

[PATCH 2.6.20-rc3] sata_promise: ATAPI support

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-01/msg01648.html>

- *From:* Mikael Pettersson <mikpe@xxxxxxxx>
 - *Date:* Sat, 6 Jan 2007 21:29:50 +0100 (MET)
-

This patch adds ATAPI support to the sata_promise driver. This has been tested on both first- and second-generation chips (20378 and 20575), and with both SATAPI and PATAPI devices. CD-writing works, and bulk data transfers use DMA.

SATAPI works on second-generation chips, but not on first-generation chips due to HW limitations. PATAPI works on both first- and second-generation chips, but requires the PATA support patch as well (which needs an update due to a conflict in the board_2037x definition).

The functional changes to the driver are:

- remove ATA_FLAG_NO_ATAPI from PDC_COMMON_FLAGS, and add it back to the boards that cannot support SATAPI (20319, 2037x)
- add ->check_atapi() operation to enable DMA for bulk data transfers but force PIO for other ATAPI commands; this filter is taken from Promise's driver and pata_pdc207x.c
- add handling of ATAPI protocols to pdc_qc_prep(), pdc_host_intr(), and pdc_qc_issue_prot(): ATAPI_DMA is handled by the driver while non-DMA protocols are handed over to libata generic code
- add pdc_issue_atapi_pkt_cmd() to handle the initial steps in issuing ATAPI DMA commands before sending the actual CDB; this procedure was ported from Promise's driver

With this patch ATAPI will work on SATA ports on second-generation chips, and on the first-generation PATA-only 20619. ATAPI on the 2057x' PATA port will work automatically when #promise-sata-pata is updated. ATAPI on the 2037x' PATA port is enabled by a followup patch for the #promise-sata-pata branch.

Signed-off-by: Mikael Pettersson <mikpe@xxxxxxxx>

```
--- linux-2.6.20-rc3/drivers/ata/sata_promise.c~1~ 2007-01-01 13:32:35.000000000 +0100
+++ linux-2.6.20-rc3/drivers/ata/sata_promise.c 2007-01-06 15:36:43.000000000 +0100
@@ -39,6 +39,7 @@
#include <linux/interrupt.h>
#include <linux/sched.h>
#include <linux/device.h>
+#include <scsi/scsi.h>
```

[PATCH 2.6.20-rc3] sata_promise: ATAPI support

```
#include <scsi/scsi_host.h>
#include <scsi/scsi_cmnd.h>
#include <linux/libata.h>
@@ -77,7 +78,7 @@ enum {
PDC_RESET = (1 << 11), /* HDMA reset */

PDC_COMMON_FLAGS = ATA_FLAG_NO_LEGACY |
- ATA_FLAG_MMIO | ATA_FLAG_NO_ATAPI |
+ ATA_FLAG_MMIO |
ATA_FLAG_PIO_POLLING,

/* hp->flags bits */
@@ -105,6 +106,7 @@ static void pdc_pata_phy_reset(struct at
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 +141,7 @@ 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,
@@ -164,6 +167,7 @@ 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,

.phy_reset = pdc_pata_phy_reset,

@@ -183,7 +187,7 @@ 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 */
@@ -193,7 +197,7 @@ static const struct ata_port_info pdc_po
/* board_20319 */
{
.sht = &pdc_ata_sht,
- .flags = PDC_COMMON_FLAGS | ATA_FLAG_SATA,
+ .flags = PDC_COMMON_FLAGS | ATA_FLAG_NO_ATAPI | ATA_FLAG_SATA,
```

[PATCH 2.6.20-rc3] sata_promise: ATAPI support

```
.pio_mask = 0x1f, /* pio0-4 */
.mwdma_mask = 0x07, /* mwdma0-2 */
.udma_mask = 0x7f, /* udma0-6 ; FIXME */
@@ -391,6 +395,30 @@ static void pdc_sata_scr_write (struct ata_
writel(val, (void __iomem *) ap->ioaddr.scr_addr + (sc_reg * 4));
}

+static void pdc_atapi_dma_pkt(struct ata_taskfile *tf,
+ dma_addr_t sg_table,
+ unsigned int cdb_len, u8 *cdb,
+ u8 *buf)
+{
+ u32 *buf32 = (u32 *) buf;
+
+ /* set control bits (byte 0), zero delay seq id (byte 3),
+ * and seq id (byte 2)
+ */
+ if (!(tf->flags & ATA_TFLAG_WRITE))
+ buf32[0] = cpu_to_le32(PDC_PKT_READ);
+ else
+ buf32[0] = 0;
+ buf32[1] = cpu_to_le32(sg_table); /* S/G table addr */
+ buf32[2] = 0; /* no next-packet */
+
+ /* we can represent cdb lengths 2/4/6/8/10/12/14/16 */
+ BUG_ON(cdb_len & ~0x1E);
+
+ buf[12] = (((cdb_len >> 1) & 7) << 5) | ATA_REG_DATA | PDC_LAST_REG;
+ memcpy(buf+13, cdb, cdb_len);
+}
+
static void pdc_qc_prep(struct ata_queued_cmd *qc)
{
struct pdc_port_priv *pp = qc->ap->private_data;
@@ -415,6 +443,16 @@ 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);
+ pdc_atapi_dma_pkt(&qc->tf, qc->ap->prd_dma, qc->dev->cdb_len, qc->cdb, pp->pkt);
+ break;
+
default:
break;
}
}
```

[PATCH 2.6.20-rc3] sata_promise: ATAPI support

```
@@ -528,6 +566,7 @@ 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;
@@ -624,18 +663,106 @@ static inline void pdc_packet_start(stru
readl((void __iomem *) ap->ioaddr.cmd_addr + PDC_PKT_SUBMIT); /* flush */
}

+static unsigned int pdc_wait_for_drq(struct ata_port *ap)
+{
+ void __iomem *port_mmio = (void __iomem *) ap->ioaddr.cmd_addr;
+ unsigned int i;
+ unsigned int status;
+
+ /* 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)
+ ata_port_printk(ap, KERN_WARNING, "%s timed out", __FUNCTION__);
+ 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 */
+}
```

[PATCH 2.6.20-rc3] sata_promise: ATAPI support

```
+ /* 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 */
+ ata_busy_wait(ap, ATA_BUSY, 1000);
+
+ 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) */
+ 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 */
+
+ /*
+ * At this point in the issuing of a packet command, the Promise
+ * driver busy-waits for INT (CTLSTAT bit 27) if it detected
+ * (at port init time) that the device interrupts with assertion
+ * of DRQ after receiving a packet command.
+ *
+ * XXX: Do we need to handle this case as well? Does libata detect
+ * this case for us, or do we have to do our own per-port init?
+ */
+
+ 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();
- break;
-
default:
break;
}
@@ -658,6 +785,32 @@ static void pdc_exec_command_mmio(struct
ata_exec_command(ap, tf);
}

+static int pdc_check_atapi_dma(struct ata_queued_cmd *qc)
+{
+ u8 *scsicmd = qc->scsicmd->cmd;
+ int pio = 1; /* atapi dma off by default */
+
+ /* Whitelist commands that may use DMA. */
+ switch (scsicmd[0]) {
+ case WRITE_12:
+ case WRITE_10:
+ case WRITE_6:
+ case READ_12:
+ case READ_10:
+ case READ_6:
+ case 0xad: /* READ_DVD_STRUCTURE */
+ case 0xbe: /* READ_CD */
+ pio = 0;
+ }
+ /* -45150 (FFFF4FA2) to -1 (FFFFFFF) shall use PIO mode */
+ if (scsicmd[0] == WRITE_10) {
+ unsigned int lba;
+ lba = (scsicmd[2] << 24) | (scsicmd[3] << 16) | (scsicmd[4] << 8) | scsicmd[5];
+ if (lba >= 0xFFFF4FA2)
+ pio = 1;
+ }
+ return pio;
+}

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

```

[PATCH 2.6.20-rc3] sata_promise: ATAPI support

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/>