14

Input / Output Concepts And Terminology

14.1 Introduction

Previous chapters of the text describe two of the major components found in a computer system: processors and memories. In addition to describing technologies used for each component, the chapters illustrate how processors and memory interact.

This chapter introduces the third major aspect of architecture, connections between a computer and the external world. We will learn that on most computers, the connection between a processor and an I/O device uses the same basic paradigm as the connection between a processor and memory. Furthermore, we will see that although they operate under control of a processor, I/O devices can interact directly with memory.

14.2 Input And Output Devices

The earliest electronic computers, which consisted of a numerical processor plus a memory, resembled a calculator more than a modern computer. The human interface was crude — values were entered through a set of manual switches, and results of calculations were viewed through a series of lights. By the late 1940s, it had become obvious that better interfaces were needed before digital computers could be useful for more than basic calculations. Engineers began devising ways to connect computers to external devices, which became known as *Input* and *Output* (*I/O*) devices. Modern I/O devices include cameras, earphones, and microphones, as well as keyboards, mice, monitors, sensors, hard disks, DVD drives and printers.

14.3 Control Of An External Device

The earliest external devices attached to computers consisted of independent units that operated under control of the CPU. That is, an external device usually occupied a separate physical cabinet, had an independent source of electrical power, and contained internal circuitry that was separate from the computer. The small set of wires that connected the computer to the external device, only carried control signals (i.e., signals from the digital logic in the computer to the digital logic in the device). Circuitry in the device monitored the control signals, and changed the device accordingly.

Many early computers provided a set of lights that displayed values. Typically, the display contained one light for each bit in the computer's accumulator — the light was on when the bit was set to one, and off when the bit was zero. However, it is not possible to connect a light bulb directly to the accumulator circuit because even a small light bulb requires more power than a digital circuit can deliver. Therefore, a display unit contains circuitry that receives a set of digital logic signals and controls a set of light bulbs accordingly. Figure 14.1 illustrates how the hardware is organized.

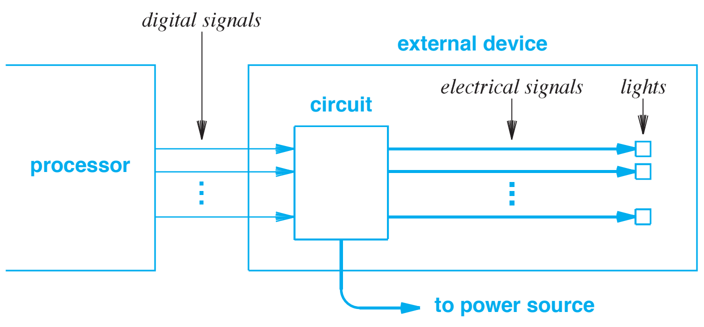

Figure 14.1 Example of an external circuit that controls a set of lights. The device contains circuitry that converts incoming digital logic signals into the signals needed to operate a set of light bulbs.

As the figure illustrates, we think of an external device as independent from the processor except for digital signals that pass between them. In practice, of course, some devices reside in the same enclosure with the processor, and both receive power from a common source. We will ignore such details and concentrate on the control signals.

A computer interacts with a device in two ways: the computer controls a device and the computer exchanges data with a device. For example, a processor can start a disk spinning, control the volume on an external speaker, tell a camera to snap a picture, or turn off a printer. In the next chapter, we will learn how a computer passes control information to external devices.

14.4 Data Transfer

Although control of external devices is essential, for most devices, control functions are secondary. The primary function of external devices is *data transfer*. Indeed, most of the architectural choices surrounding external devices focus on mechanisms that permit the device and processor to exchange data.

We will consider several questions regarding data transfer. First, exactly how is data communicated? Second, which side initiates transfer (i.e., does the processor request a transfer or does the device)? Third, what techniques and mechanisms are needed to enable high-speed transfers?

Other questions that are less pertinent to programmers concern low-level details. What voltages are used to communicate with an external device, and how is data represented? The answers depend on the type of device, the speed with which data must be transferred, the type of cabling used, and the distance between the processor and the device. However, as Figure 14.1 illustrates, the digital signals used internally by a processor are insufficient to drive circuits in an external device.

Because the voltages and encodings used for external connections differ from those used internally, special hardware is needed to translate between the two representations. We use the term *interface controller* to refer to the hardware that provides the interface to an external device. Figure 14.2 illustrates that interface controllers are needed at both ends of a physical connection.

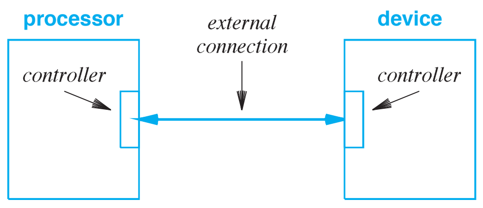

Figure 14.2 Illustration of controller hardware on each end of an external connection. The voltages and signals used on the external connection can differ from the voltages used internally.

14.5 Serial And Parallel Data Transfers

All the I/O interfaces on a computer can be classified in two broad categories:

- Parallel interface
- Serial interface

Parallel Interface. An interface between a computer and an external device is classified as *parallel* if the interface allows the transfer of multiple bits of data simultaneously. In essence, a parallel interface contains many wires — at any instant, each wire carries one bit of data.

We use the term *interface width* to refer to the number of parallel wires an interface uses. Thus, one might hear an engineer talk about an eight-bit interface or a sixteen-bit interface. We will learn more about how interfaces use parallel wires in the next chapter.

Serial Interface. The alternative to a parallel interface is one in which only one bit of data can be transferred at any time; an interface that transfers one bit at a time is classified as *serial*.

The chief advantages of a serial interface are fewer wires and less interference from signals traveling at the same time. In principle, only two wires are needed for serial data transmission — one to carry the signal and a second to serve as an electrical ground against which voltage can be measured. The chief disadvantage of a serial interface arises from increased latency: when sending multiple bits, serial hardware must wait until one bit has been sent before sending another.

14.6 Self-Clocking Data

Recall that digital circuits operate according to a *clock*, a signal that pulses continuously. Clocks are especially significant for I/O because each I/O device and processor can have a separate clock rate (i.e., each controller can have its own clock). Thus, one of the most significant aspects of an external interface concerns how the interface accommodates differences in clock rates.

The term *self-clocking* describes a mechanism in which signals sent across an interface contain information that allows the receiver to determine exactly how the sender encoded the data. For example, some external devices use a method similar to the clockless logic mechanism that Chapter 2 describes†. Others employ an extra set of wires that pass clocking information: when transmitting data, the sender uses the extra wires to inform the receiver about the location of bit boundaries in the data.

14.7 Full-Duplex And Half-Duplex Interaction

Many external I/O devices provide *bidirectional transfer* which means the processor can send data to the device or the device can send data to the processor. For example, a disk drive supports both *read* and *write* operations. Interface hardware uses two methods to accommodate bidirectional transfer:

- Full-duplex interaction
- Half-duplex interaction

Full-Duplex Interaction. An interface that allows transfer to proceed in both directions simultaneously is known as a *full-duplex* interface. In essence, full-duplex

†The description of clockless logic can be found on page 37.

hardware consists of two parallel devices with two independent sets of wires connecting them. One set is used to transfer data in each direction.

Half-Duplex Interaction. The alternative to a full-duplex interface, known as a *half-duplex* interface, only allows transfer to proceed in one direction at a time. That is, a single set of wires that connects the processor and the external device must be shared. In the next chapter, we will see that sharing requires negotiation — before it can perform a transfer, a processor or device must wait for the current transfer to finish, and must obtain exclusive use of the underlying wires.

14.8 Interface Throughput And Latency

The *throughput* of an interface is measured in the number of bits that can be transferred per unit time, and is usually measured in *Megabits per second* (*Mbps*) or *Megabytes per second* (*MBps*). It may seem that a serial interface would always have a lower throughput because serial transmission only transfers one bit at a time, whereas a parallel interface can transfer multiple bits at the same time. However, when parallel wires are close together, the data rate must be limited or electromagnetic interference can result. Therefore, in some cases, engineers have been able to send bits over a serial interface with higher throughput than a parallel interface.

The second major measure of an interface is *latency*. Latency refers to the delay between the time a bit is sent and the time the bit is received (i.e., how long it takes to transfer a single bit), which is usually measured in nanoseconds (ns). As we have seen with memories, we must be careful to distinguish between latency and throughput because some devices need low latency and others need high throughput.

We can summarize:

The latency of an interface is a measure of the time required to perform a transfer, the throughput of an interface is a measure of the data that can be transferred per unit time.

14.9 The Fundamental Idea Of Multiplexing

It may seem that choosing an interface is trivial: we want full-duplex, low latency, and high throughput. Despite the desire for high performance, many other factors make the choice of interfaces complex. Recall, for example, each integrated circuit has a fixed number of *pins* that provide external connections. A wider interface uses more pins, which means fewer pins for other functions. An interface that provides full-duplex capability uses approximately twice as many pins as an interface that provides half-duplex capability.

Most architects choose a compromise for external connections. The connection has *limited parallelism*, and the hardware uses a technique known as *multiplexing* to send data. Although details are complex, the concept of multiplexing is easy to understand. The idea is that the hardware breaks a large data transfer into pieces and sends one piece at a time. We use the terms *multiplexor* and *demultiplexor* to describe the hardware that handles data multiplexing. For example, Figure 14.3 illustrates how multiplexing hardware can divide sixty-four bits of data into sixteen-bit chunks and transmit chunks over an interface that has a width of sixteen bits. Only one chunk can be sent at a given time.

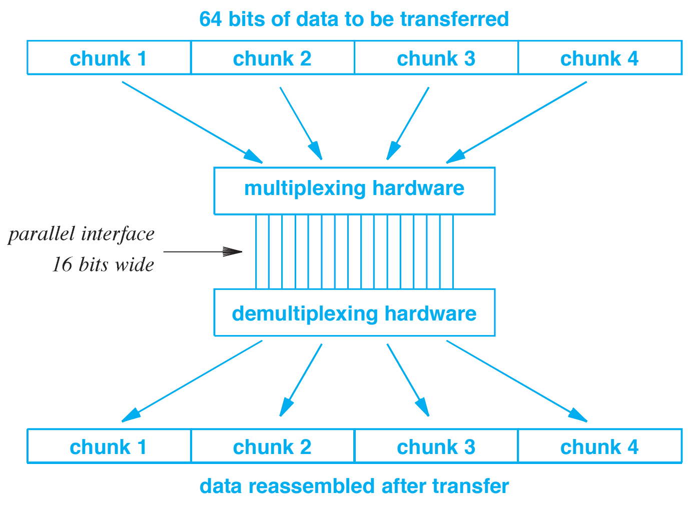

Figure 14.3 Illustration of the transfer of sixty-four bits of data over a sixteen-bit interface. Multiplexing hardware divides the data into sixteen-bit units and sends one unit at a time.

In practice, most physical connections between a processor and external devices use multiplexing. Doing so allows the processor to transfer arbitrary amounts of data without devoting many physical pins to the connection. In the next chapter, we will learn how multiplexing also improves CPU performance.

To summarize:

Multiplexing is used to construct an I/O interface that can transfer arbitrary amounts of data over a fixed number of parallel wires. Multiplexing hardware divides the data into blocks, and transfers each block independently.

Note that our definition applies equally to serial transmission — we simply interpret a serial interface as multiplexing transfers over a single wire. Thus, the chunk size for a serial interface is a single bit.

14.10 Multiple Devices Per External Interface

The examples in this chapter imply that each external connection from a processor attaches to one device. To help conserve pins and external connections, most processors do not have a single device per external connection. Instead, a set of pins attaches to multiple devices, and the hardware is configured to permit the processor to communicate with one of the devices at a given time. The next chapter explains the concept in detail and gives examples.

14.11 A Processor's View Of I/O

Recall that interface controller hardware is associated with an external connection. Thus, when a processor interacts with an external device, the processor must do so through the controller. The processor makes requests to the controller, and receives replies; the controller translates each request into the appropriate external signals that perform the requested function on the external device. The point is that the processor can only interact with the interface controller and not with the external device.

To capture the architectural concept, we say that the controller presents a *programming interface* to the processor. Interestingly, the programming interface does not need to model the operations of the underlying device exactly. In the next chapter, we will see an example of a widely used programming interface that casts all external interactions into a simplified paradigm. To summarize:

> *A processor uses interface controller hardware to interact with a device; the controller translates requests into the appropriate external signals.*

14.12 Summary

Computer systems interact with external devices either to control the device (e.g., to change the status) or to transfer data. An external interface can use a serial or parallel approach; the number of bits that can be sent simultaneously is known as the *width* of a parallel interface. A bidirectional interface can use full-duplex or half-duplex interaction.

There are two measures of interface performance. Latency refers to the time required to send a bit from a given source to a given destination (e.g., from memory to a printer), and throughput refers to the number of bits that can be sent per unit time.

Because the number of pins is limited, a processor does not have arbitrarily wide external connections. Instead, interface hardware is designed to multiplex large data transfers over fewer pins. In addition, multiple external devices can attach to a single external connection; the interface controller hardware communicates with each device separately.

EXERCISES

14.1 The speaker in a smart phone or laptop is, in fact, an analog device in which the volume is proportional to the voltage supplied. Does that mean a processor must have an analog output for audio? Explain.

14.2 What are the primary and secondary functions associated with external devices?

14.3 Can a device that operates on 3.3-volt digital signals be connected to a processor that operates on 5-volt digital signals? Explain.

14.4 If the *interface width* is 16, is the interface parallel or serial? Explain.

14.5 USB is classified as a serial interface. What does the classification mean?

14.6 Suppose you are purchasing a network I/O device, and the vendor gives you the choice of a half-duplex or full-duplex interface. Which do you choose and why?

14.7 If the interface between a processor and storage device has a width of thirty-two bits, how can the processor transfer a data item that consists of sixty-four bits?

14.8 Create a parallel interface that is self-clocking and can send data from one side to the other. Hint: have two wires that the two ends use to coordinate and additional wires that are used to transfer data.

14.9 Suppose a serial interface has a latency of 200 microseconds. How long does it take to transfer one bit over the interface? How long does it take to transfer sixty-four bits over the interface?

14.10 Suppose a parallel interface has a width of thirty-two bits and a latency of 200 microseconds. How long does it take to transfer thirty-two bits over the interface? How long does it take to transfer sixty-four bits over the interface? Explain.