DMA, System Buses and Peripheral Buses

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.
System buses are the buses used to interface the CPU with memory and peripherals on separate PC cards. The ISA and PCI buses are used as examples. You should be able to state the most important specifications of both buses.
Serial interfaces are typically used to connect computer systems and low-speed external peripherals such as modems and printers. Serial interfaces reduce the cost of interconnecting devices because the bits in a byte are time-multiplexed onto a single wire and connector.
After this lecture you should be able to describe the operation and format of data and handshaking signals on an RS-232 interface.
Parallel ports are also used to interface the CPU to I/O devices. Two common parallel port standards, the "Centronics" parallel printer port and the SCSI interface are described.
After this lecture you should be able to design simple input and output I/O ports using address decoders, registers and tri-state buffers.

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.

For example, the following section of code reads 512 bytes from an input port 380H and writes the data to a buffer in memory:

```
    mov    bx,buf ; destination address
    mov    cx,512 ; count
    mov    dx,380H ; source port
loop:
    in     al,dx ; get byte from i/o port
    mov    [bx],al ; store in buffer
    inc    bx ; next memory location
    dec    cx ; decrement bytes left
    jnz loop
```

The main advantage of PIO is that it is simple and inexpensive to implement. In many cases the CPU will be fast enough to transfer data as fast as the peripherals (e.g. a hard disk) can supply or accept it. Often the only additional hardware required is a status port that the CPU can poll or a circuit to generate interrupt requests.

The main disadvantage of PIO is the need to execute many instructions for each byte transferred. This limits the maximum possible data transfer rate. In a multi-tasking system this CPU overhead and also reduces the CPU time available for other tasks.

Exercise 80: Using the example above, approximately how many instructions need to be executed to transfer each byte? Approximately how many bus cycles will be required?

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 independently transfer data between memory and I/O devices. The DMA controller periodically takes over control of the bus from the CPU, and 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. Thus the bus (and the CPU) is only tied up for one bus cycle per byte transferred.

Cycle Stealing and Burst Modes

DMA controllers can operate in a *cycle stealing* mode in which they take over the bus for each byte of

data to be transferred and then return control to the CPU. They can also operate in *burst mode* in which a block of data is transferred before returning bus control to the CPU. The choice depends on the speed at which data is arriving relative to the bus bandwidth and whether applications will allow the CPU to be cut off from the bus for the duration of one transfer.

One- and Two-Step DMA Transfers

In most cases the DMAC is part of a peripheral device. The DMAC takes over the bus and transfers one value between the device and the memory in one bus cycle.

A DMA controller can also be designed to 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. This allows the DMAC to transfer data between memory and registers on other devices.

On the IBM PC the DMA controller uses special DMA control lines on the system bus transfer data directly between the I/O device and memory in the same bus cycle.

DMA versus PIO

It makes sense to use DMA when it is necessary to transfer data faster than the CPU can transfer it or when it is desirable to reduce the CPU or bus bandwidth overhead used by I/O operations. The disadvantage of using 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 polling or interrupts determines how the CPU determines *when* the data is ready to be transferred. All four combinations of polling/interrupts and PIO/DMA are possible.

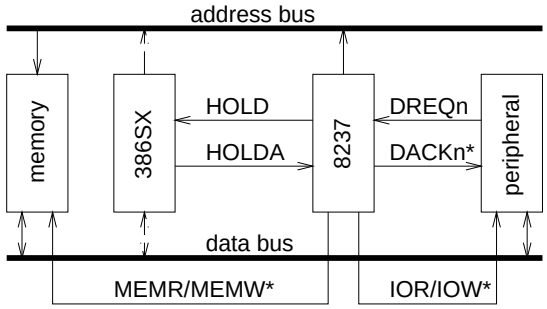
DMA and interrupts are features common in larger systems where it is desirable to minimize the CPU overhead required for I/O. It is common for a peripheral to issue an interrupt when data is available, then for the CPU to set up the DMA controller to do the actual data transfer and finally for the DMA 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 the CPU's 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 use them to carry out a data 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*).

The peripheral interface (e.g. floppy disk controller) must be designed to assert a DREQ line when a byte can be read or written from the peripheral. It must also be designed to read/write from/to the I/O device when the corresponding DACK signal is asserted by the DMA controller.

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 during a DMA transfer:

- each time the peripheral is ready 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 (typically at the end of an instruction) it asserts the hold acknowledge (HOLDA) signal to the DMA controller and floats ("tri-states") 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 continues to execute until the next DMA request

Exercise 81: Is this an example of a one- or two-step transfer? Is this a cycle-stealing or burst mode transfer?

Exercise 82: Draw a timing diagram to illustrate the behaviour of the signals involved in a DMA cycle.

Limitations of the IBM PC's DMA

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 access 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.

Exercise 83: Most peripherals' DMA operations can be disabled. How many DMA-using peripherals can be installed in a PC ? How many DMA transfers can be taking place at the same time?

System Buses

To increase their flexibility, most general-purpose microcomputers include a system bus that allows printed circuit boards (PCBs) containing memory or I/O devices to be connected to the CPU. This allows microcomputer systems to be customized for different applications. The use of a standard bus also allows manufacturers to produce peripherals that will work properly with other manufacturers' computer systems.

The system bus consists of a number of parallel conductors on a "backplane" or "motherboard." There are a number of connectors or "slots" into which other PCBs containing memory and I/O devices can be plugged in.

In most cases the system bus is very similar to the processor bus. In fact, the simplest system buses simply consists of a connector that allows access to all of the pins on the CPU chip. Like the processor bus, the system bus can be subdivided into an address bus, data bus, control bus and power connectors.

Some microcomputer designs place the CPU and some auxiliary circuits on a PCB (the "motherboard") along with system bus connectors. Examples of this approach include most popular IBM PC-compatible systems. Other designs use a simpler passive "backplane" type of bus and place the CPU on its own PCB. This approach is used by most VME-based microcomputers. The latter approach has the advantage that the CPU card can be replaced if required. Because the bandwidth required for the RAM interface is much higher than that for most

other peripherals, most modern systems use a system bus only for I/O devices and either use a proprietary interface for RAM or simply place all of the RAM memory on the same PCB as the CPU.

A bus can be described by its mechanical (size, types of connectors, etc), electrical (voltage, clock rates, etc), and protocol (read/write cycles, master/slave operation, interrupt acknowledgement, etc) characteristics.

We will look briefly at two examples of system buses. The ISA (Industrial Standard Architecture) bus is commonly used in IBM-PC compatibles and is one of the most widely-used system buses. The PC-104 bus (the one used in the lab computers) uses the same signals as the ISA bus but has a different connector. The PCI (Peripheral Component Interconnect) bus is a flexible high-performance peripheral bus that can efficiently interconnect peripherals and processors of widely different speeds.

Mechanical Characteristics

Low-cost consumer-grade buses use card-edge connectors to minimize the cost of the peripheral. The plug-in card has contact pads along the edges of the PCB. The motherboard has connectors on the motherboard that contact these pads. Typical examples include the ISA and PCI busses.

Some buses use connectors on both the motherboard and the card. This adds to the cost of the card but makes for more reliable contacts. A typical example is the 3-row by 32-pin (96-pin) "DIN" connector used by the VME bus.

Electrical Characteristics

The electrical characteristics include the voltage and current specifications, types of bi-directional signals and impedance matching.

Bus Drivers and Receivers

Bus signals such as clocks or data lines often drive several chips on a peripheral card. If the load presented by the card exceeds the bus specifications then buffer chips ("bus receivers") must be used. Similarly, if the output driving capacity of a chip on the peripheral card is not enough to drive the load presented by the bus, a "bus driver" must be used to buffer the signal.

Bidirectional Buses

Some bus signals can be driven by multiple cards. For example, the data bus will be driven by all peripheral devices as well as the CPU. Two common methods are open-collector and tri-state outputs. As always, care must be taken in the design to prevent multiple simultaneous outputs on the same bus line (bus conflicts).

Open-Collector (OC) Logic

OC outputs can only sink current, not source it. A pull-up resistor is used to pull that particular bus line to V_{cc}. Any device connected to that line can pull the signal low. The choice of pull-up resistance is also a compromise between rise time and power consumption. The resistance of the pull-up resistor must low enough to obtain a sufficiently fast rise time and yet the resistance must be high enough to limit the current through it to a value that will not damage the OC outputs.

Tri-State Logic

Tri-state logic outputs can be set to a high output, a low output or a high-impedance mode. They are used to drive bus signals that might be driven by more than one device.

For example, on a system that has multiple bus masters the address data bus must be driven by tri-state outputs and the data bus driven by tri-state bus transceivers.

Dynamic Contention

It should be remembered that buffers (receivers, drivers and transceivers) take a finite time to switch modes. A detailed timing analysis is necessary to ensure that two outputs will not be enabled simultaneously. This analysis will needs to take into account the CPU timing as well propagation delays for the circuits that control the buffer directions.

Impedance and Termination

To analyze the performance of the bus for the short rise times required in high-speed buses the bus must be treated as a transmission line.

The pulse generated by a bus driver will propagate down the bus. If the bus is not terminated on both ends by the characteristic impedance of the bus a portion of the signal will be reflected. The reflections will propagate back along the bus and appear as ringing and noise. In addition, there will be reflections from each point where a card is connected to the bus.

For the above reasons it is important that high-speed buses be terminated in the characteristic impedance of the bus. The typical characteristic impedance is on the order of 100 ohms. "Active" terminations connect a resistor of about this value to a low-impedance voltage source at about half of V_{cc} to minimize the current drawn by the termination.

ISA Bus

The bus used for expansion cards by the original IBM PC was designed to support expansion memory and peripherals such as video displays and parallel printer ports. It had the same 8-bit data bus and 20-bit address space as the Intel 8088 processor in the PC. The subsequent PC/AT (ISA) bus used a second card-edge connector to extended the address space to 24 bits and the data bus to 16 bits.

The PC cards are about 10 cm high, up to 33 cm long and use one (PC) or two (AT) card-edge connectors.

Intel chips have a separate I/O address space and the ISA bus includes MEMR* (memory read) and MEMW* (memory write) and IOR* (I/O read) and IOW* (I/O write) strobes.

The ISA bus is synchronous. The CPU performs fixed-length read and write cycles although a WAIT signal is available on the bus so that slow peripherals can request wait states.

The PC bus has 6 active-low interrupt request lines (IRQ2* to IRQ7*). The PC motherboard has a programmable interrupt controller chip (intel 8259) that arbitrates different levels of interrupt requests and generates an interrupt number in response to the processor's interrupt acknowledge cycle.

The bus also has 3 DMA request (DRQ*) and acknowledge (DACK) lines that can be used by peripherals with DMA capabilities.

The PC/AT (ISA) bus (but not the PC/XT bus) allows for a limited form of external bus mastering.

PCI Bus

The PCI bus is a higher-performance peripheral (system) bus. The PCI bus maximizes CPU performance by decoupling the CPU and peripheral buses as much as possible. This decoupling also makes the bus relatively independent of the host CPU architecture.

The PCI bus is connected to the CPU bus by means of a *bridge*. This is an interface circuit that multiplexes address and data signals from the CPU, buffers (caches) data transferred between the CPU and the cards on the PCI bus, and optimizes access to sequential memory locations.

The PCI bus has a multiplexed 32-bit address/data bus and runs at 33 MHz. A read cycle takes 3 clock cycles and a write cycle takes 2 clock cycles. Thus the bus bandwidth is 44 or 66 MB/s.

If consecutive memory locations are being accessed the PCI bus allows sequential read or write cycles to proceed without explicitly putting the address on the bus. In this case one 32-bit word can be transferred in each clock cycle for a bus bandwidth of 132 MB/s.

The buffering provided by the PCI bridge allows the CPU to write data to PCI peripherals without incurring wait states. The CPU can continue executing at full speed while the bridge takes care of the handshaking with the peripheral. The PCI bridge can also pre-fetch data to improve performance on reads.

It is also possible to use a second bridge between the PCI bus and another type of bus, for example, an ISA or VME bus. This allows slow peripherals to be efficiently interfaced to the CPU because this second bridge takes care of the details of interfacing the PCI bus and the slower bus.

The basic PCI bus signals include:

Signal	Purpose
AD0 to AD31	multiplexed address and data
FRAME*	indicates start of transfer
C/BE0 to C/BE3	type of cycle and byte enables
IRDY*, TRDY*	ready (to generate wait states)
CLK	clock
REQ*/GNT*	bus request/grant

Unlike the ISA bus, the PCI bus allows DMA by allowing different cards to acquire control over the bus. Each card has bus REQuest* and bus GraNT* signals that are used to communicate with a bus arbitration circuit.

There is also a 64-bit extension to the PCI bus that increases the maximum bus bandwidth to $33 \times 8 =$ 266 MB/s.

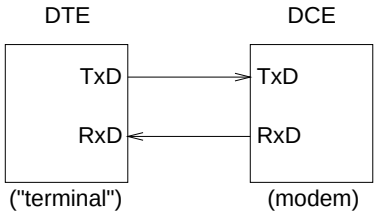
RS-232 Interface

In addition to system buses that allow PC cards to be connected to a CPU, there are also peripheral "buses" that allow peripheral devices to be connected to a computer.

The most widely used peripheral interface is the "RS-232" serial interface. This interface is available on most general-purpose microcomputers.

DTE and DCE

The serial interface was originally designed to connect modems (Data Communications Equipment - DCE) to computer terminals (Data Terminal Equipment - DTE). In its simplest form the interface has two signal lines, Transmit Data (TxD or TD) and Receive Data (RxD or RD), and a ground reference. The TxD signal is an output on a DTE device and an input on a DCE device. Similarly, RxD is an output on a DCE device and input on a DTE device.

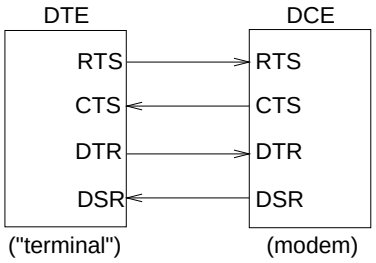


Exercise 84: Is the "Transmit Data" (TxD) signal an input or an output? How about "Receive Data" (RxD)? Is a computer a 'modem' or a 'terminal'?

The standard RS-232 connector is a 25-pin D-style connector (a "DB-25"). Pin 2 is TxD, pin 3 is RxD and pin 7 is signal ground. Two serial devices are connected pin-to-pin (RxD is connected to RxD and TxD is connected to TxD). This means that RxD must be an input on one device and an output on the other device. Thus the terms RxD and TxD *do not* say whether a pin is an input or output but are instead names for pins on the connector. Typically DTE connectors are male and DCE connectors are female.

In addition to the two data lines, most RS-232 devices implement additional handshaking pins. Of these, the most useful are called RTS (Request To Send) and CTS (Clear To Send). The CTS pin is a DCE output and is used by the DCE to indicate that it can accept data on the TxD line. The RTS line is an output on a DTE and is used to indicate that the DTE wants to send (RTS was originally used to control half-duplex modems – these are rarely seen today).

Since these signals are used to control the flow of data from the DTE (and optionally from the DCE) these pins are called [hardware] "flow control" signals.



The second set of control signals are DTR (Data Terminal Ready) and DSR (Data Set Ready). These signals indicate that the DTE and DCE devices respectively are connected and operational (typically, simply that the power is turned on). Some modems can use DTR to force a reset and DSR as a replacement for CD (see below).

In the original RS-232 specification there was no provision for hardware flow control of data from the DCE to the DTE. However many modems change the semantics of RTS so that it is used to flow control data from the modem to the terminal ("bidirectional RTS/CTS flow control") .

Exercise 85: Which data line (TxD or RxD) would RTS flow control in this case?

A number of other handshaking signals are available but are less widely used. Carrier Detect (CD) is asserted by modem-type DCEs when a carrier signal is present. This signal is typically used by system software to indicate the start and end of a dial-up session. This signal is seldom used. The RS-232 specification defines a number of other signals (e.g. a secondary serial interface) but they are almost never used.

In addition to the standard DB-25 serial connector, there are a number of smaller connectors that are widely used. These connectors are physically smaller and carry a subset of the RS-232 pins. The most common are the DB-9 connectors popular on IBM PC-AT clones, the round DIN connectors (popular on Apple computers), and the inexpensive telephone-style "RJ-11" (6-pin) and "RJ-45" (8-pin) connectors (popular on devices with many serial interfaces).

Adapters are often used not only to convert between different styles of connectors but also to convert between male and female connectors (a "gender adapter" which allows two males or two females to be connected together) and to switch between DCE and DTE pinouts (a "null modem" which allows two DCEs or two DTEs to be connected together).

Interface Voltages

The serial interface voltage levels are bipolar with respect to ground. The table below summarizes the relationship between voltage level, logical meaning on handshaking lines and data bit value (values on TxD and RxD lines).

Signal Level	Line State	For Handshaking	For Data
negative	mark	false	1
positive	space	true	0

The received signal must be greater than +3 volts to be considered positive and less than -3 volts for negative. Intermediate values are considered invalid. This allows disconnected pins to be detected.

Note: *The data lines (TxD and RxD) are asserted when* **negative**. *The control lines (e.g. CTS) are asserted when* **positive**.

Character Format

Data is transferred over the serial interface one bit at a time. A **positive** (zero) bit (the "start bit") is sent to indicate the start of the character being sent. This is followed by the bits in the character, from LS to MS bit. After sending the 7 (for plain ASCII) or 8 (for arbitrary bytes) bits, an optional parity bit (even or odd) can be sent, followed by a one "stop" bit.

Exercise 86: Draw the waveform used to send the ASCII character 'e' (hex 65) at 9600 bps with no parity.

The start bit allows a receiver to re-synchronize itself at the start of each character. This allows for small variations between transmitter and receiver timing.

Exercise 87: What happens if the receiver's clock is running faster than the transmitter clock?

The stop bit guarantees that there will be a transition at the start of each character. It also allows a receiver to re-synchronize to a character boundary in the middle of a continuous data stream. If the receiver does not see a 'one' stop bit (called a "framing error") it knows it is unsynchronized and treats that bit as a start bit. Eventually the receiver will synchronize to an actual start bit.

Exercise 88: What would happen if the receiver was expecting 8-bit characters and the transmitter was sending 7-bit characters? What about the reverse case?

There are a number of standard bit rates, typically powers of two times 1200 bps (1200, 2400, 4800 bps etc). The RS-232 standard specifies maximum bit rates, distances, etc but these are usually ignored in practical applications. For short distances it's possible to send in excess of 100 kbps.

The RS-422 serial interface specification uses a similar signaling scheme but uses differential signals (opposite voltages on two signal lines) to increase immunity to noise and increase maximum transmission distance. Data rates up to 1 Mbps are common.

Serial Interface Chips (UARTs)

Except at the lowest speeds it is not possible for a microcomputer to generate the serial bit stream using software. A peripheral chip called a Universal Asynchronous Receiver Transmitter (UART) is used to do the conversion from bytes to serial bits and vice-

versa. UARTs designed as microcomputer peripherals have control registers that are used to configure the chip for various baud rates, word lengths, number of stop bits, FIFO sizes, and interrupt options. UARTs also have status registers that can be used to indicate whether a character has been received or not, whether the previous transmitted character has been sent, and any error conditions.

Since the CPU may not be able to retrieve a received character as soon as it is read, most UARTS have FIFO (first in first out) buffers to store one or more received characters while other characters are being received.

Special chips are used to convert the logic voltages used by the UARTs to/from the RS-232 signal levels.

There are also chips (USARTs) that perform the same functions as UARTs but can also handle synchronous data streams including the HDLC framing, bit stuffing and error checking.

Synchronous Interfaces

One problem with asynchronous interfaces is the 2 bit per byte (25%) penalty imposed by the start and stop bits. In a synchronous interface the data source supplies a clock signal along with the data to be sent and the receiver is supplied with a clock as well as the received data.

Device such as modems that need to transmit the signal over a single pair of wires use circuits to regenerate the clock signal at the receiver. Synchronous serial interfaces are often embedded in links between peripherals rather than between the computer and an external device. For example all modern modems use synchronous signaling between themselves.

A problem with synchronous interfaces is that some means must be provided to find the start of the desired data within a continuous bit stream. The HDLC (High-level Data Link Control) protocol, often used over synchronous bit stream channels, uses a procedure called bit stuffing. HDLC uses "flag" sequences of a zero, 6 consecutive '1' bits and another zero at the start and end of each frame to divide the data stream into frames. In order to prevent sequences of 6 '1' bits in the data from being misinterpreted as terminating a frame, a zero bit is inserted immediately after into any data sequence of 5 '1' bits. At the receiver a '0' received after 5 '1's is removed. The HDLC frame includes one-byte address and control fields at the start of the frame and a 16-bit check-sum (a CRC, actually) at the end.

Exercise 89: Approximately how much overhead does bit stuffing add to a random data stream (hint: what is the probability that the previous 5 bits in a random data stream were 0?)?

Serial Device Interfaces

Another type of serial interface is used to connect peripheral chips such as A/D converters, serial EEPROMs, FPGAs or other microprocessors to a microprocessor. The purpose of these serial interfaces is to reduce the pin count. These interfaces are often found on microcontrollers and DSP microprocessors and include both a data line and a clock line. The clock can be driven by the processor (the CPU is called the "master") or by the source of the data (the CPU is the "slave").

Exercise 90: What type of sequential logic device(s) would be used to implement this type of serial interface?

Other Serial Interfaces

The Universal Serial Bus (USB) peripheral interface is a relatively new bus designed to connect "desktop" peripherals to a PC. It uses a 4-wire cable carrying power and a differential signaling. The data rate is either 1.5 or 12 MB/s. The data format is synchronous, with NRZI (0 is a change in level, 1 is no change in level) encoding, bit stuffing and CRCs used. An 8-bit sync pattern is sent at the start of each packet.

All devices on the USB bus are controlled by one host (typically a PC) and peripherals are connect to the host in a tree topology using "hubs.' The host polls devices for data and control messages and data are exchanged in packets. The USB protocol allows devices to be inserted and removed dynamically.

There are many other serial interface standards. For example, the IEEE 1394 ("Firewire") bus is often used in digital video systems.

Parallel I/O Ports

The interface between a CPU and I/O devices is through registers that appear in the memory or I/O

address spaces of the processor. Through these registers the CPU can input (read) or output (write) a number of bits (often one byte) at a time.

Typical examples of I/O port include output ports that drive LEDs, ports to scan a keypad, ports to control machinery, etc. More complex I/O interfaces such as floppy disk controllers or serial interface chips usually contain several I/O ports. Some ports are used to obtain status information about the interface through "status registers" and other ports can control the interface's operation through "control registers."

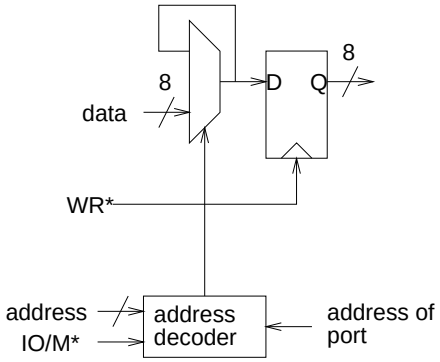
For example, a printer interface on the IBM PC has associated with it a status port that can be used to obtain certain status information (busy, on-line, out of paper, etc). The printer interface also has a control port that can be used to reset the printer and set the automatic line feed mode. In addition, there is an output port that is used to output the character to be printed.

Implementation of I/O Ports

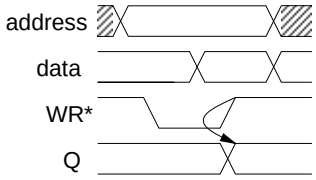
Output

Output ports are implemented using registers. The register's data inputs (D) are connected to the CPU data bus and the register's clock input is driven by a write strobe (e.g. MEMW*). In addition, an address decoder is used to make sure the register is only reloaded when the CPU is addressing the desired I/O or memory address. The rising edge of the write strobe loads the data into the register output (Q) and this output stays fixed until the register is written again.

The following schematic shows how a register could be connected to operate as an output port. The CPU's write strobe (WR*) is used to clock the data into the register, but only if the address on the CPU bus corresponds to the address of the output port:



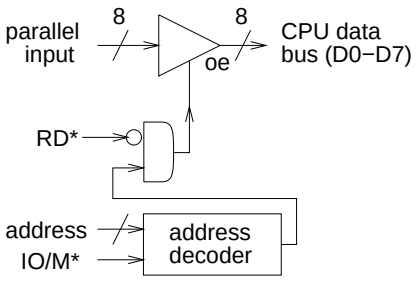
The following timing diagram shows the relationship between the signals. Note that the output is held after the rising edge of the write strobe (WR*):



Input

Input ports can also be implemented with a minimum of hardware. A tri-state buffer is used to connect the external digital input to the CPU's data bus during a read cycle if the CPU is addressing the memory or IO address assigned to that input port. The read strobe (RD*) is used to enable the buffer so that it connects the external input to the CPU data bus.

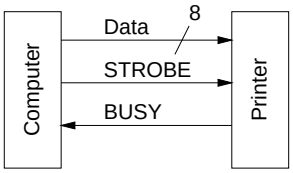
The following schematic shows how a register could be connected to operate as a parallel input port. The CPU's read strobe (RD*) is used to enable the output of a tri-state buffer when RD* is active and the address corresponds to the address of the input port:



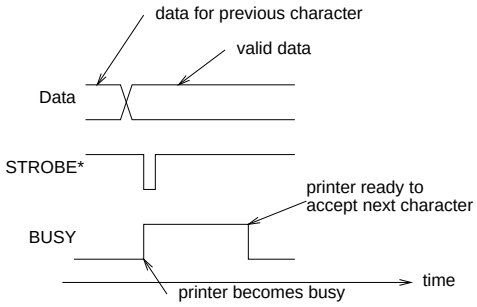
The value read by the CPU will be the value on the input at the time that the port is read.

"Centronics" Parallel Printer Port

This simple unidirectional (output only) interface is used to drive printers. There are 8 data lines and two data transfer control lines, STROBE* and BUSY. BUSY is an output from the printer that is high when the printer cannot accept data. STROBE* is a an output from the PC which is strobed (brought low and then high again) to transfer the data on the data lines to the printer. This interface uses TTL (L= 0V H$\approx +5$V) signal levels.



To write a character to the printer the computer waits until busy is low, puts the character on data lines and brings STROBE* low and then high again.



There are additional handshaking lines to control various printer features (e.g. auto line feed) and to indicate various printer status conditions (e.g. out of paper).

The original IBM PC's parallel port was an output-only Centronics-compatible interface but in recent designs the port can also be configured as an input. The maximum speed depends on the CPU speed and software used but is typically 50 to 100 kbytes/s.

IEEE standard 1284 specifies a parallel port that is bidirectional and allows for higher-speed transfers by using hardware to take care of the handshaking operations.

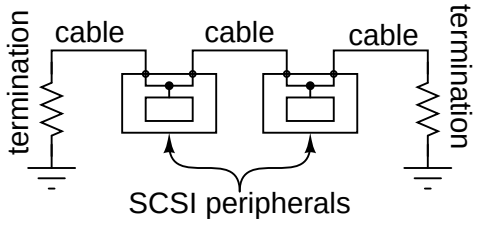
Small Computer System Interface (SCSI)

This parallel interface allows for bidirectional data transfer. Up to 8 hosts ("initiators") and peripherals ("targets") can be connected together in a bus (parallel) topology. The SCSI interface is well defined and is available on many different computers. It is widely used to connect computers to disk and tape drives, CD-ROMs, scanners, high-speed printers, etc.

The SCSI interface includes a protocol for arbitrating access to the bus by initiators and for selecting a specific target. The actual data transfers over the SCSI bus use a similar request/acknowledge protocol with each byte transfer being acknowledged by the target before another byte is transferred.

Depending on the speed of the peripheral and the host interface the bus can transfer data at up to several megabytes per second. The SCSI devices attached to the bus are electrically connected in parallel with each device configured to respond to a particular bus ID number (ID).

The physical interface uses a 50-pin cable with two connectors on each device so that they can be daisy-chained. Because of the high bus speeds, care has to be taken to properly terminate the bus in it's characteristic impedance to minimize reflections. The SCSI bus uses TTL-level signals but, since the signals are bi-directional, open-collector or tri-state drivers are used.



Another advantage of the SCSI interface is that it defines a set of common commands for devices with similar characteristics. This allows the same software to drive different devices. For example, the same generic commands (rewind, skip forward, etc) can be used to control tape drives from different manufacturers.

Although a SCSI interface can be built using a simple parallel interface and programmed i/o, this type of interface is too slow for most applications.

SCSI interface chips are available that implement the interface between the CPU and the SCSI bus and transfer data using either DMA or on-board FIFOs.

Software Aspects

The value on the output port is set with MOV (if the port is memory-mapped) or OUT (if the port is mapped into the I/O space) instructions. Similarly, the value on an input port is read with a MOV or IN instruction.

It's often necessary to set or clear a particular bit on an output port or to test the value of a particular bit on an input port. This can be done with bit masks and the bit-wise logical operations AND and OR.

Other Parallel Interfaces

The IDE/ATAPI parallel bus is used mainly to interface a CPU to disk drivers. It is commonly seen on IBM-PC compatible computers.

The IEEE-488 standard (also known as the General Purpose Interface Bus (GPIB) and HPIB) is another bidirectional interface. Like the SCSI bus it allows multiple bus masters ("talkers") and slaves ("listeners"). It was developed by HP who named it HPIB (HP Interface Bus). The standard is called IEEE-488. This bus is used mostly for control of laboratory instruments.