

Fwd: Re: [RFC] pata_icside driver

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-04/msg11338.html>

- *From:* Russell King <rmk+lkml@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Sun, 29 Apr 2007 08:27:38 +0100
-

Resend. Copying linux-ide as requested appears to result in being ignored. :(

----- Forwarded message from Russell King <rmk+lkml@xxxxxxxxxxxxxxxxxxx> -----

Date: Sat, 21 Apr 2007 16:09:03 +0100
From: Russell King <rmk+lkml@xxxxxxxxxxxxxxxxxxx>
To: linux-kernel@xxxxxxxxxxxxxxxxxxx,
Andrew Morton <akpm@xxxxxxxxxxxxxxxxxxx>,
Jeff Garzik <jgarzik@xxxxxxxxxxx>, linux-ide@xxxxxxxxxxxxxxxxxxx
Subject: Re: [RFC] pata_icside driver

On Sun, Apr 08, 2007 at 11:18:26AM +0100, Russell King wrote:

Below is an initial attempt at converting the ICS IDE driver to fit into the PATA infrastructure.

There's a number of FIXMEs in there: due to the hardware missing resistors on the interrupt signals from the drives, a port without any drives attached results in spurious interrupts being generated.

To prevent this, we need to disable the interrupts from the port on the card if no drives are found, but unfortunately ATA doesn't call the "port_disable" method in this circumstance.

Here's an updated version. I've removed the correction of the cycle time – since we're checking whether all of active, recovery and cycle periods fit the hardware, the correction becomes unnecessary.

I still suggest that the PATA core folk consider fixing their timing calculation function in that respect though.

This driver continues to have the so far ignored issue concerning port_disable. It would be good to have some feedback on this instead of this driver continuing to be crippled by the libata core code. This really needs resolving before this driver can be merged, though I'm not sure how.

Fwd: Re: [RFC] pata_icside driver

```
diff --git a/drivers/ata/Kconfig b/drivers/ata/Kconfig
index 7bdb5a..9cd8a61 100644
--- a/drivers/ata/Kconfig
+++ b/drivers/ata/Kconfig
@@ -552,6 +552,14 @@ config PATA_PLATFORM
```

If unsure, say N.

```
+config PATA_ICSIDE
+ tristate "Acorn ICS PATA support"
+ depends on ARM && ARCH_ACORN
+ help
+ On Acorn systems, say Y here if you wish to use the ICS PATA
+ interface card. This is not required for ICS partition support.
+ If you are unsure, say N to this.
+
+config PATA_IXP4XX_CF
+ tristate "IXP4XX Compact Flash support"
+ depends on ARCH_IXP4XX
diff --git a/drivers/ata/Makefile b/drivers/ata/Makefile
index 13d7397..cc8798b 100644
--- a/drivers/ata/Makefile
+++ b/drivers/ata/Makefile
@@ -61,6 +61,7 @@ obj-$(CONFIG_PATA_TRIFLEX) += pata_triflex.o
obj-$(CONFIG_PATA_IXP4XX_CF) += pata_ixp4xx_cf.o
obj-$(CONFIG_PATA_SCC) += pata_scc.o
obj-$(CONFIG_PATA_PLATFORM) += pata_platform.o
+obj-$(CONFIG_PATA_ICSIDE) += pata_icside.o
# Should be last but one libata driver
obj-$(CONFIG_ATA_GENERIC) += ata_generic.o
# Should be last libata driver
diff --git a/drivers/ata/pata_icside.c b/drivers/ata/pata_icside.c
new file mode 100644
index 0000000..75b22da
--- /dev/null
+++ b/drivers/ata/pata_icside.c
@@ -0,0 +1,686 @@
+#include <linux/kernel.h>
+#include <linux/module.h>
+#include <linux/init.h>
+#include <linux/blkdev.h>
+#include <scsi/scsi_host.h>
+#include <linux/ata.h>
+#include <linux/libata.h>
+
+#include <asm/dma.h>
+#include <asm/ecard.h>
+
+#define DRV_NAME "pata_icside"
+
```

Fwd: Re: [RFC] pata_icside driver

```
+#define ICS_IDENT_OFFSET 0x2280
+
+#define ICS_ARCIN_V5_INTRSTAT 0x0000
+#define ICS_ARCIN_V5_INTROFFSET 0x0004
+
+#define ICS_ARCIN_V6_INTROFFSET_1 0x2200
+#define ICS_ARCIN_V6_INTRSTAT_1 0x2290
+#define ICS_ARCIN_V6_INTROFFSET_2 0x3200
+#define ICS_ARCIN_V6_INTRSTAT_2 0x3290
+
+struct portinfo {
+ unsigned int dataoffset;
+ unsigned int ctrloffset;
+ unsigned int stepping;
+};
+
+static const struct portinfo pata_icside_portinfo_v5 = {
+ .dataoffset = 0x2800,
+ .ctrloffset = 0x2b80,
+ .stepping = 6,
+};
+
+static const struct portinfo pata_icside_portinfo_v6_1 = {
+ .dataoffset = 0x2000,
+ .ctrloffset = 0x2380,
+ .stepping = 6,
+};
+
+static const struct portinfo pata_icside_portinfo_v6_2 = {
+ .dataoffset = 0x3000,
+ .ctrloffset = 0x3380,
+ .stepping = 6,
+};
+
+#define PATA_ICSIDE_MAX_SG 128
+
+struct pata_icside_state {
+ void __iomem *irq_port;
+ void __iomem *ioc_base;
+ unsigned int type;
+ unsigned int dma;
+ struct {
+ u8 port_sel;
+ u8 disabled;
+ unsigned int speed[ATA_MAX_DEVICES];
+ } port[2];
+ struct scatterlist sg[PATA_ICSIDE_MAX_SG];
+};
+
+#define ICS_TYPE_A3IN 0
+#define ICS_TYPE_A3USER 1
```

```

+#define ICS_TYPE_V6 3
+#define ICS_TYPE_V5 15
+#define ICS_TYPE_NOTYPE ((unsigned int)-1)
+
+/* ----- Version 5 PCB Support Functions ----- */
+/* Prototype: pata_icside_irqenable_arcin_v5 (struct expansion_card *ec, int irqnr)
+ * Purpose : enable interrupts from card
+ */
+static void pata_icside_irqenable_arcin_v5 (struct expansion_card *ec, int irqnr)
+{
+ struct pata_icside_state *state = ec->irq_data;
+
+
+ writeb(0, state->irq_port + ICS_ARCIN_V5_INTROFFSET);
+}
+
+/* Prototype: pata_icside_irqdisable_arcin_v5 (struct expansion_card *ec, int irqnr)
+ * Purpose : disable interrupts from card
+ */
+static void pata_icside_irqdisable_arcin_v5 (struct expansion_card *ec, int irqnr)
+{
+ struct pata_icside_state *state = ec->irq_data;
+
+
+ readb(state->irq_port + ICS_ARCIN_V5_INTROFFSET);
+}
+
+static const expansioncard_ops_t pata_icside_ops_arcin_v5 = {
+ .irqenable = pata_icside_irqenable_arcin_v5,
+ .irqdisable = pata_icside_irqdisable_arcin_v5,
+};
+
+
+/* ----- Version 6 PCB Support Functions ----- */
+/* Prototype: pata_icside_irqenable_arcin_v6 (struct expansion_card *ec, int irqnr)
+ * Purpose : enable interrupts from card
+ */
+static void pata_icside_irqenable_arcin_v6 (struct expansion_card *ec, int irqnr)
+{
+ struct pata_icside_state *state = ec->irq_data;
+ void __iomem *base = state->irq_port;
+
+
+ if (!state->port[0].disabled)
+ writeb(0, base + ICS_ARCIN_V6_INTROFFSET_1);
+ if (!state->port[1].disabled)
+ writeb(0, base + ICS_ARCIN_V6_INTROFFSET_2);
+}
+
+/* Prototype: pata_icside_irqdisable_arcin_v6 (struct expansion_card *ec, int irqnr)
+ * Purpose : disable interrupts from card
+ */
+static void pata_icside_irqdisable_arcin_v6 (struct expansion_card *ec, int irqnr)
+{

```

Fwd: Re: [RFC] pata_icside driver

```
+ struct pata_icside_state *state = ec->irq_data;
+
+ readb(state->irq_port + ICS_ARCIN_V6_INTROFFSET_1);
+ readb(state->irq_port + ICS_ARCIN_V6_INTROFFSET_2);
+}
+
+/* Prototype: pata_icside_irqprobe(struct expansion_card *ec)
+ * Purpose : detect an active interrupt from card
+ */
+static int pata_icside_irqpending_arcin_v6(struct expansion_card *ec)
+{
+ struct pata_icside_state *state = ec->irq_data;
+
+ return readb(state->irq_port + ICS_ARCIN_V6_INTRSTAT_1) & 1 ||
+ readb(state->irq_port + ICS_ARCIN_V6_INTRSTAT_2) & 1;
+}
+
+static const expansioncard_ops_t pata_icside_ops_arcin_v6 = {
+ .irqenable = pata_icside_irqenable_arcin_v6,
+ .irqdisable = pata_icside_irqdisable_arcin_v6,
+ .irqpending = pata_icside_irqpending_arcin_v6,
+};
+
+
+
+/*
+ * SG-DMA support.
+ *
+ * Similar to the BM-DMA, but we use the RiscPCs IOMD DMA controllers.
+ * There is only one DMA controller per card, which means that only
+ * one drive can be accessed at one time. NOTE! We do not enforce that
+ * here, but we rely on the main IDE driver spotting that both
+ * interfaces use the same IRQ, which should guarantee this.
+ */
+
+/*
+ * Configure the IOMD to give the appropriate timings for the transfer
+ * mode being requested. We take the advice of the ATA standards, and
+ * calculate the cycle time based on the transfer mode, and the EIDE
+ * MW DMA specs that the drive provides in the IDENTIFY command.
+ *
+ * We have the following IOMD DMA modes to choose from:
+ *
+ * Type Active Recovery Cycle
+ * A 250 (250) 312 (550) 562 (800)
+ * B 187 (200) 250 (550) 437 (750)
+ * C 125 (125) 125 (375) 250 (500)
+ * D 62 (50) 125 (375) 187 (425)
+ *
+ * (figures in brackets are actual measured timings on DIOR/DIOW)
+ *
+ * However, we also need to take care of the read/write active and
```

```

+ * recovery timings:
+ *
+ * Read Write
+ * Mode Active --- Recovery --- Cycle IOMD type
+ * MW0 215 50 215 480 A
+ * MW1 80 50 50 150 C
+ * MW2 70 25 25 120 C
+ */
+static void pata_icside_set_dmamode(struct ata_port *ap, struct ata_device *adev)
+{
+ struct pata_icside_state *state = ap->host->private_data;
+ struct ata_timing t;
+ unsigned int cycle;
+ char iomd_type;
+
+ /*
+ * DMA is based on a 16MHz clock
+ */
+ if (ata_timing_compute(adev, adev->dma_mode, &t, 1000, 1))
+ return;
+
+ /*
+ * Choose the IOMD cycle timing which ensure that the interface
+ * satisfies the measured active, recovery and cycle times.
+ */
+ if (t.active <= 50 && t.recover <= 375 && t.cycle <= 425)
+ iomd_type = 'D', cycle = 187;
+ else if (t.active <= 125 && t.recover <= 375 && t.cycle <= 500)
+ iomd_type = 'C', cycle = 250;
+ else if (t.active <= 200 && t.recover <= 550 && t.cycle <= 750)
+ iomd_type = 'B', cycle = 437;
+ else
+ iomd_type = 'A', cycle = 562;
+
+ ata_dev_printk(adev, KERN_INFO, "timings: act %dns rec %dns cyc %dns (%c)\n",
+ t.active, t.recover, t.cycle, iomd_type);
+
+ state->port[ap->port_no].speed[adev->devno] = cycle;
+}
+
+static void pata_icside_bmdma_setup(struct ata_queued_cmd *qc)
+{
+ struct ata_port *ap = qc->ap;
+ struct pata_icside_state *state = ap->host->private_data;
+ struct scatterlist *sg, *rsg = state->sg;
+ unsigned int write = qc->tf.flags & ATA_TFLAG_WRITE;
+
+ /*
+ * We are simplex; BUG if we try to fiddle with DMA
+ * while it's active.
+ */

```

Fwd: Re: [RFC] pata_icside driver

```
+ BUG_ON(dma_channel_active(state->dma));
+
+ /*
+ * Copy ATAs scattered sg list into a contiguous array of sg
+ */
+ ata_for_each_sg(sg, qc) {
+ memcpy(rsg, sg, sizeof(*sg));
+ rsg++;
+ }
+
+ /*
+ * Route the DMA signals to the correct interface
+ */
+ writeb(state->port[ap->port_no].port_sel, state->ioc_base);
+
+ set_dma_speed(state->dma, state->port[ap->port_no].speed[qc->dev->devno]);
+ set_dma_sg(state->dma, state->sg, rsg - state->sg);
+ set_dma_mode(state->dma, write ? DMA_MODE_WRITE : DMA_MODE_READ);
+
+ /* issue r/w command */
+ ap->ops->exec_command(ap, &qc->tf);
+ }
+
+static void pata_icside_bmdma_start(struct ata_queued_cmd *qc)
+{
+ struct ata_port *ap = qc->ap;
+ struct pata_icside_state *state = ap->host->private_data;
+
+ BUG_ON(dma_channel_active(state->dma));
+ enable_dma(state->dma);
+ }
+
+static void pata_icside_bmdma_stop(struct ata_queued_cmd *qc)
+{
+ struct ata_port *ap = qc->ap;
+ struct pata_icside_state *state = ap->host->private_data;
+
+ disable_dma(state->dma);
+
+ /* see ata_bmdma_stop */
+ ata_altstatus(ap);
+ }
+
+static u8 pata_icside_bmdma_status(struct ata_port *ap)
+{
+ struct pata_icside_state *state = ap->host->private_data;
+ void __iomem *irq_port;
+
+ irq_port = state->irq_port + (ap->port_no ? ICS_ARCIN_V6_INTRSTAT_2 :
+ ICS_ARCIN_V6_INTRSTAT_1);
+ }
```

Fwd: Re: [RFC] pata_icside driver

```
+ return readb(irq_port) & 1 ? ATA_DMA_INTR : 0;
+}
+
+static int icside_dma_init(struct ata_probe_ent *ae, struct expansion_card *ec)
+{
+ struct pata_icside_state *state = ae->private_data;
+ int i;
+
+ for (i = 0; i < ATA_MAX_DEVICES; i++) {
+ state->port[0].speed[i] = 480;
+ state->port[1].speed[i] = 480;
+ }
+
+ if (ec->dma != NO_DMA && !request_dma(ec->dma, DRV_NAME)) {
+ state->dma = ec->dma;
+ ae->mwdma_mask = 0x07; /* MW0..2 */
+ }
+
+ return 0;
+}
+
+
+static int pata_icside_port_start(struct ata_port *ap)
+{
+ /* No PRD to alloc */
+ return ata_pad_alloc(ap, ap->dev);
+}
+
+static struct scsi_host_template pata_icside_sht = {
+ .module = THIS_MODULE,
+ .name = DRV_NAME,
+ .ioctl = ata_scsi_ioctl,
+ .queuecommand = ata_scsi_queuecmd,
+ .can_queue = ATA_DEF_QUEUE,
+ .this_id = ATA_SHT_THIS_ID,
+ .sg_tablesize = PATA_ICSIDE_MAX_SG,
+ .cmd_per_lun = ATA_SHT_CMD_PER_LUN,
+ .emulated = ATA_SHT_EMULATED,
+ .use_clustering = ATA_SHT_USE_CLUSTERING,
+ .proc_name = DRV_NAME,
+ .dma_boundary = ~0, /* no dma boundaries */
+ .slave_configure = ata_scsi_slave_config,
+ .slave_destroy = ata_scsi_slave_destroy,
+ .bios_param = ata_std_bios_param,
+};
+
+ /* wish this was exported from libata-core */
+static void ata_dummy_noret(struct ata_port *port)
+{
+}
+
```

```

+/*
+ * We need to shut down unused ports to prevent spurious interrupts.
+ * FIXME: the libata core doesn't call this function for PATA interfaces.
+ */
+static void pata_icside_port_disable(struct ata_port *ap)
+{
+ struct pata_icside_state *state = ap->host->private_data;
+
+ ata_port_printk(ap, KERN_ERR, "disabling icside port\n");
+
+ ata_port_disable(ap);
+
+ state->port[ap->port_no].disabled = 1;
+
+ if (state->type == ICS_TYPE_V6) {
+ /*
+ * Disable interrupts from this port, otherwise we
+ * receive spurious interrupts from the floating
+ * interrupt line.
+ */
+ void __iomem *irq_port = state->irq_port +
+ (ap->port_no ? ICS_ARCIN_V6_INTROFFSET_2 : ICS_ARCIN_V6_INTROFFSET_1);
+ readb(irq_port);
+ }
+ }
+
+static u8 pata_icside_irq_ack(struct ata_port *ap, unsigned int chk_drq)
+{
+ unsigned int bits = chk_drq ? ATA_BUSY | ATA_DRQ : ATA_BUSY;
+ u8 status;
+
+ status = ata_busy_wait(ap, bits, 1000);
+ if (status & bits)
+ if (ata_msg_err(ap))
+ printk(KERN_ERR "abnormal status 0x%X\n", status);
+
+ if (ata_msg_intr(ap))
+ printk(KERN_INFO "%s: irq ack: drv_stat 0x%X\n",
+ __FUNCTION__, status);
+
+ return status;
+ }
+
+static struct ata_port_operations pata_icside_port_ops = {
+ .port_disable = pata_icside_port_disable,
+
+ .set_dmamode = pata_icside_set_dmamode,
+
+ .tf_load = ata_tf_load,
+ .tf_read = ata_tf_read,
+ .exec_command = ata_exec_command,

```

```

+ .check_status = ata_check_status,
+ .dev_select = ata_std_dev_select,
+
+ .bmdma_setup = pata_icside_bmdma_setup,
+ .bmdma_start = pata_icside_bmdma_start,
+
+ .data_xfer = ata_data_xfer_noirq,
+
+ /* no need to build any PRD tables for DMA */
+ .qc_prep = ata_noop_qc_prep,
+ .qc_issue = ata_qc_issue_prot,
+
+ .freeze = ata_bmdma_freeze,
+ .thaw = ata_bmdma_thaw,
+ .error_handler = ata_bmdma_error_handler,
+ .post_internal_cmd = pata_icside_bmdma_stop,
+
+ .irq_handler = ata_interrupt,
+ .irq_clear = ata_dummy_noret,
+ .irq_on = ata_irq_on,
+ .irq_ack = pata_icside_irq_ack,
+
+ .port_start = pata_icside_port_start,
+
+ .bmdma_stop = pata_icside_bmdma_stop,
+ .bmdma_status = pata_icside_bmdma_status,
+};
+
+static void
+pata_icside_add_port(struct ata_probe_ent *ae, void __iomem *base,
+ const struct portinfo *info)
+{
+ struct ata_ioports *ioaddr = &ae->port[ae->n_ports++];
+ void __iomem *cmd = base + info->dataoffset;
+
+ ioaddr->cmd_addr = cmd;
+ ioaddr->data_addr = cmd + (ATA_REG_DATA << info->stepping);
+ ioaddr->error_addr = cmd + (ATA_REG_ERR << info->stepping);
+ ioaddr->feature_addr = cmd + (ATA_REG_FEATURE << info->stepping);
+ ioaddr->nsect_addr = cmd + (ATA_REG_NSECT << info->stepping);
+ ioaddr->lbal_addr = cmd + (ATA_REG_LBAL << info->stepping);
+ ioaddr->lbam_addr = cmd + (ATA_REG_LBAM << info->stepping);
+ ioaddr->lbah_addr = cmd + (ATA_REG_LBAH << info->stepping);
+ ioaddr->device_addr = cmd + (ATA_REG_DEVICE << info->stepping);
+ ioaddr->status_addr = cmd + (ATA_REG_STATUS << info->stepping);
+ ioaddr->command_addr = cmd + (ATA_REG_CMD << info->stepping);
+
+ ioaddr->ctl_addr = base + info->ctrloffset;
+ ioaddr->altstatus_addr = ioaddr->ctl_addr;
+}
+

```

Fwd: Re: [RFC] pata_icside driver

```
+static int __init
+pata_icside_register_v5(struct ata_probe_ent *ae, struct expansion_card *ec)
+{
+ struct pata_icside_state *state = ae->private_data;
+ void __iomem *base;
+
+ base = ioremap(ecard_resource_start(ec, ECARD_RES_MEMC),
+ ecard_resource_len(ec, ECARD_RES_MEMC));
+ if (!base)
+ return -ENOMEM;
+
+ state->irq_port = base;
+
+ ec->irqaddr = base + ICS_ARCIN_V5_INTRSTAT;
+ ec->irqmask = 1;
+ ec->irq_data = state;
+ ec->ops = &pata_icside_ops_arcin_v5;
+
+ /*
+ * Be on the safe side - disable interrupts
+ */
+ ec->ops->irqdisable(ec, ec->irq);
+
+ pata_icside_add_port(ae, base, &pata_icside_portinfo_v5);
+
+ return 0;
+}
+
+static int __init
+pata_icside_register_v6(struct ata_probe_ent *ae, struct expansion_card *ec)
+{
+ struct pata_icside_state *state = ae->private_data;
+ void __iomem *ioc_base, *easi_base;
+ unsigned int sel = 0;
+ int ret;
+
+ ioc_base = ioremap(ecard_resource_start(ec, ECARD_RES_IOCFAST),
+ ecard_resource_len(ec, ECARD_RES_IOCFAST));
+ if (!ioc_base) {
+ ret = -ENOMEM;
+ goto out;
+ }
+
+ easi_base = ioc_base;
+
+ if (ecard_resource_flags(ec, ECARD_RES_EASI)) {
+ easi_base = ioremap(ecard_resource_start(ec, ECARD_RES_EASI),
+ ecard_resource_len(ec, ECARD_RES_EASI));
+ if (!easi_base) {
+ ret = -ENOMEM;
+ goto unmap_slot;
+ }
+ }
+}
```

```

+ }
+
+ /*
+ * Enable access to the EASI region.
+ */
+ sel = 1 << 5;
+ }
+
+ writeb(sel, ioc_base);
+
+ ec->irq_data = state;
+ ec->ops = &pata_icside_ops_arcin_v6;
+
+ state->irq_port = easi_base;
+ state->ioc_base = ioc_base;
+ state->port[0].port_sel = sel;
+ state->port[1].port_sel = sel | 1;
+
+ /*
+ * Be on the safe side – disable interrupts
+ */
+ ec->ops->irqdisable(ec, ec->irq);
+
+ /*
+ * Find and register the interfaces.
+ */
+ pata_icside_add_port(ae, easi_base, &pata_icside_portinfo_v6_1);
+ pata_icside_add_port(ae, easi_base, &pata_icside_portinfo_v6_2);
+
+ /*
+ * FIXME: work around libata's aversion to calling port_disable.
+ * This permanently disables interrupts on port 0 – bad luck if
+ * you have a drive on that port.
+ */
+ state->port[0].disabled = 1;
+
+ return icside_dma_init(ae, ec);
+
+ unmap_slot:
+ iounmap(ioc_base);
+ out:
+ return ret;
+ }
+
+static int __devinit
+pata_icside_probe(struct expansion_card *ec, const struct ecard_id *id)
+{
+ struct pata_icside_state *state;
+ struct ata_probe_ent ae;
+ void __iomem *idmem;
+ int ret;

```

```

+
+ ret = ecard_request_resources(ec);
+ if (ret)
+ goto out;
+
+ state = kzalloc(sizeof(struct pata_icside_state), GFP_KERNEL);
+ if (!state) {
+ ret = -ENOMEM;
+ goto release;
+ }
+
+ state->type = ICS_TYPE_NOTYPE;
+ state->dma = NO_DMA;
+
+ idmem = ioremap(ecard_resource_start(ec, ECARD_RES_IOCFAST),
+ ecard_resource_len(ec, ECARD_RES_IOCFAST));
+ if (idmem) {
+ unsigned int type;
+
+ type = readb(idmem + ICS_IDENT_OFFSET) & 1;
+ type |= (readb(idmem + ICS_IDENT_OFFSET + 4) & 1) << 1;
+ type |= (readb(idmem + ICS_IDENT_OFFSET + 8) & 1) << 2;
+ type |= (readb(idmem + ICS_IDENT_OFFSET + 12) & 1) << 3;
+ iounmap(idmem);
+
+ state->type = type;
+ }
+
+ memset(&ae, 0, sizeof(ae));
+ INIT_LIST_HEAD(&ae.node);
+ ae.dev = &ec->dev;
+ ae.port_ops = &pata_icside_port_ops;
+ ae.sht = &pata_icside_sht;
+ ae.pio_mask = 0x1f;
+ ae.irq = ec->irq;
+ ae.port_flags = ATA_FLAG_SLAVE_POSS | ATA_FLAG_SRST;
+ ae._host_flags = ATA_HOST_SIMPLEX;
+ ae.private_data = state;
+
+ switch (state->type) {
+ case ICS_TYPE_A3IN:
+ dev_warn(&ec->dev, "A3IN unsupported\n");
+ ret = -ENODEV;
+ break;
+
+ case ICS_TYPE_A3USER:
+ dev_warn(&ec->dev, "A3USER unsupported\n");
+ ret = -ENODEV;
+ break;
+
+ case ICS_TYPE_V5:

```

Fwd: Re: [RFC] pata_icside driver

```
+ ret = pata_icside_register_v5(&ae, ec);
+ break;
+
+ case ICS_TYPE_V6:
+ ret = pata_icside_register_v6(&ae, ec);
+ break;
+
+ default:
+ dev_warn(&ec->dev, "unknown interface type\n");
+ ret = -ENODEV;
+ break;
+ }
+
+ if (ret == 0)
+ ret = ata_device_add(&ae) == 0 ? -ENODEV : 0;
+
+ if (ret == 0)
+ goto out;
+
+ kfree(state);
+ release:
+ ecard_release_resources(ec);
+ out:
+ return ret;
+}
+
+static void pata_icside_shutdown(struct expansion_card *ec)
+{
+ struct ata_host *host = ecard_get_drvdata(ec);
+ unsigned long flags;
+
+ /*
+ * Disable interrupts from this card. We need to do
+ * this before disabling EASI since we may be accessing
+ * this register via that region.
+ */
+ local_irq_save(flags);
+ if (ec->ops)
+ ec->ops->irqdisable(ec, ec->irq);
+ local_irq_restore(flags);
+
+ /*
+ * Reset the ROM pointer so that we can read the ROM
+ * after a soft reboot. This also disables access to
+ * the IDE taskfile via the EASI region.
+ */
+ if (host) {
+ struct pata_icside_state *state = host->private_data;
+ if (state->ioc_base)
+ writeb(0, state->ioc_base);
+ }
+}
```

```

+}
+
+static void __devexit pata_icside_remove(struct expansion_card *ec)
+{
+ struct ata_host *host = ecard_get_drvdata(ec);
+ struct pata_icside_state *state = host->private_data;
+
+ ata_host_detach(host);
+
+ pata_icside_shutdown(ec);
+
+ /*
+ * don't NULL out the drvdata - devres/libata wants it
+ * to free the ata_host structure.
+ */
+ ec->ops = NULL;
+ ec->irq_data = NULL;
+
+ if (state->dma != NO_DMA)
+ free_dma(state->dma);
+ if (state->ioc_base)
+ iounmap(state->ioc_base);
+ if (state->ioc_base != state->irq_port)
+ iounmap(state->irq_port);
+
+ kfree(state);
+ ecard_release_resources(ec);
+}
+
+static const struct ecard_id pata_icside_ids[] = {
+ { MANU_ICS, PROD_ICS_IDE },
+ { MANU_ICS2, PROD_ICS2_IDE },
+ { 0xffff, 0xffff }
+};
+
+static struct ecard_driver pata_icside_driver = {
+ .probe = pata_icside_probe,
+ .remove = __devexit_p(pata_icside_remove),
+ .shutdown = pata_icside_shutdown,
+ .id_table = pata_icside_ids,
+ .drv = {
+ .name = DRV_NAME,
+ },
+};
+
+static int __init pata_icside_init(void)
+{
+ return ecard_register_driver(&pata_icside_driver);
+}
+
+static void __exit pata_icside_exit(void)

```

Fwd: Re: [RFC] pata_icside driver

```
+{  
+ ecard_remove_driver(&pata_icside_driver);  
+}  
+  
+MODULE_AUTHOR("Russell King <rmk@xxxxxxxxxxxxxxxxxxx>");  
+MODULE_LICENSE("GPL");  
+MODULE_DESCRIPTION("ICS PATA driver");  
+  
+module_init(pata_icside_init);  
+module_exit(pata_icside_exit);
```

--
Russell King
Linux kernel 2.6 ARM Linux – <http://www.arm.linux.org.uk/>
maintainer of:

–
To unsubscribe from this list: send the line "unsubscribe linux–kernel" in
the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo–info.html>
Please read the FAQ at <http://www.tux.org/lkml/>

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

--
Russell King
Linux kernel 2.6 ARM Linux – <http://www.arm.linux.org.uk/>
maintainer of:

–
To unsubscribe from this list: send the line "unsubscribe linux–kernel" in
the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo–info.html>
Please read the FAQ at <http://www.tux.org/lkml/>