ISA DMA

From OSDev Wiki

The main points about ISA DMA are

- ISA DMA is not the same thing as PCI Busmastering DMA;
- ISA DMA channels 1, 2 and 3 are available for 8 bit transfers to ISA peripherals;
- ISA DMA channels 5, 6 and 7 are available for 16 bit transfers to ISA peripherals;
- Transfers must not cross physical 64 KB boundaries and must never be bigger than 64 KB;
- Transfers must be physically contiguous, and can only target the lowest 16 MB of physical memory;
- ISA DMA is slow - theoretically 4.77 MB/second, but more like 400 KB/second due to ISA bus protocols;
- ISA DMA frees up CPU resources, but adds an extremely heavy load to the memory bus;
- Very few devices currently use ISA DMA -- only internal floppies, some embedded sound chips, some parallel ports, and some serial ports.

Notes:

- Sound Blaster and Sound Blaster PRO only support 8 bit DMA;
- Sound Blaster 16+ supports both;
- Floppy disk controllers only support 8 bit DMA and are hardwired to use DMA Channel 2.

Contents

There Is More Than One Kind of DMA on a PC

Modern PCI controllers always have their own 'Busmastering DMA', which is far better than ISA DMA. Even USB floppy drives send their DMA data using PCI Busmastering, through the PCI USB controller. PCI Busmasters can access memory with 32 bit addressing. Newer PCI cards are starting to support 64 bit addressing (although at the moment most don't). Typically PCI cards use 'scatter-gather' bus mastering, where one page is used as a directory of data pages. This almost completely overcomes the "physical memory only" limitation of all forms of DMA.

ISA DMA Background

ISA DMA (*Industry Standard Architecture Direct Memory Access*), like ISA itself, is an appendix for modern PCs. It is used by the internal floppy disk controller, ISA sound cards, ISA network cards, and parallel ports (if they support ECP mode). Whilst interrupt, keyboard and timer interface circuits have obvious and relevant uses, the ISA DMA controller and its programming interface are still well and truly stuck in the 1970s where they were first designed.

The idea behind DMA is that you can set up a 'channel' with an address pointing into memory and the length of the data to be transferred. Once set up, the CPU can tell the peripheral owning the channel to do whatever it is supposed to do (e.g. read a sector). Then the CPU can go do something else. When the memory bus isn't being used by the CPU, the DMA chip takes over and transfers data between the peripheral and memory without involving the CPU. When the transfer is complete (e.g. an entire sector has been sent to the floppy drive) the DMA chip then signals that it is finished. The DMA chip can even signal if it has run out of data, allowing the system to locate the next block of data to transfer on the same DMA transaction. DMA can improve the speed of a system quite a bit and was borrowed by Intel (who designed the DMA controller chip) from the old 1960s mainframes which had DMA channels for all devices (CPUs weren't all on a single chip and very slow back then).

Of course all good ideas can have downsides and while Intel can't really be blamed for what is about to be described, IBM certainly can.

In the beginning there was a PC, but the PC was slow. IBM looked down from the heavens and said "Slap on a DMA controller -- that should speed it up." IBM's heart was in the right place; its collective brains were elsewhere as the DMA controller never met the needs of the system. The PC/AT standard contains 2 Intel 8237A DMA chips, connected as Master/Slave. The second chip is Master, and its first line (Channel 4) is used by the first chip, which is Slave. (This is unlike the interrupt controller, where the first chip is Master.) The 8237A was designed for the old 8080 8-bit processor and this is probably the main reason for so many DMA problems. The 8088 and 8086 processors chosen by IBM for its PC were too advanced for the DMA controller.

Previously it was mentioned that a DMA controller is able to signal completion and even ask for more information. Unfortunately this would make expansion slots too big, so IBM left all of the connections to the DMA chips off. The only time you know when a transfer is complete is for a peripheral to signal an interrupt. This implies that all peripherals using an ISA DMA channel are limited to no more than 64 KB transfers for fear of upsetting the DMA controller.

Even with the PC/AT, IBM began bypassing the ISA DMA used in the PC/XT and used ATA PIO Mode for the hard disk. This was because of the 64 KB limitations outlined above and the fact that the 286 processor could perform 16 bit transactions at 6 MHz. Even the ISA bus could run at a speed of up to 12 MHz, far faster than the 4.77 MHz the DMA controller was running at.

Expansion card designers were also upset with DMA's lack of capabilities, noticeably 'Hard-Card' hard disk expansion card manufacturers who depended on the speed of data transfer.

To get around the limitations of the 'on board' DMA controller, expansion card manufacturers began to put their own DMA controllers on their expansion cards. They functioned exactly the same way as the 'on board' DMA, 'stealing memory bus cycles' when the processor wasn't looking and thus improving the performance of the system as a whole. These "ISA Bus Masters" are still usually limited to the lower 16 MiB of memory, but do not have the 4.77 MHz issue. This trend continued through the creation of the PCI bus, which eventually entirely replaced the ISA bus in PCs.

Technical Details

Each 8237 DMA chip has 4 DMA channels. DMA0-DMA3 on the first chip and DMA4-DMA7 on the second. The first DMA controller is wired up for 8 bit transfers, while the second is wired up for 16 bit transfers. On some tutorials or other wiki articles, you will sometimes see the **second** DMA chip (channels 4 to 7) labeled as the "**master**" controller, and the first (channels 0 to 3) called the "slave". This is highly confusing, and these terms will not be used again, here.

DMA Channel 0 is unavailable as it was used for a short time for DRAM memory refresh, and remains reserved because of this (even though modern computers don't use it). DMA channel 4 cannot be used for peripherals because it is used for cascading the other DMA controller.

The internal address registers of the DMA controller are only 16 bits. In order to extend this, IBM added one *external* "page register" byte per channel, allowing access to 16 MB of memory (24 bits total). If a DMA transfer crosses a 64 KB boundary, the internal address register wraps around to zero, and the external page register is **not** incremented. The DMA controller will happily continue the transfer with whatever data it finds at the new address.

ISA-based DMA controllers are specified to run at 4.77 MHz. **No exceptions**. If the "front-side" (memory) bus of a system runs at 133 MHz, it will be artificially slowed down to 4.77 MHz when transferring each ISA DMA byte/word. This includes EISA and PS/2 32-bit controllers, even though these controllers have an extra page register (which allows a 4 GiB addressing space) and the ability to do 32 bit transfers. (These DMA controllers exist only on EISA and MCA systems, which are now obsolete and are not further described here.)

Programming Details

16 bit issues

The 16 bit channels (5, 6, and 7) have a special addressing scheme to handle the way they increment addresses. The internal registers increment by 1, but the memory address needs to be incremented by 2 between each access. The solution is that the "start address" that the CPU stores in the DMA controller needs to be shifted right by 1 bit. On each memory access, this internal 16 bit address value is incremented by one, then that value is taken and shifted one bit to the left (clearing the lower bit) before being used as an address. The external "page register" addressing byte is then appended in the normal way. It is important to note that the upper bit of the internal address is **lost** when it is shifted left -- it is not ORed into the page register byte. This prevents DMA transfers bigger than 64 KB from working even though they should be technically possible for 16 bit channels, because the address register will effectively wrap around to zero without incrementing the page register byte.

Physical Memory vs. Paging

Paged memory mapping is exclusively controlled by the CPU. The whole point of DMA is to bypass the CPU. Therefore, no DMA can ever access any virtual memory addresses. All DMA is always done on physical memory addresses only. ISA DMA has a 16 MB physical address limit.

Since DMA runs independently of the CPU, it is important that an OS allocates a block of contiguous physical memory for the DMA transfer in a way that prevents that memory from being used for any other purpose, or swapped out, until the DMA transfer is complete.

Note: VM86 mode does not use *physical* addressing. The memory addresses are *fake*. In VM86 mode the OS must emulate any DMA transactions on behalf of an application.

Buffer Size

A typical 1.44 MB floppy disk can easily transfer 36 sectors of data in a single transfer. This is only 18 KB. The biggest internal floppy with worst-case formatting may be able to transfer 84 sectors at once. This is still only 42 KB. A Soundblaster card may run best with a 64 KB buffer. It is never necessary to try to double-buffer ISA DMA transfers, because they are so slow anyway. There are at most 6 usable DMA channels, and it is not necessary to allocate a full 64 KB to each of them. Putting all of this together, any OS should be easily able to allocate all the ISA DMA physical memory that it needs from a 256 KB pool, or even only half of that.

The Flip-Flop

Many devices on a PC (e.g. ATA disk drives) use 8 bit IO Ports to receive 16 bit values. This is done using a flip-flop. The device expects the low byte first. As soon as it receives a byte, the flip-flop changes state and then the device expects the high byte. When the high byte is received, the flip-flop changes state again, and the device expects a new low byte. Usually, each 16 bit 'register' will have its own flip-flop, but the ISA DMA controller has a problem regarding this.

On a 8237 chip, there is **only one** flip-flop. And there are eight of the 16 bit registers. And there can be up to three device drivers all competing for the use of that one flip-flop simultaneously.

This creates two serious problems. One is "contention issues". The other is that it is difficult to be sure what state the flip-flop is currently in. The standard solution for dealing with the flip-flop state issue is to reset the flip-flop to "low byte" state every single time you want to use it, just so you can be certain it is in the proper state before sending bytes. There are only two solutions to "contention": either use a lock, or allow only one ISA DMA driver, so that contention is impossible.

Masking DRQ

Setting up a DMA transfer always requires setting up both "ends" of the transfer. That is, whichever peripheral owns the DMA channel needs to be told to transfer a block of data via DMA. And the DMA controller needs to be told the memory address, the transfer length, perhaps a transfer "mode", and a transfer direction (read or write). So, one of these two things always needs to be done *first* -- and it is usually the peripheral that has a long latency time before being ready to transfer the first byte. If you set up the peripheral first, however, its first DMA request signal (often called DRQ) may arrive while you are in the middle of setting up the DMA controller.

The answer is to mask DRQ for a particular channel while you are initializing the DMA controller. There are three ways of temporarily disabling channels described below.

Transfer Length

The value that gets stored into each Count Register is always the transfer length (either bytes or words) **minus 1**. If you forget to subtract the one, you will get an error on your transfer.

Interrupt on Completion

As implemented in a PC, the DMA controller can not send interrupts. Hopefully, whichever peripheral "owns" each DMA channel will send the CPU an interrupt when a transfer completes. However, some peripherals may **not** send an interrupt if a transfer fails with an error. As always, timeouts are important.

The Registers

The master and slave DMA controllers are very similar, so (to save space) both of them have been combined into the following table. Please try not to let this confuse you. Note: for Address and Count Registers on channels 5 to 7, see 16 bit issues above.

Each 8237A has 18 registers, addressed via the I/O Port bus:

Channels 0-3	**Channels 4-7**			
IO Port	**IO Port**	**Size**	**Read or Write**	**Function**
0x00	0xC0	Word	W	Start Address Register channel 0/4 (unusable)
0x01	0xC2	Word	W	Count Register channel 0/4 (unusable)
0x02	0xC4	Word	W	Start Address Register channel 1/5
0x03	0xC6	Word	W	Count Register channel 1/5
0x04	0xC8	Word	W	Start Address Register channel 2/6
0x05	0xCA	Word	W	Count Register channel 2/6
0x06	0xCC	Word	W	Start Address Register channel 3/7
0x07	0xCE	Word	W	Count Register channel 3/7
0x08	0xD0	Byte	R	Status Register
0x08	0xD0	Byte	W	Command Register
0x09	0xD2	Byte	W	Request Register
0x0A	0xD4	Byte	W	Single Channel Mask Register
0x0B	0xD6	Byte	W	Mode Register
0x0C	0xD8	Byte	W	Flip-Flop Reset Register
0x0D	0xDA	Byte	R	Intermediate Register
0x0D	0xDA	Byte	W	Master Reset Register
0x0E	0xDC	Byte	W	Mask Reset Register

0x0F	0xDE	Byte	RW	MultiChannel Mask Register (reading is undocumented, but it works!)

Each Channel also has an external R/W Page Address Register that contains the upper 8 bits of the 24 bit transfer memory address:

0x87	Channel 0 Page Address Register (unusable)
0x83	Channel 1 Page Address Register
0x81	Channel 2 Page Address Register
0x82	Channel 3 Page Address Register
0x8F	Channel 4 Page Address Register (unusable)
0x8B	Channel 5 Page Address Register
0x89	Channel 6 Page Address Register
0x8A	Channel 7 Page Address Register

Useful Registers

Single Channel Mask Registers 0x0A and 0xD4 (Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
					MASK_ON	SEL 1	SEL 0

These registers are used to mask (or unmask) DRQ for a single channel only, on either the master or slave DMA chip. That is, if you do not want to figure out the mask states of all the other channels, you can mask/unmask DRQ for one channel at a time. Use the SEL 0 and 1 bits to select the channel, and the MASK_ON bit to set or clear masking for it. Note that masking DMA channel 4 will mask 7, 6, 5 and 4 due to cascading.

MultiChannel Mask Registers 0x0F and 0xDE (Read and Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
				MASK3	MASK2	MASK1	MASK0

Setting the appropriate bits to 0 or 1 allows you to unmask or mask (respectively) DRQ for those channels. Using this register means that your driver needs to know the desired mask states of *all* the channels at that moment. There are several ways to do this, but one is simply to read this register, first. Note that masking DMA channel 4 will mask 7, 6, 5 and 4 due to cascading.

DMA Mode Registers 0x0B and 0xD6 (Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MOD1	MOD0	DOWN	AUTO	TRA1	TRA0	SEL1	SEL0

Setting this register is a little tricky as it depends highly on the peripheral you are programming the DMA controller for. However, the driver for the peripheral is the entity that needs to set this register, and it should know what mode the peripheral needs.

- **SEL0** and **SEL1** select the channel you want to change;
- **TRA0** and **TRA1** selects the transfer type;
  - 0b00 runs a self test of the controller;
  - 0b01 Peripheral is writing to memory;
  - 0b10 Peripheral is reading from memory;
  - 0b11 invalid.
- **AUTO**: When this bit is set, after a transfer has completed the channel resets itself to the address and count values you programmed into it. This is great for floppy transfers. Read in a track - the values set themselves up for reading again immediately. For writing you'd only need to alter the transfer mode - not the addresses. Some expansion cards do not support auto-init DMA such as Sound Blaster 1.x. These devices will crash if used with auto-init DMA. Sound Blaster 2.0 and later do support auto-init DMA.
- **DOWN**: Reverses the memory order of the data, when set. Memory is accessed from high addresses down to low addresses (the address is decremented between each transfer).
- **MOD0** and **MOD1**: This is where some problems can arise based on the peripheral the DMA controller is attached to. The DMA controller has several modes:
  - 0b00 = Transfer on Demand;
  - 0b01 = Single DMA Transfer;
  - 0b10 = Block DMA Transfer;
  - 0b11 = Cascade Mode (use to cascade another DMA controller).

Single transfer mode is good for peripherals than cannot cache a lot of data at once. Non-82077AA Floppy controllers, Sound Blaster, and Sound Blaster Pro should use Single Transfer DMA Mode.

Block transfer mode is good for peripherals that can buffer entire blocks of information. An example of this is a hard disk controller board.

Demand transfer mode is good for peripherals that start and stop intermittently such as a tape drive. The drive can read a whole load of information for as long as it can and the suspend the transfer to move to another section of the tape. Newer floppy controllers also work well with demand transfer because they have FIFO buffers to store information being read and written (but you need to set up the FIFO properly). The peripheral controls the flow, and as the information flow is uninterrupted, performance can be gained. CPUs in these later computers generally have caches and can continue working uninterrupted during a demand DMA transfer. Older computers will slow down as their CPUs wait for the memory bus to become available.

Flip-Flop Reset Registers 0x0C and 0xD8 (Write)
Master Reset Registers 0x0D and 0xDA (Write)
Mask Reset Registers 0x0E and 0xDC (Write)

Send any value to the Reset registers to activate them. Master Reset sets Flip-Flop low, clears Status, and sets all Mask bits ON. Mask Reset sets all Mask bits OFF.

The following statement from the previous wiki article needs to be verified on real hardware, because it is likely to be wrong: "The Reset Flip-Flop command must be sent *before any* 16 bit transaction. The flip-flop *does not reset* after the DMA controller has received the second byte."

The Other Registers

Status Registers 0x08 and 0xD0 (Read)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
REQ3	REQ2	REQ1	REQ0	TC3	TC2	TC1	TC0

- REQ3-0: When set: DMA Request Pending.
- TC3-0: When set: Transfer Complete.
- Reading this register will clear the TC bits.

This register isn't very important in light of the fact that the 8237 can't send an IRQ to tell you that it has finished. Usually there is no need to poll this register as the peripheral (at the other end of the DMA) will send an interrupt when a transaction has completed.

Command Registers 0x08 and 0xD0 (Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DACKP	DRQP	EXTW	PRIO	COMP	COND	ADHE	MMT

This register really shows how incompatible the 8237 is with the PC hardware.

- Let's start with EXTW and COMP. These increase the speed of DMA transfer by 25% be removing one of the clock cycles. Does it work? No.
- PRIO. When zeroed, this allows DMA priorities to be rotated allowing freedom and liberty for all peripherals that share the data bus. Does it work? No.
- MMT and ADHE. Did you know that the IBM PC could do memory to memory transfers since 1981? That's right, hardware sprites, hardware frame buffering from one location to another. Does it work? No.
- COND. Hooray the only bit in the control register that does something useful. Setting this bit disables the DMA controller. This is one way to set up multiple DMA channels without masking each and every channel.

Request Registers 0x09 and 0xD2 (Write)

Used for memory to memory transfers and setting up priority rotation -- absolutely useless.

"Intermediate" Registers 0x0D and 0xDA (Read)

Never implemented on PCs. Useless.

Examples

Floppy Disk DMA Initialization

You need only implement 1 to 3 tiny routines to perform a DMA transfer. This example is the Floppy Drive controller (probably the most common followed by SoundBlaster).

Note: the following code is not optimal, because there is an OUT to the same IO port twice (in two places). This causes an extra delay on the IO Port bus. Real code should separate the two "out 0x4" and "out 0x5" calls with an "out" to some other port.

```
initialize_floppy_DMA:
; set DMA channel 2 to transfer data from 0x1000 - 0x33ff in memory
; paging must map this _physical_ memory elsewhere and _pin_ it from pagi
; set the counter to 0x23ff, the length of a track on a 1.44 MiB floppy -
; transfer length = counter + 1
    out 0x0a, 0x06      ; mask DMA channel 2 and 0 (assuming 0 is already
    out 0x0c, 0xFF      ; reset the master flip-flop
    out 0x04, 0         ; address to 0 (low byte)
    out 0x04, 0x10      ; address to 0x10 (high byte)
    out 0x0c, 0xFF      ; reset the master flip-flop (again!!!)
    out 0x05, 0xFF      ; count to 0x23ff (low byte)
    out 0x05, 0x23      ; count to 0x23ff (high byte),
    out 0x81, 0         ; external page register to 0 for total address c
    out 0x0a, 0x02      ; unmask DMA channel 2
    ret
```

Once you have set up your start address and transfer length you do not need to touch it again, if you are using autoinit. Once reading or writing is selected, you don't need to change that, either. To *change* selecting reading or writing you use the mode register.

```
prepare_for_floppy_DMA_write:
    out 0x0a, 0x06      ; mask DMA channel 2 and 0 (assuming 0 is already
    out 0x0b, 0x5A      ; 01011010
                        ; single transfer, address increment, autoinit, w
    out 0x0a, 0x02      ; unmask DMA channel 2
    ret

prepare_for_floppy_DMA_read:
    out 0x0a, 0x06      ; mask DMA channel 2 and 0 (assuming 0 is already
    out 0x0b, 0x56      ; 01010110
                        ; single transfer, address increment, autoinit, r
    out 0x0a, 0x02      ; unmask DMA channel 2
    ret
```

Some hardware, as well as VirtualPC do not support autoinit. You may want to set the Mode registers to 0x4A and 0x46 in the above routines, instead.

The above routines use single transfer mode for compatibility, but during the initialization of your floppy driver if you detect an "advanced" floppy controller (using the Version command), "demand transfer" should be used to reduce overhead.

References

Articles

- Floppy Disk Controller

External Links

- Intel 8237A datasheet (http://www.intel-assembler.it/PORTALE/4/231466_8237A_DMA.pdf)
- http://bos.asmhackers.net/docs/dma/docs/

Retrieved from "http://wiki.osdev.org/index.php?title=ISA_DMA&oldid=14073"
Category: Storage

- This page was last modified on 2 November 2012, at 21:44.
- This page has been accessed 71,400 times.