ATAPI

From OSDev Wiki

ATAPI refers to devices that use the Packet Interface of the ATA6 (or higher) standard command set. It is basically a way to issue SCSI commands to a CDROM, CD-RW, DVD, or tape drive, attached to the ATA bus.

ATAPI uses a very small number of ATA commands. The most important are the PACKET command (0xA0), and IDENTIFY PACKET DEVICE (0xA1).

Contents

PACKET command (0xA0)

Each ATAPI command packet is made of a 'command byte' (from the SCSI command set -- see below for a partial list), followed by 11 'data' bytes. For instance, reading the table of contents is achieved by sending the following byte string to the device, as a "command".

```
uint8_t atapi_readtoc[]=  { 0x43 /* ATAPI_READTOC */, 0, 1, 0, 0, 0, 0, 
```

The ATA PACKET command works in three phases, in PIO mode.

Phase 1) Set up the standard ATA IO port registers with ATAPI specific values. Then Send the ATA PACKET command to the device exactly as you would with any other ATA command: outb (ATA Command Register, 0xA0)

Phase 2) Prepare to do a PIO data transfer, the normal way, to the device. When the device is ready (BSY clear, DRQ set) you send the ATAPI command string (like the one above) as a 6 word PIO transfer to the device.

Phase 3) Wait for an IRQ. When it arrives, you must read the LBA Mid and LBA High IO port registers. They tell you the packet byte count that the ATAPI drive will send to you, or must receive from you. In a loop, you transmit that number of bytes, then wait for the next IRQ.

In DMA mode, only the first two phases happen. The device handles the details of phase 3 itself, with the PCI drive controller.

IDENTIFY PACKET DEVICE command (0xA1)

This command is a "normal" ATA PIO mode command, used during initialization. It is an exact mirror of the ATA IDENTIFY command, except that it only returns information about ATAPI devices. Use it in exactly the same way as you use IDENTIFY, including the meanings of all the bits in all the 256 words of data returned.

Disc Content

Optical media and drives are governed by the MMC part of SCSI specs. They are structured in sessions and tracks. The readable entities are called Logical Track. They are contiguous strings of blocks. Some media types can bear several sessions with several tracks each, others bear only one session with one track.

For more informations about assessing, reading and writing optical media, see the section about Readable Disc Content of article Optical Drive.

x86 Directions

Important note: on the Primary bus, the standard set of ATA IO ports is 0x1F0 through 0x1F7. In much or all of the ATAPI documentation, you will see this set of IO ports called the "Task File". The term seems very confusing.

First, you need to have a buffer. If it is going to be a DMA buffer, it needs to follow the PRD rules (see the ATA/ATAPI using DMA article). If it is going to be a PIO buffer, then you need to know the size of the buffer. Call this size "maxByteCount". It must be an unsigned word, and 0 is illegal for PIO mode. A value of zero is **mandatory** for DMA mode, no matter what size the PRD buffers are.

Assume that the command is in words Command1 through Command6. Device is the Primary slave. Select the target device by setting the master/slave bit in the Device Select Register. There are no other bits needed.

```
outb (0x1F6, slavebit<<4)
```

If the command is going to use DMA, set the Features Register to 1, otherwise 0 for PIO.

```
outb (0x1F1, isDMA)
```

The Sectorcount Register and LBA Low Register are unused currently. Send maxByteCount in LBA Mid and LBA High Registers.

```
outb (0x1F4, (maxByteCount & 0xff))
outb (0x1F5, (maxByteCount >> 8))
```

Send the ATAPI PACKET command to the Command Register

```
outb (0x1F7, 0xA0)
```

Wait for an IRQ, or poll for BSY to clear and DRQ to set.

Then send the ATAPI command as 6 words, to the data port.

```
outw (0x1F0, Command1)
outw (0x1F0, Command2)
outw (0x1F0, Command3)
outw (0x1F0, Command4)
outw (0x1F0, Command5)
outw (0x1F0, Command6)
```

Then wait for another IRQ. You cannot poll.

If this was a DMA command (isDMA == 1), then you are done. When the IRQ arrives, the transfer is complete.

If it was a PIO command, when the IRQ arrives, read the LBA Mid and LBA High Registers. This is vital. You told the drive the *maximum* amount of data to transfer at one time. Now the drive has to tell you the **actual** transfer size.

Once you have the transfer size (bytecount = LBA High << 8 | LBA Mid), do the PIO transfer.

```
wordcount = bytecount/2;
```

loop on inw(0x1F0) or outw(0x1f0) wordcount times.

If the transfer is complete, BSY and DRQ will clear. Otherwise, wait for the next IRQ, and read or write the same number of words again.

Notes: there is a possible future change planned to increase the length of ATAPI command strings to 8 words. Check the two bottom bits of ATAPI Identify word 0 to verify 6 or 8 word command size.

Once again, if you use polling to check BSY, DRQ, and ERR after sending the PACKET command, then you should probably ignore the ERR bit for the first four loops. (ATAPI calls this the "CHECK" bit, instead of ERR, but it means the same thing.)

Complete Command Set

SCSI Command Name	Command Byte (OpCode)
TEST UNIT READY	0x00
REQUEST SENSE	0x03
FORMAT UNIT	0x04
INQUIRY	0x12
START STOP UNIT	0x1B
PREVENT ALLOW MEDIUM REMOVAL	0x1E
READ FORMAT CAPACITIES	0x23
READ CAPACITY	0x25
READ (10)	0x28
WRITE (10)	0x2A
SEEK (10)	0x2B
WRITE AND VERIFY (10)	0x2E
VERIFY (10)	0x2F
SYNCHRONIZE CACHE	0x35
WRITE BUFFER	0x3B
READ BUFFER	0x3C
READ TOC/PMA/ATIP	0x43
GET CONFIGURATION	0x46
GET EVENT STATUS NOTIFICATION	0x4A
READ DISC INFORMATION	0x51
READ TRACK INFORMATION	0x52
RESERVE TRACK	0x53
SEND OPC INFORMATION	0x54
MODE SELECT (10)	0x55
REPAIR TRACK	0x58
MODE SENSE (10)	0x5A
CLOSE TRACK SESSION	0x5B
READ BUFFER CAPACITY	0x5C
SEND CUE SHEET	0x5D
REPORT LUNS	0xA0
BLANK	0xA1
SECURITY PROTOCOL IN	0xA2
SEND KEY	0xA3
REPORT KEY	0xA4
LOAD/UNLOAD MEDIUM	0xA6
SET READ AHEAD	0xA7
READ (12)	0xA8

WRITE (12)	0xAA
READ MEDIA SERIAL NUMBER / SERVICE ACTION IN (12)	0xAB / 0x01
GET PERFORMANCE	0xAC
READ DISC STRUCTURE	0xAD
SECURITY PROTOCOL OUT	0xB5
SET STREAMING	0xB6
READ CD MSF	0xB9
SET CD SPEED	0xBB
MECHANISM STATUS	0xBD
READ CD	0xBE
SEND DISC STRUCTURE	0xBF

x86 Examples

Here is an example adapted from a working driver implementation.

```
/* The default and seemingly universal sector size for CD-ROMs. */
#define ATAPI_SECTOR_SIZE 2048

/* The default ISA IRQ numbers of the ATA controllers. */
#define ATA_IRQ_PRIMARY     0x0E
#define ATA_IRQ_SECONDARY   0x0F

/* The necessary I/O ports, indexed by "bus". */
#define ATA_DATA(x)         (x)
#define ATA_FEATURES(x)     (x+1)
#define ATA_SECTOR_COUNT(x) (x+2)
#define ATA_ADDRESS1(x)     (x+3)
#define ATA_ADDRESS2(x)     (x+4)
#define ATA_ADDRESS3(x)     (x+5)
#define ATA_DRIVE_SELECT(x) (x+6)
#define ATA_COMMAND(x)      (x+7)
#define ATA_DCR(x)          (x+0x206)   /* device control register */

/* valid values for "bus" */
#define ATA_BUS_PRIMARY     0x1F0
#define ATA_BUS_SECONDARY   0x170
/* valid values for "drive" */
#define ATA_DRIVE_MASTER    0xA0
#define ATA_DRIVE_SLAVE     0xB0

/* ATA specifies a 400ns delay after drive switching -- often
 * implemented as 4 Alternative Status queries. */
#define ATA_SELECT_DELAY(bus) \
  {inb(ATA_DCR(bus));inb(ATA_DCR(bus));inb(ATA_DCR(bus));inb(ATA_DCR(bus)

/* Use the ATAPI protocol to read a single sector from the given
```

```
 * bus/drive into the buffer using logical block address lba. */
int
atapi_drive_read_sector (uint32_t bus, uint32_t drive, uint32_t lba, uint
{
        /* 0xA8 is READ SECTORS command byte. */
        uint8_t read_cmd[12] = { 0xA8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        uint8_t status;
        int size;
        /* Tell the scheduler that this process is using the ATA subsyste
        ata_grab ();
        /* Select drive (only the slave-bit is set) */
        outb (drive & (1 << 4), ATA_DRIVE_SELECT (bus));
        ATA_SELECT_DELAY (bus);       /* 400ns delay */
        outb (0x0, ATA_FEATURES (bus));       /* PIO mode */
        outb (ATAPI_SECTOR_SIZE & 0xFF, ATA_ADDRESS2 (bus));
        outb (ATAPI_SECTOR_SIZE >> 8, ATA_ADDRESS3 (bus));
        outb (0xA0, ATA_COMMAND (bus));       /* ATA PACKET command */
        while ((status = inb (ATA_COMMAND (bus))) & 0x80)     /* BUSY */
        asm volatile ("pause");
        while (!((status = inb (ATA_COMMAND (bus))) & 0x8) && !(status &
        asm volatile ("pause");
        /* DRQ or ERROR set */
        if (status & 0x1) {
        size = -1;
        goto cleanup;
        }
        read_cmd[9] = 1;              /* 1 sector */
        read_cmd[2] = (lba >> 0x18) & 0xFF;   /* most sig. byte of LBA */
        read_cmd[3] = (lba >> 0x10) & 0xFF;
        read_cmd[4] = (lba >> 0x08) & 0xFF;
        read_cmd[5] = (lba >> 0x00) & 0xFF;   /* least sig. byte of LBA
        /* Send ATAPI/SCSI command */
        outsw (ATA_DATA (bus), (uint16_t *) read_cmd, 6);
        /* Wait for IRQ that says the data is ready. */
        schedule ();
        /* Read actual size */
        size =
        (((int) inb (ATA_ADDRESS3 (bus))) << 8) |
        (int) (inb (ATA_ADDRESS2 (bus)));
        /* This example code only supports the case where the data trans
        * of one sector is done in one step. */
        ASSERT (size == ATAPI_SECTOR_SIZE);
        /* Read data. */
        insw (ATA_DATA (bus), buffer, size / 2);
        /* The controller will send another IRQ even though we've read al
        * the data we want.  Wait for it -- so it doesn't interfere with
        * subsequent operations: */
        schedule ();
        /* Wait for BSY and DRQ to clear, indicating Command Finished */
        while ((status = inb (ATA_COMMAND (bus))) & 0x88)
        asm volatile ("pause");
        cleanup:
        /* Exit the ATA subsystem */
        ata_release ();
        return size;
}
```

Detecting a Medium's Capacity

A medium is any media inserted in the ATAPI Drive, like a CD or a DVD. By using the 'SCSI Read Capacity' command, you can read the last LBA of the medium, then you calculate the medium's capacity using this relationship:

Capacity = (Last LBA + 1) * Block Size;

Last LBA and Block Size are returned after processing the command. Almost all CDs and DVDs use blocks with size of 2KB each.

Processing this command goes in the following algorithm:

- Selecting the Drive [Master/Slave].
- Waiting 400ns for select to complete.
- Setting FEATURES Register to 0 [PIO Mode].
- Setting LBA1 and LBA2 Registers to 0x0008 [Number of Bytes will be returned].
- Sending Packet Command, then Polling.
- Sending the ATAPI Packet, then polling. ATAPI packet must be 6 words long (12 bytes).
- If there isn't an error, reading 4 Words [8 bytes] from the DATA Register.

The ATAPI packet should be in this formulation:

bit→ ↓byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Operation code = 25h							
1	**LUN**			Reserved				**RelAdr**
2	**LBA** (MSB)							
3								
4								
5	**LBA** (LSB)							
6	Reserved							
7	Reserved							
8	Reserved							**PMI**
9	Control							
10	Reserved*							
11	Reserved*							

Note: The last two reserved fields are ATAPI specific. They are not part of the SCSI command packet version.

The special control fields in the CDB have the following meaning:

- **RelAdr** - indicates that the logical block address (LBA) value is relative (only used with linked commands).
- **PMI** - partial medium indicator:
  - 0 - return value for the last LBA
  - 1 - return value for the last LBA after which a substantial delay in data transfer will be encountered (e.g., the current track or cylinder)

The target will return capacity data structured as follows:

bit→ ↓byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0–3	Returned LBA							
4–7	Block length in bytes							

If there is an error after polling, there may be no medium inserted, so this command may be also used to detect whether there is a medium or not, and if there is a medium, its capacity is read.

Operating Theory

Quoting The Guide to ATA/ATAPI documentation (stanford.edu) (http://suif.stanford.edu/~csapuntz/ide.html)

> *Hosts control ATAPI devices using SCSI command packets. The SCSI command packets are transported over the ATA interface, instead of the parallel SCSI bus. This cool hack is described in ATA/ATAPI-6.* The set of SCSI command packets applicable to all SCSI devices is described in the SCSI-3 Primary Commands PDF (ftp://ftp.t10.org/t10/drafts/spc/spc-r11a.pdf) . Again, ATAPI devices don't implement all of these, so it's best to consult the associated ATAPI spec for a device. CD-ROM command packets were originally described in INF-8020 PDF (http://suif.stanford.edu/~csapuntz/specs/INF-8020.PDF) . This document contains many inaccuracies in its description of the ATA bus interface, so please double check any statements against ATA/ATAPI-6. (Unfortunately, some of those inaccuracies were implemented!)

Comments

See Also

Articles

- ATA/ATAPI Power Management
- ATA/ATAPI using DMA
- Optical Drive

External Links

- http://www.t10.org/ -- T10, the group that creates the SCSI (and therefore ATAPI) command set.
- http://www.t10.org/ftp/x3t9.2/document.87/87-106r0.txt -- Direct link to 1987 documentation of SCSI commands related to CD-ROMS. Old but useful.
- http://www.ata-atapi.com -- Public Domain C driver sourcecode, including SATA, Busmatering DMA, ATAPI -- not perfect, but good.
- The Guide to ATA/ATAPI documentation (http://suif.stanford.edu/~csapuntz/ide.html)
- Old document about ATA/ATAPI errata (http://suif.stanford.edu/~csapuntz/blackmagic.html)
- Beta release of original ATAPI spec (PDF) (http://suif.stanford.edu/~csapuntz/specs/INF-8020.PDF) Has partial list of CD-ROM ATAPI commands. **MANY** errors. If something in this document looks wrong to you, it **IS** wrong. It is still very informative.
- http://www.osta.org/specs/pdf/udf201.pdf -- UDF filesystem format PDF
- http://www.osdever.net/downloads/docs/iso9660.zip -- ISO 9660 filesystem format
- http://bmrc.berkeley.edu/people/chaffee/jolspec.html -- Joliet filesystem specification
- http://www.osdever.net/downloads/docs/susp112.zip -- Rock Ridge Filesystem Sharing Protocol (POSIX)
- http://www.osdever.net/downloads/docs/rrip112.zip -- Rock Ridge Interchange specification (POSIX)