TABLE 1.18 Tri-state Buffer Truth Table

EN	D	Y
0	x	Z
1	0	0
1	1	1

ers can be arbitrarily expanded by adding more devices without the need to add a full set of input or output signals each time a new device is added. In a logical context, a *bus* is a collection of wires that serve a common purpose. For example, a computer's data bus might be eight wires that travel together and collectively represent a byte of data. Electrical contention on a bus is often called a *bus-fight*. Schematically, multiple tri-state buffers might be drawn as shown in Fig. 1.23.

Each tri-state buffer contains its own enable signal, which is usually driven by some type of decoder. The decoder guarantees that only one tri-state buffer is active at any one time, preventing contention on the common wire.

Registers are collections of multiple flops arranged in a group with a common function. They are a common synchronous-logic building block and are commonly found in multiples of 8-bit widths, thereby representing a byte, which is the most common unit of information exchange in digital systems. An 8-bit register provides a common clock and clock enable for all eight internal flops. The clock enable allows external control of when the flops get reloaded with new D-input values and when they retain their current values. It is common to find registers that have a built-in tri-state buffer, allowing them to be placed directly onto a shared bus without the need for an additional tri-state buffer component.

Whereas normal registers simply store values, synchronous elements called *shift registers* manipulate groups of bits. Shift registers exist in all permutations of serial and parallel inputs and outputs. The role of a shift register is to somehow change the sequence of bits in an array of bits. This includes creating arrays of bits from a single bit at a time (serial input) or distributing an array of bits one bit at a time (serial output). A serial-in, parallel-out shift register can be implemented by chaining several flops together as shown in Fig. 1.24.

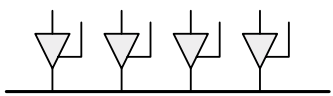

FIGURE 1.23 Multiple tri-state buffers on a single wire.

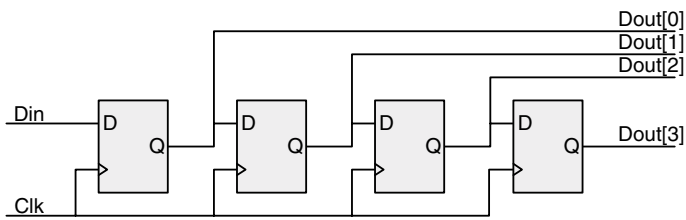

FIGURE 1.24 Serial-in, parallel-out shift register.

On each rising clock edge, a new serial input bit is clocked into the first flop, and each flop in succession loads its new value based on its predecessor's value. At any given time, the parallel output of an N-bit shift register reflects the state of the last N bits shifted in up to that time. In this example (N = 4), a serial stream of bits collected in four clock cycles can be operated upon as a unit of four bits once every fourth cycle. As shown, data is shifted in MSB first, because Dout[3] is shown in the last bit position. Such a simple transformation is useful, because it is often more practical to communicate digital data in serial form where only one bit of information is sent per clock cycle, but impractical to operate on that data serially. An advantage of serial communication is that fewer wires are required as compared to parallel. Yet, parallel representation is important because arithmetic logic can get overly cumbersome if it has to keep track of one bit at a time. A parallel-in, serial-out shift register is very similar, as shown in Fig. 1.25, with the signals connected for MSB first operation to match the previous example.

Four flops are used here as well. However, instead of taking in one bit at a time, all flops are loaded when the load signal is asserted. The 2-to-1 muxes are controlled by the load signal and determine if the flops are loaded with new parallel data or shifted serial data. Over each of the next four clock cycles, the individual bits are shifted out one at a time. If these two shift register circuits were connected together, a crude serial data communications link could be created whereby parallel data is converted to serial and then back to parallel at each end.

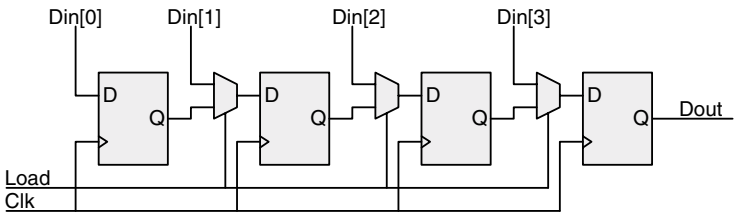

FIGURE 1.25 Parallel-in, serial-out shift register.