is needed for the communications interface, a burst of data can be read and stored in a FIFO. Each time the interface is ready for a new byte, it reads it from the FIFO. In this case, only a single-clock FIFO is required, because these devices operate on a common clock domain. To keep this process running smoothly, control logic is needed to watch the state of the FIFO and perform a new burst read from DRAM when the FIFO begins to run low on data. This scheme is illustrated in Fig. 4.20.

For data-rate matching to work properly, the average bandwidth over time of the input and output ports of the FIFO must be equal, because FIFO capacity is finite. If data is continuously written faster than it can be read, the FIFO will eventually overflow and lose data. Conversely, if data is continuously read faster than it can be written, the FIFO will underflow and cause invalid bytes to be inserted into the outgoing data stream. The depth of a FIFO indicates how large a read/write rate disparity can be tolerated without data loss. This disparity is expressed as the product of rate mismatch and time. A small mismatch can be tolerated for a longer time, and a greater rate disparity can be tolerated for a shorter time.

In the rate-matching example, a large rate disparity of brief duration is balanced by a small rate disparity of longer duration. When the DRAM is read, a burst of data is suddenly written into the FIFO, creating a temporarily large rate disparity. Over time, the communications interface reads one byte at a time while no writes are taking place, thereby compensating with a small disparity over time.

DRAM reads to refill the FIFO must be carefully timed to simultaneously prevent overflow and underflow conditions. A threshold of FIFO fullness needs to be established below which a DRAM read is triggered. This threshold must guarantee that there is sufficient space available in the FIFO to accept a full DRAM burst, avoiding an overflow. It must also guarantee that under the worst-case response time of the DRAM, enough data exists in the FIFO to satisfy the communications interface, avoiding an underflow. In most systems, the time between issuing a DRAM read request and actually getting the data is variable. This variability is due to contention with other requesters (e.g., the CPU) and waiting for overhead operations (e.g., refresh) to complete.

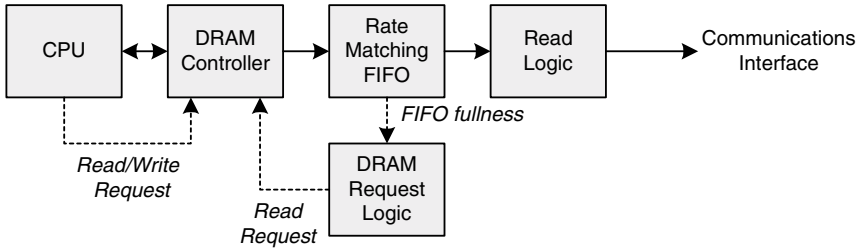

FIGURE 4.20 Synchronous FIFO application: data rate matching.

CHAPTER 5
Serial Communications

Serial communication interfaces are commonly used to exchange data with other computers. Serial interfaces are ubiquitous, because they are economical to implement over long distances as a result of their requirement of relatively few wires. Many types of serial interfaces have been developed, with speeds ranging to billions of bits per second. Regardless of the bit rate, serial communication interfaces share many common traits. This chapter introduces the fundamentals of serial communication in the context of popular data links such as RS-232 and RS-485 in which bandwidths and components lend themselves to basic circuit fabrication techniques.

The chapter first deals with the basic parallel-to-serial-to-parallel conversion process that is at the heart of all serial communication. Wide buses must be serialized at the transmitter and reconstructed at the receiver. Techniques for accomplishing this vary with the specific type of data link, but basic concepts of framing and error detection are universal.

Two widely deployed point-to-point serial communication standards, RS-232 and RS-422, are presented, along with the standard ASCII character set, to see how theory meets practice. Standards are important to communications in general because of the need to connect disparate equipment. ASCII is one of the most fundamental data representation formats with global recognition. RS-232 has traditionally been found in many digital systems, because it is a reliable standard. Understanding RS-232, its relative RS-422, and ASCII enables an engineer to design a communication interface that can work with an almost infinite range of complementary equipment ranging from computers to modems to off-the-shelf peripherals.

Systems may require more advanced communication schemes to enable data exchange between many nodes. Networks enable such communication and can range in complexity according to an application's requirements. Networking adds a new set of fundamental concepts on top of basic serial communication. Topics including network topologies and packet formats are presented to explain how networks function at a basic hardware and software level. Once networking fundamentals have been discussed, the RS-485 standard is introduced to show how a simple and fully functional network can be constructed. A complete network design example using RS-485 is offered with explanations of why various design points are included and how they contribute to the network's overall operation.

The chapter closes with a presentation of small-scale networking employed within a digital system to economically connect peripherals to a microprocessor. Interchip networks are of such narrow scope that they are usually not referred to as networks, but they can possess many fundamental properties of a larger network. Peripherals with low microprocessor bandwidth requirements can be connected using a simple serial interface consisting of just a few wires, as compared to the full complexity of a parallel bus.