

[PATCH] preliminary sata_promise ATAPI support

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-12/msg08014.html>

- *From:* Mikael Pettersson <mikpe@xxxxxxxx>
 - *Date:* Sat, 30 Dec 2006 18:37:18 +0100 (MET)
-

This patch against 2.6.20-rc2 adds partial ATAPI support to the sata_promise driver. This patch is preliminary and for review only.

Current Status:

- Tested with PATA optical devices on 20378 and 20575 chips.
- Mounting and reading CD/DVD discs in DMA mode works fine.
- Using cdrecord to write CD-R discs works if DMA mode is disabled (s/0/1/ in pdc_check_atapi_dma()). With DMA enabled, cdrecord first complains that capability checking commands fail, and then fails to write due to error interrupts.

The most delicate change is pdc_issue_atapi_pkt_cmd(), which is ported from Promise's pdc-ultra/pdc-ulsata2 drivers. There's no documentation to explain why this code is needed, but it looks to me as if these steps should have been done in a taskfile, but for some reason they need to do them manually.

As to why CD-writing fails with DMA enabled, I suspect that I either need to blacklist specific packet commands from using DMA, or I have some bug in pdc_qc_prep() or pdc_qc_issue_prot() for the ATA_PROT_ATAPI_DMA case.

To use this with PATA optical devices you also need the forever pending sata_promise PATA patches. I didn't want ATAPI support to be delayed by those, so I based ATAPI on the vanilla kernel and rebased PATA on top of ATAPI; you can get my sata_promise patch kit from <<http://www.csd.uu.se/~mikpe/linux/patches/2.6/>>.

I should be able to do some testing with a SATA ATAPI CD/DVD writer later next week.

/Mikael

```
diff -rupN linux-2.6.20-rc2/drivers/ata/sata_promise.c
linux-2.6.20-rc2.sata_promise-ataapi-wip-1/drivers/ata/sata_promise.c
--- linux-2.6.20-rc2/drivers/ata/sata_promise.c 2006-12-24 13:21:26.000000000 +0100
+++ linux-2.6.20-rc2.sata_promise-ataapi-wip-1/drivers/ata/sata_promise.c 2006-12-30
16:58:46.000000000 +0100
```

[PATCH] preliminary sata_promise ATAPI support

```
@@ -105,6 +105,7 @@ static void pdc_pata_phy_reset(struct ata
static void pdc_qc_prep(struct ata_queued_cmd *qc);
static void pdc_tf_load_mmio(struct ata_port *ap, const struct ata_taskfile *tf);
static void pdc_exec_command_mmio(struct ata_port *ap, const struct ata_taskfile *tf);
+static int pdc_check_atapi_dma(struct ata_queued_cmd *qc);
static void pdc_irq_clear(struct ata_port *ap);
static unsigned int pdc_qc_issue_prot(struct ata_queued_cmd *qc);
static void pdc_host_stop(struct ata_host *host);
@@ -139,6 +140,7 @@ static const struct ata_port_operations
.check_status = ata_check_status,
.exec_command = pdc_exec_command_mmio,
.dev_select = ata_std_dev_select,
+ .check_atapi_dma = pdc_check_atapi_dma,

.qc_prep = pdc_qc_prep,
.qc_issue = pdc_qc_issue_prot,
@@ -183,7 +185,7 @@ static const struct ata_port_info pdc_po
/* board_2037x */
{
.sht = &pdc_ata_sht,
- .flags = PDC_COMMON_FLAGS | ATA_FLAG_SATA,
+ .flags = (PDC_COMMON_FLAGS & ~ATA_FLAG_NO_ATAPI) | ATA_FLAG_SATA,
.pio_mask = 0x1f, /* pio0-4 */
.mwdma_mask = 0x07, /* mwdma0-2 */
.udma_mask = 0x7f, /* udma0-6 ; FIXME */
@@ -213,7 +215,7 @@ static const struct ata_port_info pdc_po
/* board_2057x */
{
.sht = &pdc_ata_sht,
- .flags = PDC_COMMON_FLAGS | ATA_FLAG_SATA,
+ .flags = (PDC_COMMON_FLAGS & ~ATA_FLAG_NO_ATAPI) | ATA_FLAG_SATA,
.pio_mask = 0x1f, /* pio0-4 */
.mwdma_mask = 0x07, /* mwdma0-2 */
.udma_mask = 0x7f, /* udma0-6 ; FIXME */
@@ -415,6 +417,30 @@ static void pdc_qc_prep(struct ata_queue
pdc_pkt_footer(&qc->tf, pp->pkt, i);
break;

+ case ATA_PROT_ATAPI:
+ case ATA_PROT_ATAPI_NODATA:
+ ata_qc_prep(qc);
+ break;
+
+ case ATA_PROT_ATAPI_DMA:
+ ata_qc_prep(qc);
+ /* let pdc_pkt_header() set up the header proper (12 bytes),
+ it will set up another 4 bytes too which we'll overwrite */
+ qc->tf.protocol = ATA_PROT_DMA;
+ pdc_pkt_header(&qc->tf, qc->ap->prd_dma, qc->dev->devno, pp->pkt);
+ qc->tf.protocol = ATA_PROT_ATAPI_DMA;
+ if (qc->dev->cdb_len & ~0x1E) { /* 2/4/6/8/10/12/14/16 are Ok */
```

[PATCH] preliminary sata_promise ATAPI support

```
+ printk(KERN_ERR "%s: bad cdb_len %u\n", __FUNCTION__, qc->dev->cdb_len);
+ BUG();
+ }
+ pp->pkt[12] = (((qc->dev->cdb_len >> 1) & 7) << 5) | 0x00 | 0x08;
+ memcpy(pp->pkt+13, qc->cdb, qc->dev->cdb_len);
+ #if 0
+ /* pdc-ultra/cam/cam_ata.c will pad SG length to a multiple
+ of 4 here, but libata has already done that for us */
+ #endif
+ break;
+
default:
break;
}
@@ -528,6 +554,7 @@ static inline unsigned int pdc_host_intr
switch (qc->tf.protocol) {
case ATA_PROT_DMA:
case ATA_PROT_NODATA:
+ case ATA_PROT_ATAPI_DMA:
qc->err_mask |= ac_err_mask(ata_wait_idle(ap));
ata_qc_complete(qc);
handled = 1;
@@ -615,6 +642,7 @@ static inline void pdc_packet_start(stru
VPRINTK("ENTER, ap %p\n", ap);

+ /* XXX: for SATAII ATAPI PIO write SEQ_FPDMA (1<<6) not PacketCount */
writel(0x00000001, ap->host->mmio_base + (seq * 4));
readl(ap->host->mmio_base + (seq * 4)); /* flush */

@@ -624,16 +652,115 @@ static inline void pdc_packet_start(stru
readl((void __iomem *) ap->iocmd_addr + PDC_PKT_SUBMIT); /* flush */
}

+static unsigned int pdc_wait_on_busy(struct ata_port *ap)
+{
+ /* ata_busy_wait() loops reading status while pdc-ultra's WaitOnBusy()
+ * loops reading altstatus. Both approaches seem to work here.
+ */
+ unsigned int status = ata_busy_wait(ap, ATA_BUSY, 1000);
+ if (status != 0xff && (status & ATA_BUSY))
+ printk(KERN_ERR "%s: port %u: timed out, status %#x\n", __FUNCTION__, ap->port_no, status);
+ return status;
+}
+
+static unsigned int pdc_wait_for_drq(struct ata_port *ap)
+{
+ void __iomem *port_mmio = (void __iomem *) ap->iocmd_addr;
+ unsigned int i;
+ unsigned int status;
+}
```

[PATCH] preliminary sata_promise ATAPI support

```
+ /* Following pdc-ultra's WaitForDrq() we loop here until BSY
+ * is clear and DRQ is set in altstatus. We could possibly call
+ * ata_busy_wait() and loop until DRQ is set, but since we don't
+ * know how much time a call to ata_busy_wait() took, we don't
+ * know when to time out the outer loop.
+ */
+ for(i = 0; i < 1000; ++i) {
+ status = readb(port_mmio + 0x38); /* altstatus */
+ if (status == 0xFF)
+ break;
+ if (status & ATA_BUSY)
+ ;
+ else if (status & (ATA_DRQ | ATA_ERR))
+ break;
+ mdelay(1);
+ }
+ if (i >= 1000)
+ printk("%s: port %u: timed out\n", __FUNCTION__, ap->port_no);
+ return status;
+ }
+
+ static void pdc_issue_atapi_pkt_cmd(struct ata_queued_cmd *qc)
+ {
+ struct ata_port *ap = qc->ap;
+ void __iomem *port_mmio = (void __iomem *) ap->ioaddr.cmd_addr;
+ void __iomem *host_mmio = ap->host->mmio_base;
+ unsigned int nbytes;
+ unsigned int tmp;
+
+ /* disable INTA here, it will be re-enable when CAM use SEQ 0 for packets */
+ writeb(0x00, port_mmio + PDC_CTLSTAT); /* that the drive INT pass to SEQ 0*/
+ writeb(0x20, host_mmio + 0); /* but mask SEQ 0 INT */
+
+ /* select drive */
+ if (sata_scr_valid(ap)) {
+ tmp = 0xE0;
+ } else {
+ tmp = ATA_DEVICE_OBS;
+ if (qc->dev->devno != 0)
+ tmp |= ATA_DEV1;
+ }
+ writeb(tmp, port_mmio + 0x18); /* device/head */
+ pdc_wait_on_busy(ap);
+
+ writeb(0x00, port_mmio + 0x08); /* sector count */
+ writeb(0x00, port_mmio + 0x0c); /* sector number */
+
+ /* set feature register */
+ if (qc->tf.protocol != ATA_PROT_ATAPI_DMA) {
+ tmp = 0x00; /* data xfer by PIO ! */
+ /* set real transfer byte count to byte counter register (cylinder low/high) */
```

[PATCH] preliminary sata_promise ATAPI support

```
+ nbytes = qc->nbytes;
+ if (!nbytes)
+ nbytes = qc->nsect << 9;
+ if (nbytes > 0xffff)
+ nbytes = 0xffff;
+ } else {
+ tmp = 0x01; /* set DMA bit */
+ /* byte counter register (cylinder low/high) , set to 0 */
+ nbytes = 0;
+ }
+ writeb(tmp, port_mmio + 0x04); /* feature */
+ writeb(nbytes & 0xFF, port_mmio + 0x10); /* cylinder low */
+ writeb((nbytes >> 8) & 0xFF, port_mmio + 0x14); /* cylinder high */
+
+ /* send ATAPI packet command 0xA0 */
+ writeb(ATA_CMD_PACKET, port_mmio + 0x1c); /* command/status */
+
+#if 0 /* XXX: don't know if this is relevant or how to realize the test */
+ /* Do we have to wait INT ? */
+ if (the device interrupts with assertion of DRQ after receiving ATAPI packet command)
+ WaitForINT(ap);
+#endif
+
+ pdc_wait_for_drq(ap);
+
+ /* now device wait CDB only */
+}
+
static unsigned int pdc_qc_issue_prot(struct ata_queued_cmd *qc)
{
switch (qc->tf.protocol) {
+ case ATA_PROT_ATAPI_DMA:
+ pdc_issue_atapi_pkt_cmd(qc);
+ /*FALLTHROUGH*/
case ATA_PROT_DMA:
case ATA_PROT_NODATA:
pdc_packet_start(qc);
return 0;

- case ATA_PROT_ATAPI_DMA:
- BUG();
+ case ATA_PROT_ATAPI:
+ case ATA_PROT_ATAPI_NODATA:
break;

default:
@@ -658,6 +785,10 @@ static void pdc_exec_command_mmio(struct
ata_exec_command(ap, tf);
}

+static int pdc_check_atapi_dma(struct ata_queued_cmd *qc)
```

[PATCH] preliminary sata_promise ATAPI support

```
+{  
+ return 0; /* ATAPI DMA is supported */  
+}
```

```
static void pdc_ata_setup_port(struct ata_ioports *port, unsigned long base)  
{
```

—

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

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

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