Direct Memory Access (DMA)

Direct Memory Access is a method of transferring data between peripherals and memory without using the CPU. After this lecture you should be able to: identify the advantages and disadvantages of using DMA and programmed I/O, select the most appropriate method for a particular application, and describe the sequence of events that takes place during a DMA transfer on the IBM PC.

Programmed I/O

Programmed I/O (PIO) refers to using input and output (or move) instructions to transfer data between memory and the registers on a peripheral interface.

The advantage of PIO is that it is simple to implement. In many cases the CPU will be fast enough to transfer data as fast as the peripherals (e.g. a hard disks) can supply or accept it. Often the only additional hardware required is a circuit to request CPU wait states in order to slow down or synchronize the CPU.

The disadvantage of PIO is that the CPU is tied up for the duration of the transfer while doing a relatively simple task.

Direct Memory Access (DMA)

In some cases the CPU may not be fast enough to keep up with the peripheral or it may be desirable to allow the CPU to do other useful work while the I/O is in progress.

In this case a special-purpose processor called a DMA controller (DMAC) can be used to transfer data between memory and I/O devices. The DMA controller periodically takes over control of the system bus from the CPU, and, like the CPU, generates address, data and control signals to transfer data between memory and I/O devices.

The DMA controller is a special-purpose device designed explicitly for this data transfer function. It can perform all the operations required for data transfer (increment the memory address, decrement the count, input, write, and test for operation complete) in *one* bus cycle. This speeds up the transfer of data and reduces the number of bus cycles required to transfer a given amount of data.

DMA controllers can be set up to take over the bus for each byte of data to be transferred and then return control to the CPU ("cycle stealing") or they can operate in burst mode in which a block of data is transferred before returning bus control to the CPU.

DMA controllers can transfer data in a two-step process by reading a value from one port or address in one bus cycle and writing that value to another port or address in a second bus cycle. It is also possible for the DMA controller to carry out read and write operations simultaneously. In this case the data is transferred directly between the I/O device and memory in the same bus cycle. This is the mode of operation used in the IBM PC.

DMA versus PIO

It makes sense to use DMA when it is necessary to transfer data faster than the CPU can keep up with it or when it is desirable to reduce the CPU or bus bandwidth overhead used by I/O operations. The main disadvantages of DMA are the additional cost of the hardware and the added complexity of the software.

Note that the choice of DMA versus PIO concerns *how* the data is transferred. The choice of using polling or interrupts concerns how the CPU determines *when* the data is ready to be transferred.

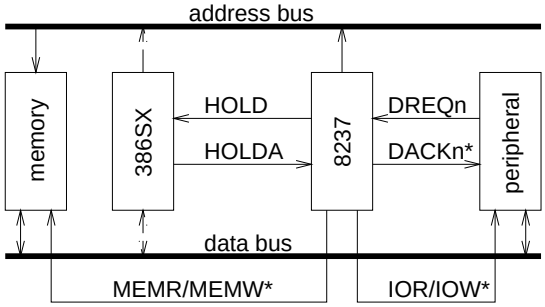
DMA and interrupts are features common in larger systems where it is desirable to minimize the CPU overhead required for I/O. It is common for a peripheral to issue an interrupt when data is available, then for the CPU to set up the DMA controller to do the actual data transfer and finally for the DMA controller to issue an interrupt when the transfer is complete. In between these events the CPU can continue with other tasks.

DMA on the IBM PC

The IBM PC and later compatible machines use the 8237 DMA controller. The controller is used to transfer data between I/O ports and memory. It supports four prioritized DMA "channels." Each channel contains independent address and count registers and peripheral control lines. Requests for DMA transfers are prioritized although only one transfer can be "active" at a time (i.e. a burst mode transfer must complete before another DMA request is serviced).

The DMA controller is initialized through a number of on-chip control registers whose details we will not cover.

The following diagram shows how the 8237 DMA controller interfaces to the CPU, the I/O devices and the memory:



The DMA controller uses hold request (HOLD) and hold acknowledge (HOLDA) signals to ask the CPU to stop driving the address, data and control buses so that the DMA controller can drive them to carry out a transfer.

The DMA controller interfaces to peripherals through 4 pairs of DMA request (DREQ0 to DREQ3) and DMA acknowledge (DACK0* to DACK3*) lines available to peripherals on the system bus. Once the DMA controller has control of the bus, it can also interface to memory and I/O peripherals using the address bus, data bus and the memory/I/O read/write control signals (MEMR*, MEMW*, IOR*, and IOW*).

8237 DMA Operation

Once the appropriate channel of the DMA controller has been programmed with the memory starting address and transfer count, the corresponding peripheral is set up to start reading or writing data. For example, a command could be issued to a disk controller to read a sector or to a sound card to start requesting audio samples.

The following sequence of steps take place for a DMA transfer in cycle-stealing mode:

- each time the peripheral is able to transfer a byte it asserts its DMA request line to the DMA controller
- the DMA controller asserts the CPU's hold request pin
- when the CPU control circuitry is able to suspend execution (at the end of an instruction or by inserting wait states in T2) it asserts the hold acknowledge (HOLDA) signal to the DMA controller and floats the address, data and control bus signals
- the DMA controller then puts the memory address on the address bus, asserts either MEMR* plus IOW* or MEMW* plus IOR* on the control bus and asserts the appropriate DMA acknowledge line to the peripheral
- the peripheral responds to the DMA acknowledge signal by reading or writing it's data to the data bus
- at the same time the memory responds to the MEMR*/MEMW* control signal which causes the data to be read/written directly from/to memory
- at the end of the bus cycle the DMA controller then negates hold request line and the CPU can continue to execute until the next DMA request

Exercise: Draw a timing diagram to illustrate the behaviour of the signals involved in a DMA cycle.

Unfortunately, the 8237 DMA controller's address registers are only 16 bit wide and so it's not possible to do DMA transfers accross 64k boundaries. Also, the DMA controller does not have use to the logical-to-physical address translation used on 386 and later CPUs so it's impossible for user-space programs to use DMA under operating systems that use virtual memory.