ATA/ATAPI using DMA

From OSDev Wiki

Contents

ISA DMA

It is very important to understand that there are several different types of DMA on a machine. The DMA that is used for ATA drives is called PCI Busmastering DMA. It is very fast. It is completely different from ISA DMA, which is limited to 4MB per second. Anything you may have read about 8bit or 16bit DMA channels 0 through 7, **does not apply to disk DMA at all.** ATA singleword DMA, ATA multiword DMA, UDMA, and ADMA are all PCI Busmastering DMA types.

UDMA

From the point of view of the OS, there is only a tiny difference between "regular" (Single/Multiword) DMA and any of the UDMA modes. Most of the actual differences involve precisely how, and how fast, data is transferred between the disk drive and the disk controller. OS software has no direct control over that part of the process. The driver software issues exactly the same commands, in exactly the same way, for all the DMA methods (except ADMA).

The only control the rest of the system has, lies in the initialization of the disk drive and disk controller to mutually compatible DMA modes, during bootup. This will hopefully always be done correctly by the BIOS.

If it is not done correctly by the BIOS, there is a chance that an OS can do the job after bootup. But it is a slim chance, and it is an ugly process. If you want to try it, then read the Intel ATA Controller chip manual linked below.

The one place where UDMA makes a difference to the driver is the concept of a UDMA CRC error. These happen when data blocks get corrupted between the drive and the PCI controller. No other form of DMA uses CRC. In case of an error during the DMA transfer, the driver needs to check if the error was CRC based, and retry the transfer at least once if it was.

Verifying CRC errors

Read the PCI Configuration Space uint16_t at offset 6 (Device Status) to get the error code.

The PRD Table

You must set up at least one Physical Region Descriptor Table (PRDT) in memory per ATA bus, which contains some number of Physical Region Descriptors (PRDs). (The PRDT must be uint32_t aligned, contiguous in physical memory, and cannot cross a 64K boundary.) Then you need to store the physical address of the current PRDT in the Bus Master Register, of the Bus Mastering ATA Disk Controller on the PCI bus (see below).

PRD setup

Half of each DMA transfer is encoded in one uint64_t PRD entry (8 bytes) in the table. (Why is it only half of a DMA transfer? Because the PRD does not contain any information about which LBAs to read from/write to the disk. That information is sent in an entirely separate way to the proper disk -- see below.)

The low uint32_t is a **physical** memory address of a data buffer. (Keep in mind on 64 bit systems that this address must fit in a uint32_t!) The next uint16_t is a **byte** count (not a sector count!) of the transfer size (64K maximum per PRD transfer). A byte count of 0 means 64K. The next uint16_t is reserved (should be 0) except for the MSB. If the MSB is set, that indicates that this PRD is the last entry in the PRDT, and the entire set of transfers is complete. The data buffers cannot cross a 64K boundary, and must be contiguous in physical memory (ie. they can't be "virtual" data buffers). The byte count on the data buffers must match the number of sectors transferred by the disk. If not, the controller will error out in various ways.

The Bus Master Register

The "address" of the Bus Master Register is stored in BAR4, in the PCI Configuration Space of the disk controller. The Bus Master Register is generally a set of 16 sequential IO ports. It can also be a 16 byte memory mapped space.

Format:

```
byte offset             function
(Primary ATA bus)
0x0                     Command (byte)
0x2                     Status (byte)
0x4-0x7                 PRDT Address (uint32_t)
(Secondary ATA bus)
0x8                     Command (byte)
0xA                     Status (byte)
0xC-0xF                 PRDT Address (uint32_t)
```

The Command Byte

The command byte has only 2 operational bits. All the rest should be 0. Bit 0 (value = 1) is the Start/Stop bit. Setting the bit puts the controller in DMA mode for that ATA channel, and it starts at the beginning of the respective PRDT. Clearing the bit terminates DMA mode for that ATA channel. If the controller was in the middle of a transfer, the remaining data is thrown away. Also, the controller does not remember how far it got in the PRDT. That information is lost, if the OS does not save it. The bit must be cleared when a transfer completes.

Bit 3 (value = 8) is the Read/Write bit. This bit is a huge problem. The disk controller does not automatically detect whether the next disk operation is a read or write. You have to *tell* it, in advance, by setting this bit. Note that when reading from the disk, you must set this bit to 1, and clear it when writing to the disk. You must first stop DMA transfers (by clearing bit 0) before you can change the Read/Write bit! Please note all the bad consequences of clearing bit 0, above! The controller loses its place in the PRDT.

In essence, this means that each PRDT must consist exclusively of either read or write entries. You set the Read/Write bit in advance, then "use up" the entire PRDT -- before you can do the opposite operation.

The Status Byte

The bits in the status byte are not usually useful. However, you are required to read it after every IRQ on disk reads anyway. Reading this byte may perform a necessary final cache flush of the DMA data to memory.

- Bit 7 (Simplex operation only) is completely obsolete. However, if you want your OS to support mid 90's era hardware, this bit means you can only use one ATA bus at a time for DMA. The other can only use PIO.
- Perhaps bits 5 and 6 (values 0x20 and 0x40) will provide useful information if examined once. They are supposedly set by the BIOS if the Master or Slave drive on the bus (respectively) are capable of, and initialized for DMA operation. They are there only for your use. They serve no other purpose.
- If bit 2 (value = 4) is not set after the OS receives an IRQ, then some other device sharing the IRQ generated the IRQ -- not the disk. (You might want to reset this bit after every IRQ, by writing a 4 to the Status Byte IO port.)
- Bit 1 (value = 2) is set if any DMA memory transfer failed for any reason in this PRDT.
- Bit 0 (value = 1) is set when the bus goes into DMA mode. It is cleared when the last PRD in the table has been used up. Generally, the OS should already know that this has happened, by receiving the proper number of IRQs from the disk.

ATA/ATAPI Commands

The PCI disk controller only handles the memory half of the DMA transfer, by interpreting the PRDT. The device driver must separately tell the drive to do its half of the work.

To use DMA with an ATAPI drive, write a 1 to the Features IO port (0x1F1 on the Primary bus), and the maxbytecount should be 0 -- when issuing the PACKET command to the drive.

For ATA, for each PRD entry in the PRDT, the driver must issue a Read/Write DMA command to the disk; specifying a StartLBA and a sector count.

When the drive completes each command it sends an IRQ (possibly also when transfer in progress), to let the driver know that it is ready to transfer the next batch of data to/from the disk controller. The driver should then read the Bus Master Register Status byte.

The formats of the commands are precisely the same as for the 28 and 48 bit PIO mode Read and Write commands, except for the value sent to the "Command" IO Port.

```
Command byte:          Function:
0xC8                   Read DMA (28 bit LBA)
0x25                   Read DMA (48 bit LBA)
0xCA                   Write DMA (28 bit LBA)
0x35                   Write DMA (48 bit LBA)
```

Handling Errors

After a transfer, if the Bus Master Status byte has its ERR bit (bit 1, value = 2) set, then clear it by writing a 2 to that port. You can then read the LBA IO ports for the bus (0x1F3 through 0x1F6) to find out which sector failed.

For any error except a UDMA CRC error, it is recommended to do a Software Reset on the bus, in order to force all the devices to take the bus out of DMA mode.

Comments

See Also

Articles

- ADMA
- DMA

External Links

- Intel manual with DMA / UDMA / PCI configuration space details (http://bos.asmhackers.net/docs/ata/docs/29860001.pdf)
- Very simple original Busmaster DMA spec with PRDT spec (http://www.osdever.net/downloads/docs/idems100.zip)

This page or section is a stub. You can help the wiki by *accurately* contributing (http://wiki.osdev.org/index.php?title=ATA/ATAPI_using_DMA&action=edit) to it.

Retrieved from "http://wiki.osdev.org/index.php?title=ATA/ATAPI_using_DMA&oldid=17425"
Categories: Stubs | ATA

- This page was last modified on 1 January 2015, at 13:40.

- This page has been accessed 25,554 times.