C h a p t e r **7 INPUT/OUTPUT**

- **I/O Module**

The I/O module contains logic for performing a communication function between the peripheral and the bus.
Thus, an I/O module is required. This module has two major functions (Figure 7.1):
• Interface to the processor and memory via the system bus or central switch
• Interface to one or more peripheral devices by tailored data links.

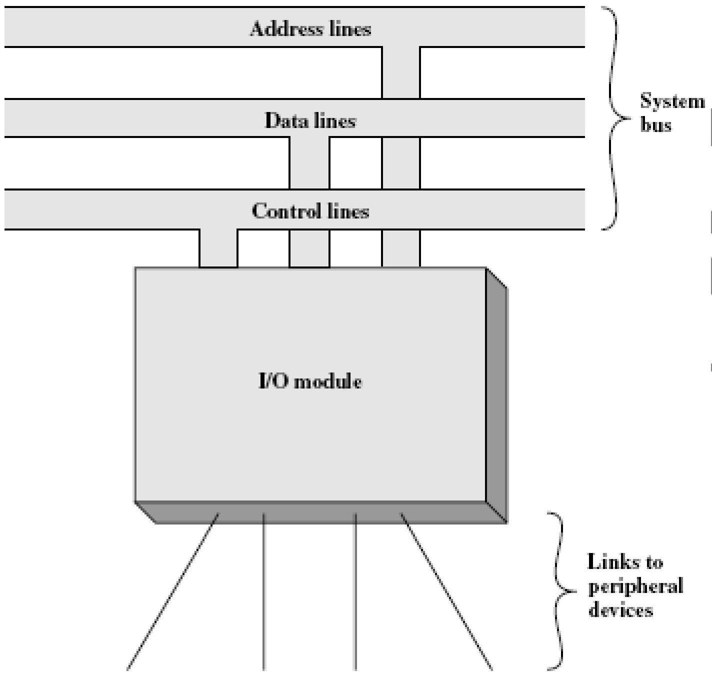


Figure 7.1 Generic Model of an I/O Module

- **Block Diagram of I/O module**

Figure 7.3 provides a general block diagram of an I/O module. The module connects to the rest of the computer through a set of signal lines (e.g., system bus lines). Data transferred to and from the module are buffered in one or more data registers. There may also be one or more status registers that provide current status information. A status register may also function as a control register, to accept detailed control information from the processor

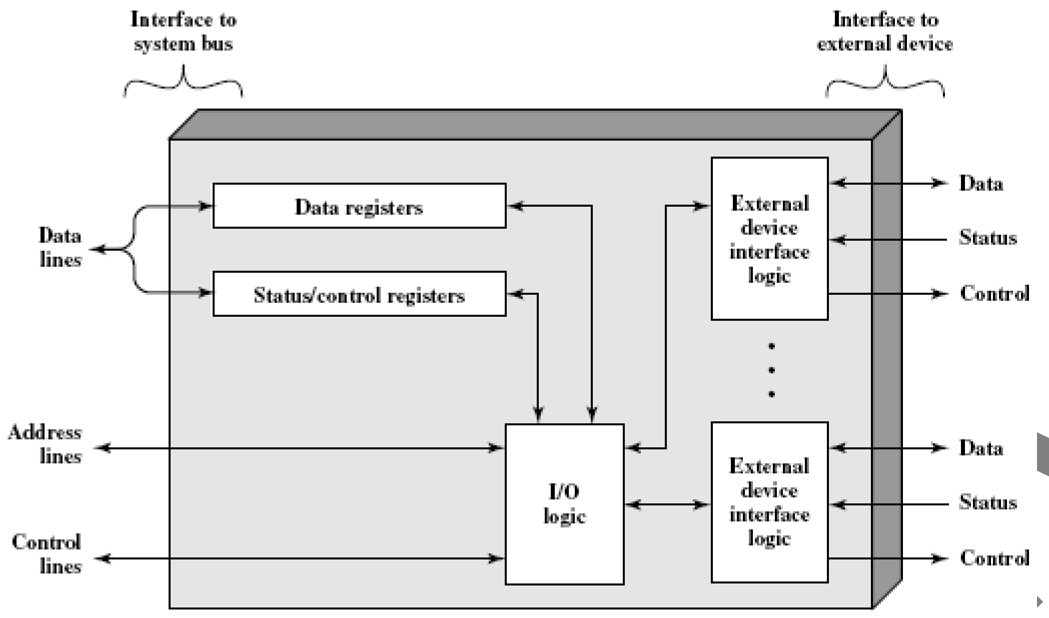


Figure 7.3 Block Diagram of an I/O Module

An I/O module that takes on most of the detailed processing burden, presenting a high-level interface to the processor, is usually referred to as an *I/O channel* or *I/O processor*. An I/O module that is quite primitive and requires detailed control is usually referred to as an *I/O controller* or *device controller*. I/O controllers are commonly seen on microcomputers, whereas I/O channels are used on mainframes.

Programmed I/O

With *programmed I/O*, data are exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data. When the processor issues a command to the I/O module, it must wait until the I/O operation
is complete.

Interrupt Driven I/O

The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The processor, while waiting, must repeatedly interrogate the status of the I/O module.

- **Intel 82C59A Interrupt Controller**

The Intel 80386 provides a single Interrupt Request (INTR) and a single Interrupt Acknowledge (INTA) line. To allow the 80386 to handle a variety of devices and priority structures, it is usually configured with an external interrupt arbiter, the 82C59A.

Figure 7.8 shows the use of the 82C59A to connect multiple I/O modules for the 80386.A single 82C59A can handle up to eight modules. If control for more than eight modules is required, a Cascade arrangement can be used to handle up to 64 modules.

TRISFY

The 82C59A's sole responsibility is the management of interrupts. It accepts interrupt requests from attached modules, determines which interrupt has the highest priority, and then signals the processor by raising the INTR line. The processor acknowledges via the INTA line.

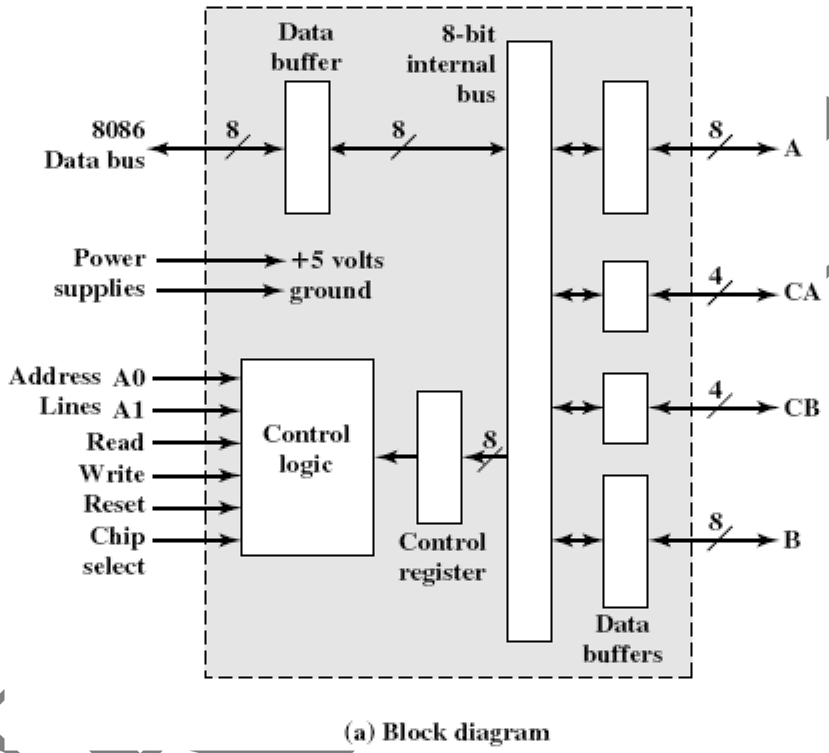


Figure 7.9 the Intel 82C55A Programmable Peripheral Interface

- **Direct Memory Access (DMA)**

When the processor wishes to read or write a block of data, it issues a command to the DMA module, by sending to the DMA module the following information:
• Whether a read or write is requested, using the read or write control line between the processor and the DMA module
• The address of the I/O device involved, communicated on the data lines
• The starting location in memory to read from or write to communicated on the data lines and stored by the DMA module in its address register
• The number of words to be read or written, again communicated via the data lines and stored in the data count register

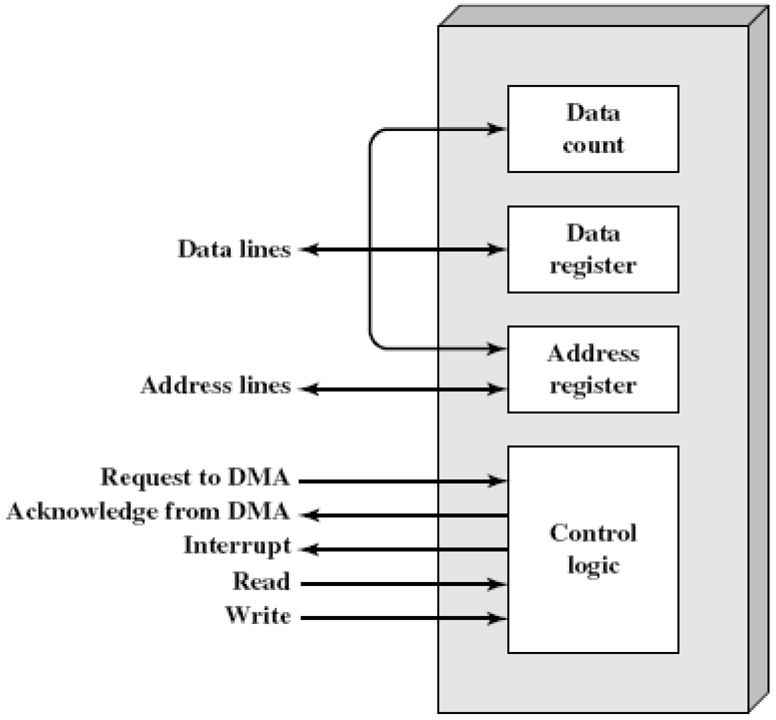


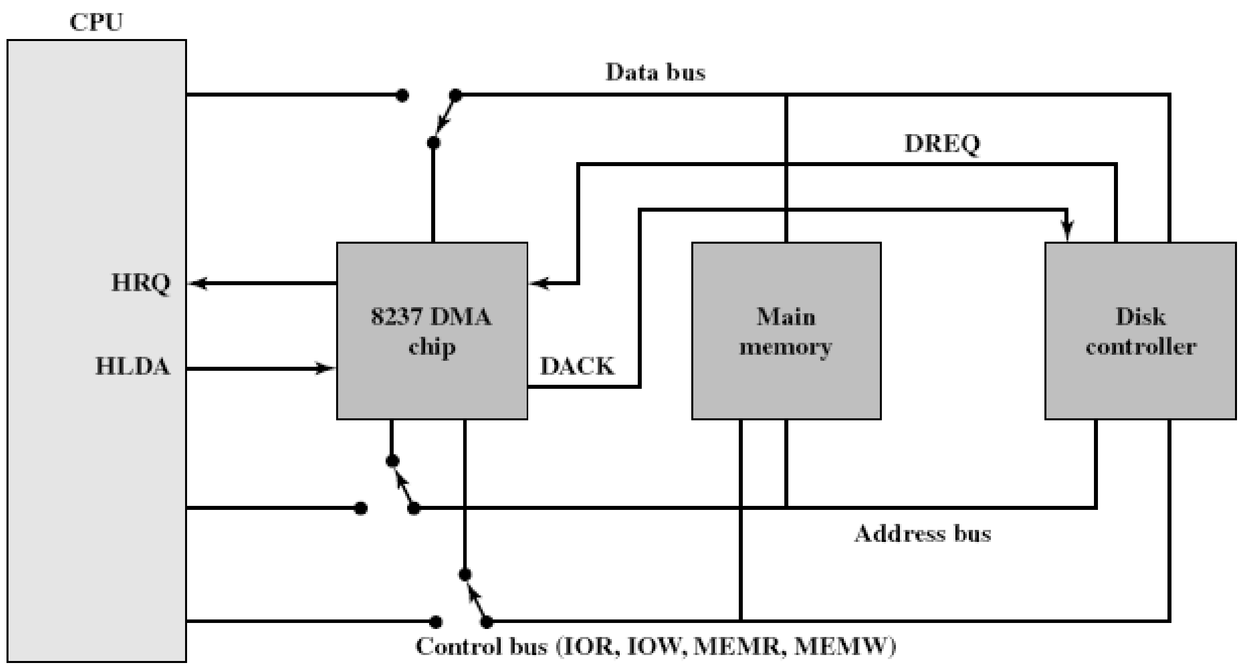
Figure 7.11 Typical DMA Block Diagram

- **8237 DMA Controller**

The Intel 8237A DMA controller interfaces to the 80x 86 families of processors and to DRAM memory to provide a DMA capability. Figure 7.14 indicates the location of the DMA module. When the DMA module needs to use the system buses (data, address, and control) to transfer data, it sends a signal called HOLD to the processor.

The processor responds with the HLDA (hold acknowledge) signal, indicating that
1. The peripheral device (such as the disk controller) will request the service of DMA by pulling DREQ (DMA request) high.
2. The DMA will put a high on its HRQ (hold request), signaling the CPU through its HOLD pin that it needs to use the buses.
3. The CPU will finish the present bus cycle (not necessarily the present instruction) and respond to the DMA request by putting high on its HDLA (hold acknowledges),
4. DMA will activate DACK (DMA acknowledge), which tells the peripheral device that it will start to transfer the data.
5. DMA starts to transfer the data from memory to peripheral by putting the address of the first byte of the block on the address bus and activating MEMR, thereby reading the byte from memory into the data bus; it then activates IOW to write it to the peripheral. Then DMA decrements the counter and increments the address pointer and repeats this process until the count reaches zero and the task is finished.
6. After the DMA has finished its job it will deactivate HRQ, signaling the CPU that it can regain control over its buses.

TRIS



DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge
HRQ = HOLD request

Figure 7.14 8237 DMA Usage of System Bus

TRL

- **I/O Channels**

The I/O channel represents an extension of the DMA concept. An I/O channel has the ability to execute I/O instructions, which gives it complete control over I/O operations. In a computer system with such devices, the CPU does not execute I/O instructions. Such instructions are stored in main memory to be executed by a special-purpose processor in the I/O channel itself.

Two types of I/O channels are common, as illustrated in Figure 7.15. A *selector channel* controls multiple high-speed devices and, at any one time, is dedicated to the transfer of data with one of those devices. Thus, the I/O channel selects one device and affects the data transfer.

Multiplexor channel can handle I/O with multiple devices at the same time. For low-speed devices, a *byte multiplexor* accepts or transmits characters as fast as possible to multiple devices. For example, the resultant character stream from three devices with different rates and individual streams A1A2A3A4 . . ., B1B2B3B4 . . ., and C1C2C3C4 . . . might be A1B1C1A2C2A3B2C3A4, and so on.

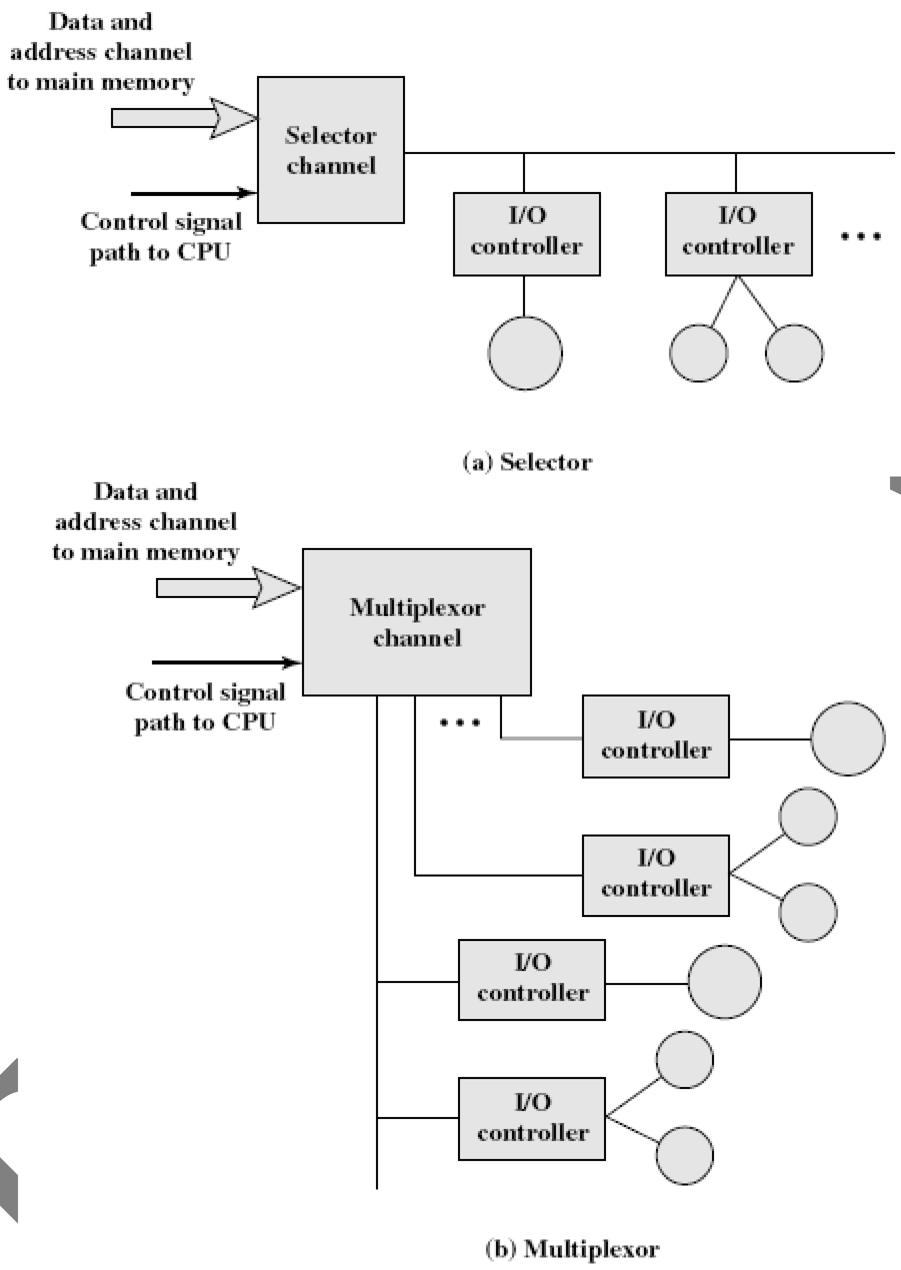


Figure 7.15 I/O Channel Architecture