How does the DMA work

by pcguts

The DMA is another two chips on your motherboard (usually is an Intel 8237A-5 chips) that allow you (the programmer) to offload data transfers between I/O boards. DMA actually stands for 'Direct Memory Access'.

DMA can work: memory->I/O, I/O->memory. The memory->memory transfer doesn't work. It doesn't matter because ISA DMA is slow as hell and thus is unusable. Futhermore, using DMA for zeroing out memory would massacre the contents of memory caches.

What about caches and DMA? L1 and L2 caches work absolutely transparently. When DMA writes to memory, caches autmatically load or least invalidate the data that go into the memory. When DMA reads memory, caches supply the unwritten bytes so not old but new values are tranferred to the peripheral.

There are signals DACK, DRQ, and TC. When a peripheral wants to move a byte or 2 bytes into memory (is dependent on whether 8 bit or 16 bit DMA channel is in use -- 0,1,2,3 are 8-bit, 5,6,7 are 16-bit), it issues DRQ. DMA controller chats with CPU and after some time DMA controller issues DACK. Seeing DACK, the peripheral puts it's byte on data bus, DMA controller takes it and puts it in memory. If it was the last byte/word to move, DMA controller sets up also TC during the DACK. When peripheral sees TC, it is possible it will not want any more movements,

In the other direction, everything is the same, but first the byte/word is fetched from the memory and then DACK is generated and the peripheral takes the data.

DMA controller has only 8-bit address counter inside. There is external ALS573 counter for each chip so it makes programmer see it as DMA controller had 16 bits of address counter per channel inside. There are more 8 bits of address per channel of so called page register in LS612 that unfortunately do not increment as those in ALS573. All these 24 bits can address 16777216 of distict addresses.

Recapitulation: for each channel, independently, you see 16 bits of auto-incrementing counter, and 8 bits of page register which doesn't increment.

The difference between 16-bit DMA channels and 8-bit DMA channels is that the address bits for 16-bit channels are wired one bit left to the address bus so every address is 2 times bigger. The lowest bit is 0. The highest bit of page register would fit into bit 24 which is not on ISA so that it is left unconnected. The bus control logic is wired for 16-bit channels in a manner every single DMA transfer, a 16-bit cycle is generated, so ISA device puts 16 bits onto the bus at the time. I don't know what happens if you use 16-bit DMA channel with XT peripheral. I guess it could work but only be slower.

8-bit DMA: increments by 1, cycles inside 65536 bytes, addresses 16MB, moves 8 bits a time.

16-bit DMA: increments by 2, goes only over even addresses, cycles inside 131072 bytes, addresses 16MB, moves 16 bits a time. Uses 16-bit ISA I/O cycle so it takes less ticks to make one move that the 8-bit DMA.

An example of DMA usage would be the Sound Blaster's ability to play samples in the background. The CPU sets up the sound card and the DMA. When the DMA is told to 'go', it simply shovels the data from RAM to the card. Since this is done off-CPU, the CPU can do other things while the data is being transferred.

Enough basics. Here's how you program the DMA chip.

When you want to start a DMA transfer, you need to know several things:

- Number of DMA channel you want to use
- What page to use
- The offset in the page
- The length
- How to tell you peripheral to ask for DMA

- You cannot transfer more than 64K or 128K of data in one shot, and
- You cannot cross a page boundary. If you cross it, the lower 16 or 17 bits of address will simply wrap and you only suddenly jump 65536 or 131072 bytes lower that where you expected. It will be absolutely OK and no screw up will be performed. If you will take it in account in your program you can use it.

Restriction #1 is rather easy to get around. Simply transfer the first block, and when the transfer is done, send the next block.

For those of you not familiar with pages, I'll try to explain.

Picture the first 16MB region of memory in your system. It is divided into 256 pages of 64K or 128 pages of 128K. Every page starts at a multiple of 65536 or 131072. They are numbered from 0 to 255 or from 0 to 127.

In plain English, the page is the highest 8 bits or 7 bits of the absolute 24 bit address of our memory location. The offset is the lower 16 or 17 bits of the absolute 24 bit address.

Now that we know where our data is, we need to find the length.

The DMA has a little quirk on length. The true length sent to the DMA is actually length + 1. So if you send a zero length to the DMA, it actually transfers one byte or word, whereas if you send 0xFFFF, it transfers 64K or 128K. I guess they made it this way because it would be pretty senseless to program the DMA to do nothing (a length of zero), and in doing it this way, it allowed a full 64K or 128K span of data to be transferred.

Now that you know what to send to the DMA, how do you actually start it? This enters us into the different DMA channels.

The following chart will describe each channel and it's corresponding port number:

DMA Channel	Page	Address	Count
0	87h	0h	1h
1	83h	2h	3h
2	81h	4h	5h
3	82h	6h	7h
4	8Fh	C0h	C2h
5	8Bh	C4h	C6h
6	89h	C8h	CAh
7	8Ah	CCh	CEh

DMA 4. Doesn't exist. DMA 4 is used to cascade the two 8237A chips. When first 8237A wants to DMA, it issues "HRQ" to second chip's DRQ 4. The second chip thinks DMA 4 is wanna be made so issues DRQ 4 to the first chip's HLDA. First chip makes it's own DMA 0-3, then sends to the second "OK second chip, my DMA 4 is complete" and second chip knows it's free on the bus. If this mechanism would not work, the two chips could peck each other on the BUS and the PC would screw up. :+)