

## [Q] Bio traversal trouble?

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-06/msg01094.html>

---

- *From:* Rene Herman <[rene.herman@xxxxxxxxx](mailto:rene.herman@xxxxxxxxx)>
  - *Date:* Mon, 04 Jun 2007 11:42:00 +0200
- 

Hi Jens, list.

The new Mitsumi legacy CD-ROM driver seems largely done. Both audio and data work (and the driver's clean) but data does still have a problem that I think needs some input from a block person. I don't have any idea...

Audio is fine, and data basically as well. It's performing at maximum drive speed (and gets the correct data):

```
root@5bt5:~# dd if=/dev/mitsumi of=/dev/null bs=2k count=8k
8192+0 records in
8192+0 records out
16777216 bytes (17 MB) copied, 111.379 seconds, 151 kB/s
```

Unfortunately, I can make the box oops by just doing enough of those dd runs in a row (this one with bs=2k count=128, oopsed the seventh time or something):

===

BUG: unable to handle kernel paging request at virtual address 8c1d2071

printing eip:

c10a6d6f

\*pde = 00000000

Oops: 0002 [#1]

Modules linked in: mitsumi nfsd exportfs lockd sunrpc nls\_cp437 msdos fat nls\_base

CPU: 0

EIP: 0060:[<c10a6d6f>] Not tainted VLI

EFLAGS: 00010246 (2.6.21.3 #1)

EIP is at ioread8\_rep+0x20/0x31

eax: 00010000 ebx: 00010300 ecx: 00000800 edx: 00000300

esi: c3145b30 edi: 8c1d2071 ebp: 8c1d2071 esp: c3145aec

ds: 007b es: 007b fs: 00d8 gs: 0033 ss: 0068

Process dd (pid: 1770, ti=c3145000 task=c3114110 task.ti=c3145000)

Stack: 00000000 c3a7ad48 c486504d c4865ab7 00000006 50020034 c1015f7b 00000292

00000000 00520300 00000100 0007e000 00000000 c3340300 c10a0531 00000000

c12adf60 00000000 00000001 00000000 00000000 00000000 00000000 dead4ead ffffffff

Call Trace:

[<c486504d>] \_\_mitsumi\_get\_frame+0xc/0x16 [mitsumi]

[<c4865ab7>] mitsumi\_get\_frame+0x120/0x134 [mitsumi]

[<c1015f7b>] lock\_timer\_base+0x15/0x2f

[<c10a0531>] cfq\_set\_request+0x0/0x144

[Q] Bio traversal trouble?

```

[<c114fefd>] _spin_unlock_irq+0x20/0x23
[<c1033bda>] mempool_free+0x5f/0x64
[<c4865b2d>] mitsumi_read+0x62/0x94 [mitsumi]
[<c4865ba8>] mitsumi_request+0x49/0xb5 [mitsumi]
[<c1099aae>] blk_start_queueing+0x14/0x1c
[<c10971c2>] elv_insert+0xa3/0x13f
[<c114fbc5>] _spin_lock_irq+0x2f/0x3a
[<c109a577>] __make_request+0x29f/0x2d3
[<c109a75b>] generic_make_request+0x136/0x146
[<c1048433>] kmem_cache_alloc+0x93/0x9e
[<c1033ae0>] mempool_alloc+0x32/0xcd
[<c1033ae0>] mempool_alloc+0x32/0xcd
[<c109a80f>] submit_bio+0xa4/0xaa
[<c106736b>] bio_alloc_bioset+0xb2/0x112
[<c1066dd1>] submit_bh+0xc2/0xdb
[<c1065fc4>] block_read_full_page+0x245/0x252
[<c1068346>] blkdev_get_block+0x0/0x36
[<c114ffff>] _write_unlock_irq+0x20/0x23
[<c103134f>] add_to_page_cache+0x71/0x78
[<c10368f3>] read_pages+0x7c/0xc1
[<c114ff7e>] _read_unlock_irq+0x20/0x23
[<c114ff7e>] _read_unlock_irq+0x20/0x23
[<c1025de0>] trace_hardirqs_on+0x11b/0x13e
[<c1036a25>] __do_page_cache_readahead+0xed/0x107
[<c1036b43>] blockable_page_cache_readahead+0x4d/0x9d
[<c1036c17>] make_ahead_window+0x84/0xa5
[<c1036d74>] page_cache_readahead+0x13c/0x15b
[<c1031cbc>] file_read_actor+0x7f/0x102
[<c1031953>] do_generic_mapping_read+0x110/0x3fa
[<c1031ec6>] generic_file_aio_read+0x187/0x1b0
[<c1031c3d>] file_read_actor+0x0/0x102
[<c104abc3>] do_sync_read+0xbf/0xfc
[<c101f2b8>] autoremove_wake_function+0x0/0x33
[<c1070fe1>] dnotify_parent+0x1b/0x66
[<c104aec2>] vfs_write+0xc9/0xff
[<c1027210>] __lock_release+0x2d/0x3f
[<c104ac87>] vfs_read+0x87/0xfd
[<c104af39>] sys_read+0x41/0x67
[<c1002490>] syscall_call+0x7/0xb

```

=====

```

Code: 00 00 89 c8 ef c3 0f c9 89 0a c3 57 3d ff ff 03 00 53 89 d7 89 c3 89 ca 77 15 66 31 c0 3d 00 00 01 00
74 04 0f 0b eb fe 0f b7 d3 <f3> 6c eb 0a 4a 78 07 8a 03 88 07 47 eb f6 5b 5f c3 57 3d ff ff

```

```

EIP: [<c10a6d6f>] ioread8_rep+0x20/0x31 SS:ESP 0068:c3145aec

```

===

This is obviously not good.

Since the last time this driver was posted it changed considerably and as one of the chances it's now requesting just one hardware sector ("frame") at a time from the drive as requesting multiple didn't actually work --- I seemed to have fouled up earlier tests somehow.

## [Q] Bio traversal trouble?

The driver could let the block layer handle just doing one frame at a time by completing only one frame (4 sectors) each time but this is not a good idea. Things are somewhat timing sensitive since if you're too late the next sector has passed below you (and if you're too early you're overrunning the drive) causing throughput to plummet.

I'd also simply like to understand it, so this is doing a manual bio traversal, requesting frames from the hardware as it goes along. The driver's main request function is:

```
static void mitsumi_request(struct request_queue *q)
{
    struct request *req;

    while ((req = elv_next_request(q)) {
        struct bio *bio;
        int sectors = 0;

        if (!blk_fs_request(req)) {
            end_request(req, 0);
            continue;
        }
        if (rq_data_dir(req) != READ) {
            printk(KERN_WARNING
                "mitsumi: non-read request to CD\n");
            end_request(req, 0);
            continue;
        }
        spin_unlock_irq(q->queue_lock);
        rq_for_each_bio(bio, req) {
            unsigned int bytes;

            bytes = mitsumi_read(req->rq_disk->private_data, bio);
            sectors += bytes >> 9;

            if (bytes != bio->bi_size)
                break;
        }
        spin_lock_irq(q->queue_lock);
        if (!sectors || end_that_request_first(req, 1, sectors)) {
            end_request(req, 0);
            continue;
        }
        blkdev_dequeue_request(req);
        end_that_request_last(req, 1);
    }
}
```

That is, it's looping around doing one bio at a time, completing the number of sectors that were done and failing the sector that had an error if any weren't. (I've had many variants of that end\_request stuff in there, none helped with this problem).

mitsumi\_read() is:

[Q] Bio traversal trouble?

## [Q] Bio traversal trouble?

```
static unsigned int mitsumi_read(struct mitsumi_cdrom *mcd, struct bio *bio)
{
    sector_t frame = sector_to_frame(bio->bi_sector);
    unsigned int bytes = 0;
    struct bio_vec *bvec;
    int segno;

    bio_for_each_segment(bvec, bio, segno) {
        unsigned char *dst;
        unsigned int len;

        dst = page_address(bvec->bv_page) + bvec->bv_offset;
        for (len = bvec->bv_len; len; len -= CD_FRAMESIZE) {
            if (mitsumi_get_frame(mcd, frame++, dst))
                goto out;

            bytes += CD_FRAMESIZE;
            dst += CD_FRAMESIZE;
        }
    }
    out:
    return bytes;
}
```

`bvec->bv_len` is guaranteed to be a multiple of the hardware blocksize (the size set with `blk_queue_hardsect_size`) meaning that "len" loop is okay, right?

`mitsumi_get_frame` is the function that requests the frame from the hardware, possibly sleeps waiting for it to arrive and then reads the frame from I/O directly to "dst" ([edit] it `_is_` legal to use `complete()` from an interrupt handler isn't it?)

Is there anything wrong in all this? The thing I could imagine is that it's not actually legal to drop the `queue_lock`, but if it's not legal to drop the lock where this does so now (back in `mitsumi_request`) it seems it wouldn't be legal to drop it anywhere since you'd only shorten the window by doing it elsewhere. If it's illegal period, this would mean this all needs a completely different setup it seems...

When it faults, it's (generally at least, I believe I saw it fault directly inside `mitsumi_read` once as well) at `__mitsumi_get_frame`, where it's trying to read the data from the drive to "dst", and "dst" seems corrupted. This in turn seems to be due to the entire bio being corrupted and the `bio_for_each_segment()` looping past the end.

The driver as it stands now is attached but most of it doesn't actually have anything to do with block, so I hope I've summarised it effectively above. Any hint you or anyone else reading this would be able to provide would be welcome.

```
Rene. diff --git a/drivers/cdrom/Kconfig b/drivers/cdrom/Kconfig
index 4b12e90..8bfdedf 100644
--- a/drivers/cdrom/Kconfig
+++ b/drivers/cdrom/Kconfig
@@ -119,6 +119,20 @@ config MCDX
```

[Q] Bio traversal trouble?

## [Q] Bio traversal trouble?

To compile this driver as a module, choose M here: the module will be called mcdx.

```
+config MITSUMI
+ tristate "New Mitsumi CDROM support"
+ depends on CD_NO_IDESCSI
+ ---help---
+ Use this driver if you want to be able to use your Mitsumi LU002S,
+ LU005S, LU006S, FX001 or FX001D CD-ROM drive.
+
+ If you say Y here, you should also say Y or M to "ISO 9660 CD-ROM
+ file system support" below, because that's the file system used on
+ CD-ROMs.
+
+ To compile this driver as a module, choose M here: the
+ module will be called mitsumi.
+
config OPTCD
tristate "Optics Storage DOLPHIN 8000AT CDROM support"
depends on CD_NO_IDESCSI
diff --git a/drivers/cdrom/Makefile b/drivers/cdrom/Makefile
index d1d1e5a..317d887 100644
--- a/drivers/cdrom/Makefile
+++ b/drivers/cdrom/Makefile
@@ -16,6 +16,7 @@ obj-$(CONFIG_CM206) += cm206.o cdrom.o
obj-$(CONFIG_GSCD) += gscd.o
obj-$(CONFIG_ISP16_CDI) += isp16.o
obj-$(CONFIG_MCDX) += mcdx.o cdrom.o
+obj-$(CONFIG_MITSUMI) += mitsumi.o cdrom.o
obj-$(CONFIG_OPTCD) += optcd.o
obj-$(CONFIG_SBPCD) += sbpcd.o cdrom.o
obj-$(CONFIG_SJCD) += sjcd.o
diff --git a/drivers/cdrom/mitsumi.c b/drivers/cdrom/mitsumi.c
new file mode 100644
index 0000000..79d95d0
--- /dev/null
+++ b/drivers/cdrom/mitsumi.c
@@ -0,0 +1,1026 @@
+/*
+ * drivers/cdrom/mitsumi.c - Mitsumi legacy CD-ROM Driver for Linux
+ *
+ * Copyright (C) 1995-1996 Heiko Schlittermann
+ * Copyright (C) 2007 Pekka Enberg
+ * Copyright (C) 2007 Rene Herman
+ *
+ * This file is released under the GPLv2.
+ */
+#include <linux/bcd.h>
+#include <linux/blkdev.h>
+#include <linux/cdrom.h>
```

[Q] Bio traversal trouble?

## [Q] Bio traversal trouble?

```
+#include <linux/delay.h>
+#include <linux/fs.h>
+#include <linux/interrupt.h>
+#include <linux/isa.h>
+#include <linux/major.h>
+#include <linux/mutex.h>
+#include <linux/slab.h>
+#include <linux/wait.h>
+
+MODULE_DESCRIPTION("Mitsumi legacy CD-ROM Driver");
+MODULE_LICENSE("GPL");
+MODULE_ALIAS_BLOCKDEV_MAJOR(MITSUMI_CDROM_MAJOR);
+
+#define NR_DRIVES 1
+
+static unsigned long port[NR_DRIVES];
+static unsigned int irq[NR_DRIVES];
+
+module_param_array(port, ulong, NULL, 0444);
+module_param_array(irq, uint, NULL, 0444);
+
+/*
+ * The known drive commands
+ */
+enum { /* LU002S LU005S LU006S FX001 FX001D */
+  CMD_GET_VOLUME_INFO = 0x10, /* . Y Y Y Y */
+  CMD_GET_Q_CHANNEL = 0x20, /* . Y Y Y Y */
+  CMD_GET_SENSE_DATA = 0x30, /* . . . . . */
+  CMD_GET_STATUS = 0x40, /* . Y Y Y Y */
+  CMD_SET_MODE = 0x50, /* . Y Y Y Y */
+  CMD_SOFT_RESET = 0x60, /* . . . . . */
+  CMD_STOP_AUDIO = 0x70, /* . Y Y Y Y */
+  CMD_STOP_AUDIO_TIME = 0x80, /* . . . . . */
+  CMD_GET_VOLUME = 0x8e, /* . . . . . */
+  CMD_CONFIG_DRIVE = 0x90, /* . Y Y Y Y */
+  CMD_SET_DRIVE_MODE = 0xa0, /* . . . . . */
+  CMD_READ_UPC = 0xa2, /* . . . . . */
+  CMD_SET_VOLUME = 0xae, /* . . . . . */
+  CMD_READ_1 = 0xb0, /* . . . . . */
+  CMD_READ_PLAY = 0xc0, /* . Y Y Y Y */
+  CMD_READ_DOUBLE = 0xc1, /* N N N N Y */
+  CMD_GET_DRIVE_MODE = 0xc2, /* . Y Y Y Y */
+  CMD_READ = 0xc3, /* . . . . . */
+  CMD_SET_INTERLEAVE = 0xc8, /* . . . . . */
+  CMD_GET_VERSION = 0xdc, /* . . . . . */
+  CMD_STOP = 0xf0, /* . Y Y Y Y */
+  CMD_EJECT = 0xf6, /* N N N . . */
+  CMD_CLOSE_TRAY = 0xf8, /* N N N . . */
+  CMD_LOCK_DOOR = 0xfe, /* N N N . . */
+};
+
```

[Q] Bio traversal trouble?

## [Q] Bio traversal trouble?

```
+/*
+ * The SET_MODE argument
+ */
+#define MODE_TESTMODE 0x80 /* DATALENGTH setting ignored */
+#define MODE_DATALENGTH 0x40 /* Read raw (2352 byte) sectors */
+#define MODE_ECCMODE 0x20 /* Do not use secondary ECC */
+#define MODE_SPINDOWN 0x08 /* Spindown */
+#define MODE_GET_TOC 0x04 /* Next Q channel read gets TOC */
+#define MODE_MUTEDATA 0x01 /* Do not playback data as audio */
+
+/*
+ * The CONFIG_DRIVE argument
+ */
+#define CONFIG_IRQFLAGS 0x10 /* Set IRQ flags */
+#define CONFIG_TIMEOUT 0x08 /* Set DMA timeout */
+#define CONFIG_UPCFLAG 0x04 /* Next command will be READ_UPC */
+#define CONFIG_DMAMODE 0x02 /* Set DMA mode */
+#define CONFIG_LENGTH 0x01 /* Set MSB/LSB of DMA block size */
+
+
+#define CONFIG_ERRIRQ 0x04 /* IRQ on error */
+#define CONFIG_POSTIRQ 0x02 /* IRQ on data transferred */
+#define CONFIG_PREIRQ 0x01 /* IRQ on data ready */
+
+/*
+ * The LOCK_DOOR argument
+ */
+enum {
+ DOOR_UNLOCK = 0,
+ DOOR_LOCK = 1,
+ };
+
+/*
+ * The device registers
+ */
+enum {
+ RREG_DATA = 0,
+ RREG_STATUS = 1,
+
+ WREG_DATA = 0,
+ WREG_RESET = 1,
+ WREG_HW_CONFIG = 2,
+ WREG_CHANNEL = 3,
+ };
+
+/*
+ * The status byte returned from every command; set if description is true
+ */
+#define STATUS_OPEN 0x80 /* Door is open */
+#define STATUS_DISKSET 0x40 /* Disk set (recognized) */
+#define STATUS_CHANGED 0x20 /* Disk was changed */
+#define STATUS_CHECK 0x10 /* Disk rotates, servo is on */
```

[Q] Bio traversal trouble?

## [Q] Bio traversal trouble?

```
+ #define STATUS_AUDIOTR 0x08 /* Current track is audio */
+ #define STATUS_RDERR 0x04 /* Read error, refer SENSE KEY */
+ #define STATUS_AUDIOBS 0x02 /* Currently playing audio */
+ #define STATUS_CMDERR 0x01 /* Command, param or format error */
+
+ /*
+ * The status byte available from RREG_STATUS; reset if description is true
+ */
+ #define STATUS_DOOR 0x10 /* Door is open */
+ #define STATUS_STEN 0x04 /* I/O base contains status */
+ #define STATUS_DTEN 0x02 /* I/O base contains data */
+
+ struct mitsumi_cdrom {
+ void __iomem *ioaddr;
+ struct mutex mutex;
+
+ int audiostatus;
+ int uptodate;
+ int media_changed;
+
+ struct completion *io_done;
+
+ struct cdrom_tochdr header;
+ struct cdrom_tocentry *toc;
+ struct cdrom_tocentry leadout;
+ struct cdrom_msf fragment;
+
+ struct gendisk *disk;
+ struct cdrom_device_info info;
+ };
+
+ static inline sector_t frame_to_sector(sector_t frame)
+ {
+ return (CD_FRAMESIZE >> 9) * frame;
+ }
+
+ static inline sector_t sector_to_frame(sector_t sector)
+ {
+ return (sector << 9) / CD_FRAMESIZE;
+ }
+
+ static void frame_to_msf(sector_t frame, struct cdrom_msf0 *msf)
+ {
+ frame += CD_MSF_OFFSET;
+ msf->frame = frame % CD_FRAMES;
+ frame /= CD_FRAMES;
+ msf->second = frame % CD_SECS;
+ msf->minute = frame / CD_SECS;
+ }
+
+ static sector_t msf_to_frame(const struct cdrom_msf0 *msf)
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+{
+ return (CD_SECS * msf->minute + msf->second) * CD_FRAMES +
+ msf->frame - CD_MSF_OFFSET;
+}
+
+/*
+ * LOCKING: mutex_lock(&mcd->mutex)
+ */
+static void __mitsumi_get_frame(struct mitsumi_cdrom *mcd, unsigned char *dst)
+{
+ ioread8_rep(mcd->ioaddr + RREG_DATA, dst, CD_FRAMESIZE);
+ /*
+ * Allow the drive some time to recover
+ */
+ udelay(100);
+}
+
+static void __mitsumi_write_cmd(struct mitsumi_cdrom *mcd, unsigned char cmd,
+ unsigned char *arg, size_t size)
+{
+ iowrite8(cmd, mcd->ioaddr + WREG_DATA);
+ while (size--)
+ iowrite8(*arg++, mcd->ioaddr + WREG_DATA);
+}
+
+static int mitsumi_get_value(struct mitsumi_cdrom *mcd, unsigned char *value,
+ unsigned long msec)
+{
+ unsigned long timeout = jiffies + msec_to_jiffies(msec);
+ int err = 0;
+
+ while (ioread8(mcd->ioaddr + RREG_STATUS) & STATUS_STEN) {
+ if (time_after(jiffies, timeout)) {
+ err = -EIO;
+ goto out;
+ }
+ cond_resched();
+ }
+ *value = ioread8(mcd->ioaddr + RREG_DATA);
+ out:
+ return err;
+}
+
+static int mitsumi_write_cmd(struct mitsumi_cdrom *mcd, unsigned char cmd,
+ unsigned char *arg, size_t size,
+ unsigned long msec)
+{
+ unsigned char status;
+ int err;
+
+ __mitsumi_write_cmd(mcd, cmd, arg, size);
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+
+ err = mitsumi_get_value(mcd, &status, msec);
+ if (err)
+ goto out;
+
+ if (status & STATUS_CMDERR) {
+ printk(KERN_ERR "mitsumi: command error: cmd=%02x, "
+ "status=%02x\n", cmd, status);
+ err = -EIO;
+ goto out;
+ }
+ if (status & STATUS_CHANGED) {
+ mcd->uptodate = 0;
+ mcd->media_changed = 1;
+ }
+ if (!(status & STATUS_AUDIOTR)) {
+ mcd->audiostatus = CDROM_AUDIO_INVALID;
+ goto out;
+ }
+ if (status & STATUS_AUDIOBS) {
+ mcd->audiostatus = CDROM_AUDIO_PLAY;
+ goto out;
+ }
+ if (mcd->audiostatus == CDROM_AUDIO_PLAY) {
+ mcd->audiostatus = CDROM_AUDIO_COMPLETED;
+ goto out;
+ }
+ if (mcd->audiostatus == CDROM_AUDIO_INVALID)
+ mcd->audiostatus = CDROM_AUDIO_NO_STATUS;
+ out:
+ return err;
+ }
+
+static int mitsumi_read_cmd(struct mitsumi_cdrom *mcd, unsigned char cmd,
+ unsigned char *buf, size_t size,
+ unsigned long msec)
+{
+ int err = mitsumi_write_cmd(mcd, cmd, NULL, 0, msec);
+
+ while (!err && size-->0)
+ err = mitsumi_get_value(mcd, buf++, msec);
+
+ return err;
+ }
+
+static int mitsumi_read_subchnl(struct mitsumi_cdrom *mcd,
+ struct cdrom_subchnl *q)
+{
+ unsigned char buf[10];
+ int err;
+
+ }
```

[Q] Bio traversal trouble?

## [Q] Bio traversal trouble?

```
+ err = mitsumi_read_cmd(mcd, CMD_GET_Q_CHANNEL, buf, sizeof buf, 2000);
+ if (err)
+ goto out;
+
+ q->cdsc_format = CDRROM_MSF;
+ q->cdsc_audiostatus = mcd->audiostatus;
+
+ q->cdsc_adr = buf[0];
+ q->cdsc_ctrl = buf[0] >> 4;
+ q->cdsc_trk = BCD2BIN(buf[1]);
+ q->cdsc_ind = BCD2BIN(buf[2]);
+
+ q->cdsc_reladdr.msf.minute = BCD2BIN(buf[3]);
+ q->cdsc_reladdr.msf.second = BCD2BIN(buf[4]);
+ q->cdsc_reladdr.msf.frame = BCD2BIN(buf[5]);
+
+ q->cdsc_absaddr.msf.minute = BCD2BIN(buf[7]);
+ q->cdsc_absaddr.msf.second = BCD2BIN(buf[8]);
+ q->cdsc_absaddr.msf.frame = BCD2BIN(buf[9]);
+ out:
+ return err;
+}
+
+static int __mitsumi_read_toc(struct mitsumi_cdrom *mcd)
+{
+ int tracks = mcd->header.cdth_trk1 - mcd->header.cdth_trk0 + 1;
+ int tries, err;
+
+ kfree(mcd->toc);
+ mcd->toc = kzalloc(tracks * sizeof *mcd->toc, GFP_KERNEL);
+ if (!mcd->toc) {
+ err = -ENOMEM;
+ goto out;
+ }
+ for (tries = 300; tries; tries--) { /* why 300? */
+ struct cdrom_subchnl q;
+ int i;
+
+ err = mitsumi_read_subchnl(mcd, &q);
+ if (err)
+ goto out;
+
+ if (q.cdsc_trk != 0)
+ continue;
+
+ i = q.cdsc_ind;
+ if (i < mcd->header.cdth_trk0 || i > mcd->header.cdth_trk1)
+ continue;
+
+ i -= mcd->header.cdth_trk0;
+ if (mcd->toc[i].cdte_track != 0)
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ continue;
+
+ mcd->toc[i].cdte_track = q.cdsc_ind;
+ mcd->toc[i].cdte_adr = q.cdsc_adr;
+ mcd->toc[i].cdte_ctrl = q.cdsc_ctrl;
+ mcd->toc[i].cdte_format = q.cdsc_format;
+ mcd->toc[i].cdte_addr = q.cdsc_absaddr;
+
+ if (!--tracks)
+ goto out;
+ }
+ err = -EIO;
+ out:
+ return err;
+}
+
+static int mitsumi_set_mode(struct mitsumi_cdrom *mcd, unsigned char mode)
+{
+ mode |= MODE_MUTEDATA;
+
+ return mitsumi_write_cmd(mcd, CMD_SET_MODE, &mode, 1, 5000);
+}
+
+static int mitsumi_stop_audio(struct mitsumi_cdrom *mcd)
+{
+ return mitsumi_write_cmd(mcd, CMD_STOP_AUDIO, NULL, 0, 5000);
+}
+
+static int mitsumi_read_toc(struct mitsumi_cdrom *mcd)
+{
+ int err;
+
+ err = mitsumi_stop_audio(mcd);
+ if (err)
+ goto out;
+
+ err = mitsumi_set_mode(mcd, MODE_GET_TOC);
+ if (err)
+ goto out;
+
+ err = __mitsumi_read_toc(mcd);
+ if (err)
+ goto out;
+
+ err = mitsumi_set_mode(mcd, 0);
+ out:
+ return err;
+}
+
+static int mitsumi_read_header(struct mitsumi_cdrom *mcd)
+{
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ unsigned char buf[8];
+ int err;
+
+ err = mitsumi_read_cmd(mcd, CMD_GET_VOLUME_INFO, buf, sizeof buf, 2000);
+ if (err)
+ goto out;
+
+ mcd->header.cdth_trk0 = BCD2BIN(buf[0]);
+ mcd->header.cdth_trk1 = BCD2BIN(buf[1]);
+
+ if (mcd->header.cdth_trk1 < mcd->header.cdth_trk0) {
+ err = -EINVAL;
+ goto out;
+ }
+
+ mcd->leadout.cdte_track = CDROM_LEADOUT;
+ mcd->leadout.cdte_format = CDROM_MSF;
+
+ mcd->leadout.cdte_addr.msf.minute = BCD2BIN(buf[2]);
+ mcd->leadout.cdte_addr.msf.second = BCD2BIN(buf[3]);
+ mcd->leadout.cdte_addr.msf.frame = BCD2BIN(buf[4]);
+
+ /*
+ * First sector MSF in 5-7
+ */
+ out:
+ return err;
+}
+
+static int mitsumi_config_drive(struct mitsumi_cdrom *mcd)
+{
+ unsigned char arg[2];
+ int err;
+
+ arg[0] = CONFIG_DMAMODE;
+ arg[1] = 0;
+
+ err = mitsumi_write_cmd(mcd, CMD_CONFIG_DRIVE, arg, sizeof arg, 1000);
+ if (err)
+ goto out;
+
+ arg[0] = CONFIG_IRQFLAGS;
+ arg[1] = CONFIG_PREIRQ;
+
+ err = mitsumi_write_cmd(mcd, CMD_CONFIG_DRIVE, arg, sizeof arg, 1000);
+ out:
+ return err;
+}
+
+static int mitsumi_update(struct mitsumi_cdrom *mcd)
+{
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ sector_t capacity = 0;
+ int err;
+
+ err = mitsumi_read_header(mcd);
+ if (err)
+ goto out;
+
+ err = mitsumi_read_toc(mcd);
+ if (err)
+ goto out;
+
+ err = mitsumi_config_drive(mcd);
+ if (err)
+ goto out;
+
+ capacity = frame_to_sector(msf_to_frame(&mcd->leadout.cdte_addr.msf));
+ out:
+ set_capacity(mcd->disk, capacity);
+ return err;
+}
+
+static int mitsumi_open(struct cdrom_device_info *cdi, int purpose)
+{
+ /*
+ * Uniform CD-ROM driver calls into audio_ioctl first even for data
+ * (cdrom_count_tracks) so we're good to go
+ */
+ return 0;
+}
+
+static void mitsumi_release(struct cdrom_device_info *cdi)
+{
+}
+
+static int mitsumi_get_status(struct mitsumi_cdrom *mcd)
+{
+ return mitsumi_write_cmd(mcd, CMD_GET_STATUS, NULL, 0, 5000);
+}
+
+static int mitsumi_media_changed(struct cdrom_device_info *cdi, int disc_nr)
+{
+ struct mitsumi_cdrom *mcd = cdi->handle;
+ int ret;
+
+ mutex_lock(&mcd->mutex);
+ ret = mcd->media_changed;
+
+ if (!ret && !mitsumi_get_status(mcd))
+ ret = mcd->media_changed;
+
+ mcd->media_changed = 0;
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ mutex_unlock(&mcd->mutex);
+ return ret;
+}
+
+static int mitsumi_read_tochdr(struct mitsumi_cdrom *mcd,
+ struct cdrom_tochdr *header)
+{
+ *header = mcd->header;
+ return 0;
+}
+
+static int mitsumi_read_tocentry(struct mitsumi_cdrom *mcd,
+ struct cdrom_tocentry *entry)
+{
+ int err = 0;
+
+ if (entry->cdte_track == CDROM_LEADOUT) {
+ *entry = mcd->leadout;
+ goto out;
+ }
+ if (entry->cdte_track < mcd->header.cdth_trk0 ||
+ entry->cdte_track > mcd->header.cdth_trk1) {
+ err = -EINVAL;
+ goto out;
+ }
+ *entry = mcd->toc[entry->cdte_track - mcd->header.cdth_trk0];
+ out:
+ return err;
+}
+
+static int mitsumi_start(struct mitsumi_cdrom *mcd)
+{
+ /*
+ * Drive spins up automatically only
+ */
+ return 0;
+}
+
+static int mitsumi_stop(struct mitsumi_cdrom *mcd)
+{
+ return mitsumi_write_cmd(mcd, CMD_STOP, NULL, 0, 2000);
+}
+
+static int mitsumi_pause(struct mitsumi_cdrom *mcd)
+{
+ struct cdrom_subchnl q;
+ int err;
+
+ if (mcd->audiostatus != CDROM_AUDIO_PLAY) {
+ err = -EINVAL;
+ goto out;
+ }
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ }
+ err = mitsumi_stop_audio(mcd);
+ if (err)
+ goto out;
+
+ err = mitsumi_read_subchnl(mcd, &q);
+ if (err)
+ goto out;
+
+ mcd->fragment.cdmsf_min0 = q.cdsc_absaddr.msf.minute;
+ mcd->fragment.cdmsf_sec0 = q.cdsc_absaddr.msf.second;
+ mcd->fragment.cdmsf_frame0 = q.cdsc_absaddr.msf.frame;
+
+ mcd->audiostatus = CDROM_AUDIO_PAUSED;
+ out:
+ return err;
+}
+
+static int mitsumi_play(struct mitsumi_cdrom *mcd)
+{
+ unsigned char arg[6];
+
+ arg[0] = BIN2BCD(mcd->fragment.cdmsf_min0);
+ arg[1] = BIN2BCD(mcd->fragment.cdmsf_sec0);
+ arg[2] = BIN2BCD(mcd->fragment.cdmsf_frame0);
+ arg[3] = BIN2BCD(mcd->fragment.cdmsf_min1);
+ arg[4] = BIN2BCD(mcd->fragment.cdmsf_sec1);
+ arg[5] = BIN2BCD(mcd->fragment.cdmsf_frame1);
+
+ return mitsumi_write_cmd(mcd, CMD_READ_PLAY, arg, sizeof arg, 5000);
+}
+
+static int mitsumi_resume(struct mitsumi_cdrom *mcd)
+{
+ int err;
+
+ if (mcd->audiostatus != CDROM_AUDIO_PAUSED) {
+ err = -EINVAL;
+ goto out;
+ }
+ err = mitsumi_play(mcd);
+ out:
+ return err;
+}
+
+static int mitsumi_play_msf(struct mitsumi_cdrom *mcd, struct cdrom_msf *msf)
+{
+ mcd->fragment = *msf;
+
+ return mitsumi_play(mcd);
+}
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+
+static int mitsumi_play_trkind(struct mitsumi_cdrom *mcd, struct cdrom_ti *ti)
+{
+ struct cdrom_tocentry *entry;
+ struct cdrom_msf0 msf;
+ int err;
+
+ if (ti->cdti_trk0 < mcd->header.cdth_trk0 ||
+ ti->cdti_trk1 < ti->cdti_trk0 ||
+ ti->cdti_trk1 > mcd->header.cdth_trk1) {
+ err = -EINVAL;
+ goto out;
+ }
+ entry = &mcd->toc[ti->cdti_trk0 - mcd->header.cdth_trk0];
+
+ mcd->fragment.cdmsf_min0 = entry->cdte_addr.msf.minute;
+ mcd->fragment.cdmsf_sec0 = entry->cdte_addr.msf.second;
+ mcd->fragment.cdmsf_frame0 = entry->cdte_addr.msf.frame;
+
+ if (ti->cdti_trk1 < mcd->header.cdth_trk1)
+ entry = &mcd->toc[ti->cdti_trk1 - mcd->header.cdth_trk0 + 1];
+ else
+ entry = &mcd->leadout;
+
+ frame_to_msf(msf_to_frame(&entry->cdte_addr.msf) - 1, &msf);
+
+ mcd->fragment.cdmsf_min1 = msf.minute;
+ mcd->fragment.cdmsf_sec1 = msf.second;
+ mcd->fragment.cdmsf_frame1 = msf.frame;
+
+ err = mitsumi_play(mcd);
+ out:
+ return err;
+}
+
+static int mitsumi_audio_ioctl(struct cdrom_device_info *cdi, unsigned int cmd,
+ void *arg)
+{
+ struct mitsumi_cdrom *mcd = cdi->handle;
+ int err;
+
+ mutex_lock(&mcd->mutex);
+ if (mcd->uptodate) {
+ err = mitsumi_get_status(mcd);
+ if (err)
+ goto out;
+ }
+ if (!mcd->uptodate) {
+ err = mitsumi_update(mcd);
+ if (err)
+ goto out;
+ }
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+
+ mcd->uptodate = 1;
+ }
+ switch (cmd) {
+ case CDROMSUBCHNL:
+ err = mitsumi_read_subchnl(mcd, arg);
+ break;
+ case CDROMREADTOCHDR:
+ err = mitsumi_read_tochdr(mcd, arg);
+ break;
+ case CDROMREADTOCENTRY:
+ err = mitsumi_read_tocentry(mcd, arg);
+ break;
+ case CDROMSTART:
+ err = mitsumi_start(mcd);
+ break;
+ case CDROMSTOP:
+ err = mitsumi_stop(mcd);
+ break;
+ case CDROMPAUSE:
+ err = mitsumi_pause(mcd);
+ break;
+ case CDROMRESUME:
+ err = mitsumi_resume(mcd);
+ break;
+ case CDROMPLAYMSF:
+ err = mitsumi_play_msf(mcd, arg);
+ break;
+ case CDROMPLAYTRKIND:
+ err = mitsumi_play_trkind(mcd, arg);
+ break;
+ default:
+ err = -ENOSYS;
+ break;
+ }
+ out:
+ mutex_unlock(&mcd->mutex);
+ return err;
+}
+
+static struct cdrom_device_ops mitsumi_dops = {
+ .open = mitsumi_open,
+ .release = mitsumi_release,
+ .media_changed = mitsumi_media_changed,
+ .audio_ioctl = mitsumi_audio_ioctl,
+ .capability = CDC_MEDIA_CHANGED | CDC_PLAY_AUDIO
+};
+
+static irqreturn_t mitsumi_intr(int irq, void *dev_id)
+{
+ struct mitsumi_cdrom *mcd = dev_id;
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ unsigned char status;
+
+ status = ioread8(mcd->ioaddr + RREG_STATUS);
+ if (status & STATUS_DTEN) {
+ /*
+ * Requested data is not ready
+ */
+ if (status & STATUS_STEN)
+ printk(KERN_INFO
+ "mitsumi: ambiguous interrupt (status %#x)\n",
+ status);
+ else
+ printk(KERN_INFO "mitsumi: interrupt (status %#x)\n",
+ ioread8(mcd->ioaddr + RREG_DATA));
+ } else {
+ struct completion *io_done = mcd->io_done;
+
+ if (io_done) {
+ mcd->io_done = NULL;
+ complete(io_done);
+ }
+ }
+ return IRQ_HANDLED;
+}
+
+static int mitsumi_get_frame(struct mitsumi_cdrom *mcd, sector_t frame,
+ unsigned char *dst)
+{
+ struct completion io_wait;
+ struct cdrom_msf0 msf;
+ unsigned char arg[6];
+ int err = 0;
+
+ init_completion(&io_wait);
+ frame_to_msf(frame, &msf);
+
+ arg[0] = BIN2BCD(msf.minute);
+ arg[1] = BIN2BCD(msf.second);
+ arg[2] = BIN2BCD(msf.frame);
+ arg[3] = 0;
+ arg[4] = 0;
+ arg[5] = 1;
+
+ mutex_lock(&mcd->mutex);
+ mcd->io_done = &io_wait;
+ __mitsumi_write_cmd(mcd, CMD_READ_PLAY, arg, sizeof arg);
+ /*
+ * Wait for the interrupt handler to notify us when the I/O completes
+ */
+ if (!wait_for_completion_timeout(&io_wait, msecs_to_jiffies(5000))) {
+ printk(KERN_ERR "mitsumi: I/O wait timed out\n");
```

[Q] Bio traversal trouble?

## [Q] Bio traversal trouble?

```
+ err = -EIO;
+ goto out;
+
+ }
+ __mitsumi_get_frame(mcd, dst);
+ out:
+ mutex_unlock(&mcd->mutex);
+ return err;
+}
+
+static unsigned int mitsumi_read(struct mitsumi_cdrom *mcd, struct bio *bio)
+{
+ sector_t frame = sector_to_frame(bio->bi_sector);
+ unsigned int bytes = 0;
+ struct bio_vec *bvec;
+ int segno;
+
+ bio_for_each_segment(bvec, bio, segno) {
+ unsigned char *dst;
+ unsigned int len;
+
+ dst = page_address(bvec->bv_page) + bvec->bv_offset;
+ for (len = bvec->bv_len; len; len -= CD_FRAMESIZE) {
+ if (mitsumi_get_frame(mcd, frame++, dst))
+ goto out;
+
+ bytes += CD_FRAMESIZE;
+ dst += CD_FRAMESIZE;
+ }
+ }
+ out:
+ return bytes;
+}
+
+static void mitsumi_request(struct request_queue *q)
+{
+ struct request *req;
+
+ while ((req = elv_next_request(q))) {
+ struct bio *bio;
+ int sectors = 0;
+
+ if (!blk_fs_request(req)) {
+ end_request(req, 0);
+ continue;
+ }
+ if (rq_data_dir(req) != READ) {
+ printk(KERN_WARNING
+ "mitsumi: non-read request to CD\n");
+ end_request(req, 0);
+ continue;
+ }
+ }
```

## [Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ }
+ spin_unlock_irq(q->queue_lock);
+ rq_for_each_bio(bio, req) {
+ unsigned int bytes;
+
+ bytes = mitsumi_read(req->rq_disk->private_data, bio);
+ sectors += bytes >> 9;
+
+ if (bytes != bio->bi_size)
+ break;
+ }
+ spin_lock_irq(q->queue_lock);
+ if (!sectors || end_that_request_first(req, 1, sectors)) {
+ end_request(req, 0);
+ continue;
+ }
+ blkdev_dequeue_request(req);
+ end_that_request_last(req, 1);
+ }
+ }
+
+static int mitsumi_block_open(struct inode *inode, struct file *file)
+{
+ struct mitsumi_cdrom *mcd = inode->i_bdev->bd_disk->private_data;
+
+ return cdrom_open(&mcd->info, inode, file);
+}
+
+static int mitsumi_block_release(struct inode *inode, struct file *file)
+{
+ struct mitsumi_cdrom *mcd = inode->i_bdev->bd_disk->private_data;
+
+ return cdrom_release(&mcd->info, file);
+}
+
+static int mitsumi_block_ioctl(struct inode *inode, struct file *file,
+ unsigned cmd, unsigned long arg)
+{
+ struct mitsumi_cdrom *mcd = inode->i_bdev->bd_disk->private_data;
+
+ return cdrom_ioctl(file, &mcd->info, inode, cmd, arg);
+}
+
+static int mitsumi_block_media_changed(struct gendisk *disk)
+{
+ struct mitsumi_cdrom *mcd = disk->private_data;
+
+ return cdrom_media_changed(&mcd->info);
+}
+
+static struct block_device_operations mitsumi_bdops = {
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ .owner = THIS_MODULE,
+ .open = mitsumi_block_open,
+ .release = mitsumi_block_release,
+ .ioctl = mitsumi_block_ioctl,
+ .media_changed = mitsumi_block_media_changed,
+};
+
+static int mitsumi_read_version(struct mitsumi_cdrom *mcd, unsigned char *ver,
+ unsigned char *rev)
+{
+ unsigned char buf[2];
+ int err;
+
+ err = mitsumi_read_cmd(mcd, CMD_GET_VERSION, buf, sizeof buf, 2000);
+ if (err)
+ goto out;
+
+ *ver = buf[0];
+ *rev = buf[1];
+ out:
+ return err;
+}
+
+static void mitsumi_reset(struct mitsumi_cdrom *mcd)
+{
+ iowrite8(0, mcd->iaddr + WREG_CHANNEL); /* no dma, no irq */
+ iowrite8(0, mcd->iaddr + WREG_RESET); /* hardware reset */
+
+ /*
+ * Nothing to poll, no interrupt, ...
+ */
+ msleep(500);
+}
+
+static int __devinit mitsumi_match(struct device *dev, unsigned int id)
+{
+ int match = port[id] != 0 && irq[id] != 0;
+
+ if (!match)
+ printk(KERN_ERR "mitsumi: please specify port and irq\n");
+
+ return match;
+}
+
+static int __devinit mitsumi_probe(struct device *dev, unsigned int id)
+{
+ struct mitsumi_cdrom *mcd;
+ unsigned char ver;
+ unsigned char rev;
+ char *model;
+ struct gendisk *disk;
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ int err;
+
+ mcd = kzalloc(sizeof *mcd, GFP_KERNEL);
+ if (!mcd) {
+ err = -ENOMEM;
+ goto out;
+ }
+ if (!request_region(port[id], 4, "mitsumi")) {
+ err = -EBUSY;
+ goto out_free;;
+ }
+ mcd->ioaddr = ioport_map(port[id], 4);
+
+ mitsumi_reset(mcd);
+
+ mutex_init(&mcd->mutex);
+ mcd->audiostatus = CDROM_AUDIO_INVALID;
+
+ err = mitsumi_read_version(mcd, &ver, &rev);
+ if (err)
+ goto out_release_region;
+
+ switch (ver) {
+ case 'M':
+ model = rev <= 2 ? "LU002S" : rev <= 5 ? "LU005S" : "LU006S";
+ break;
+ case 'F':
+ model = "FX001";
+ break;
+
+ case 'D':
+ model = "FX001D";
+ break;
+ default:
+ printk(KERN_WARNING "mitsumi: unknown firmware %c%d, driver "
+ "not initialized\n", ver, rev);
+ goto out_release_region;
+ };
+
+ err = request_irq(irq[id], mitsumi_intr, IRQF_DISABLED, "mitsumi", mcd);
+ if (err)
+ goto out_release_region;
+
+ disk = alloc_disk(1);
+ if (!disk) {
+ err = -ENOMEM;
+ goto out_free_irq;
+ }
+ mcd->disk = disk;
+ disk->private_data = mcd;
+
+ 
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ disk->major = MITSUMI_CDROM_MAJOR;
+ disk->first_minor = id;
+ disk->fops = &mitsumi_bdops;
+ disk->flags = GENHD_FL_CD;
+ sprintf(disk->disk_name, "mitsumi%u", id);
+
+ disk->queue = blk_init_queue(mitsumi_request, NULL);
+ if (!disk->queue) {
+ err = -ENOMEM;
+ goto out_put_disk;
+ }
+ blk_queue_hardsect_size(disk->queue, CD_FRAMESIZE);
+
+ mcd->info.ops = &mitsumi_dops;
+ mcd->info.speed = 1;
+ mcd->info.capacity = 1;
+ mcd->info.handle = mcd;
+ strcpy(mcd->info.name, disk->disk_name);
+
+ err = register_cdrom(&mcd->info);
+ if (err)
+ goto out_cleanup_queue;
+
+ printk(KERN_INFO "mitsumi: found %s drive (firmware %c%d) at %#lx, "
+ "irq %d\n", model, ver, rev, port[id], irq[id]);
+
+ add_disk(disk);
+ dev_set_drvdata(dev, mcd);
+ return 0;
+
+ out_cleanup_queue:
+ blk_cleanup_queue(disk->queue);
+ out_put_disk:
+ put_disk(disk);
+ out_free_irq:
+ free_irq(irq[id], mcd);
+ out_release_region:
+ ioport_unmap(mcd->ioaddr);
+ release_region(port[id], 4);
+ out_free:
+ kfree(mcd);
+ out:
+ return err;
+}
+
+static int __devexit mitsumi_remove(struct device *dev, unsigned int id)
+{
+ struct mitsumi_cdrom *mcd = dev_get_drvdata(dev);
+
+ dev_set_drvdata(dev, NULL);
+ del_gendisk(mcd->disk);
```

[Q] Bio traversal trouble?

[Q] Bio traversal trouble?

```
+ unregister_cdrom(&mcd->info);
+ blk_cleanup_queue(mcd->disk->queue);
+ put_disk(mcd->disk);
+ free_irq(irq[id], mcd);
+ ioport_unmap(mcd->ioaddr);
+ release_region(port[id], 4);
+ kfree(mcd->toc);
+ kfree(mcd);
+ return 0;
+}
+
+static struct isa_driver mitsumi_driver = {
+ .match = mitsumi_match,
+ .probe = mitsumi_probe,
+ .remove = __devexit_p(mitsumi_remove),
+
+ .driver = {
+ .name = "mitsumi"
+ }
+};
+
+static int __init mitsumi_init(void)
+{
+ int err;
+
+ err = register_blkdev(MITSUMI_CDROM_MAJOR, "mitsumi");
+ if (err)
+ goto out;
+
+ err = isa_register_driver(&mitsumi_driver, NR_DRIVES);
+ if (err)
+ unregister_blkdev(MITSUMI_CDROM_MAJOR, "mitsumi");
+ out:
+ return err;
+}
+
+static void __exit mitsumi_exit(void)
+{
+ isa_unregister_driver(&mitsumi_driver);
+ unregister_blkdev(MITSUMI_CDROM_MAJOR, "mitsumi");
+}
+
+module_init(mitsumi_init);
+module_exit(mitsumi_exit);
```