long)port_mmio);
+ }
+
+ for (hc = 0; hc < n_hc; hc++) {
+ VPRINTK("HC%i: HC config=0x%08x HC IRQ cause=0x%08x\n", hc,
+ readl(mv_hc_base(mmio, hc) + HC_CFG_OFS),
+ readl(mv_hc_base(mmio, hc) + HC_IRQ_CAUSE_OFS));
+ }
+}

```

Linux-Kernel: [RFC][PATCH 2.6.13] Marvell SATA support (PIO mode)

```

+ writel(~HC_MAIN_MASKED_IRQS, mmio + HC_MAIN_IRQ_MASK);
+ writel(PCI_UNMASK_ALL_IRQS, mmio + PCI_IRQ_MASK);
+
+ VPRINTK("HC MAIN IRQ cause/mask=0x%08x/0x%08x "
+ "PCI int cause/mask=0x%08x/0x%08x\n",
+ readl(mmio+HC_MAIN_IRQ_CAUSE),
+ readl(mmio+HC_MAIN_IRQ_MASK),
+ readl(mmio+PCI_IRQ_CAUSE),
+ readl(mmio+PCI_IRQ_MASK));
+
+ done:
+ return (rc);
+ }
+
+static int mv_init_one(struct pci_dev *pdev, const struct pci_device_id *ent)
+{
+ static int printed_version = 0;
+ struct ata_probe_ent *probe_ent = NULL;
+ struct mv_host_priv *hpriv;
+ unsigned int board_idx = (unsigned int)ent->driver_data;
+ void __iomem *mmio_base;
+ int pci_dev_busy = 0;
+ int rc;
+
+ if (!printed_version++) {
+ printk(KERN_DEBUG DRV_NAME " version " DRV_VERSION "\n");
+ }
+
+ VPRINTK("ENTER for PCI Bus:Slot.Func=%u:%u.%u\n", pdev->bus->number,
+ PCI_SLOT(pdev->devfn), PCI_FUNC(pdev->devfn));
+
+ rc = pci_enable_device(pdev);
+ if (rc) {
+ return (rc);
+ }
+
+ rc = pci_request_regions(pdev, DRV_NAME);
+ if (rc) {
+ pci_dev_busy = 1;
+ goto err_out;
+ }
+
+ pci_intx(pdev, 1);
+
+ probe_ent = kmalloc(sizeof(*probe_ent), GFP_KERNEL);
+ if (probe_ent == NULL) {
+ rc = -ENOMEM;
+ goto err_out_regions;
+ }
+
+ memset(probe_ent, 0, sizeof(*probe_ent));

```

```

+ probe_ent->dev = pci_dev_to_dev(pdev);
+ INIT_LIST_HEAD(&probe_ent->node);
+
+ mmio_base = ioremap_nocache(pci_resource_start(pdev, MV_PRIMARY_BAR),
+ pci_resource_len(pdev, MV_PRIMARY_BAR));
+ if (mmio_base == NULL) {
+ rc = -ENOMEM;
+ goto err_out_free_ent;
+ }
+
+ hpriv = kmalloc(sizeof(*hpriv), GFP_KERNEL);
+ if (!hpriv) {
+ rc = -ENOMEM;
+ goto err_out_iounmap;
+ }
+ memset(hpriv, 0, sizeof(*hpriv));
+
+ probe_ent->sht = mv_port_info[board_idx].sht;
+ probe_ent->host_flags = mv_port_info[board_idx].host_flags;
+ probe_ent->pio_mask = mv_port_info[board_idx].pio_mask;
+ probe_ent->udma_mask = mv_port_info[board_idx].udma_mask;
+ probe_ent->port_ops = mv_port_info[board_idx].port_ops;
+
+ probe_ent->irq = pdev->irq;
+ probe_ent->irq_flags = SA_SHIRQ;
+ probe_ent->mmio_base = mmio_base;
+ probe_ent->private_data = hpriv;
+
+ /* initialize adapter */
+ rc = mv_host_init(probe_ent);
+ if (rc) {
+ goto err_out_hpriv;
+ }
+ /* mv_print_info(probe_ent); */
+
+ {
+ int b, w;
+ u32 dw[4]; /* hold a line of 16b */
+ VPRINTK("PCI config space:\n");
+ for (b = 0; b < 0x40; ) {
+ for (w = 0; w < 4; w++) {
+ (void) pci_read_config_dword(pdev, b, &dw[w]);
+ b += sizeof(*dw);
+ }
+ VPRINTK("%08x %08x %08x %08x\n",
+ dw[0], dw[1], dw[2], dw[3]);
+ }
+ }
+
+ /* FIXME: check ata_device_add return value */
+ ata_device_add(probe_ent);

```

```
+ kfree(probe_ent);
+
+ return (0);
+
+ err_out_hpriv:
+ kfree(hpriv);
+ err_out_iounmap:
+ iounmap(mmio_base);
+ err_out_free_ent:
+ kfree(probe_ent);
+ err_out_regions:
+ pci_release_regions(pdev);
+ err_out:
+ if (!pci_dev_busy) {
+ pci_disable_device(pdev);
+ }
+
+ return (rc);
+}
+
+static int __init mv_init(void)
+{
+ return (pci_module_init(&mv_pci_driver));
+}
+
+static void __exit mv_exit(void)
+{
+ pci_unregister_driver(&mv_pci_driver);
+}
+
+MODULE_AUTHOR("Brett Russ");
+MODULE_DESCRIPTION("SCSI low-level driver for Marvell SATA controllers");
+MODULE_LICENSE("GPL");
+MODULE_DEVICE_TABLE(pci, mv_pci_tbl);
+MODULE_VERSION(DRV_VERSION);
+
+module_init(mv_init);
+module_exit(mv_exit);
```

Index: linux-2.6.13/drivers/scsi/Kconfig

```
=====
--- linux-2.6.13.orig/drivers/scsi/Kconfig
+++ linux-2.6.13/drivers/scsi/Kconfig
@@ -459,6 +459,15 @@ config SCSI_ATA_PIIX
```

If unsure, say N.

```
+config SCSI_SATA_MV
+ tristate "Marvell SATA support"
+ depends on SCSI_SATA && PCI && EXPERIMENTAL
+ help
+ This option enables support for the Marvell Serial ATA family.
```

Linux-Kernel: [RFC][PATCH 2.6.13] Marvell SATA support (PIO mode)

+ Currently supports 88SX[56]0[48][01] chips.

+

+ If unsure, say N.

+

```
config SCSI_SATA_NV
```

```
    tristate "NVIDIA SATA support"
```

```
    depends on SCSI_SATA && PCI && EXPERIMENTAL
```

Index: linux-2.6.13/drivers/scsi/Makefile

=====

```
--- linux-2.6.13.orig/drivers/scsi/Makefile
```

```
+++ linux-2.6.13/drivers/scsi/Makefile
```

```
@@ -132,6 +132,7 @@ obj-$(CONFIG_SCSI_SATA_SIS) += libata.o
```

```
obj-$(CONFIG_SCSI_SATA_SX4) += libata.o sata_sx4.o
```

```
obj-$(CONFIG_SCSI_SATA_NV) += libata.o sata_nv.o
```

```
obj-$(CONFIG_SCSI_SATA_ULI) += libata.o sata_uli.o
```

```
+obj-$(CONFIG_SCSI_SATA_MV) += libata.o sata_mv.o
```

```
obj-$(CONFIG_ARM) += arm/
```

-

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/>