decisions that a designer must make when an entire circuit does not fit into a single component.

A VHDL-based approach is especially appropriate for larger designs that will be realized in a single CPLD, FPGA or ASIC, as described in the third section. You may notice that these examples do not target a specific CPLD or FPGA. Indeed, this is one of the benefits of HDL-based design; most or all of the design effort is "portable" and can be targeted to any of a variety of technologies.

The only prerequisites for this chapter are the chapters that precede it. The three sections are written to be pretty much independent of each other, so you don't have to read about ABEL if you're only interested in VHDL, or vice versa. Also, the rest of the book is written so that you can read this chapter now or skip it and come back later.

6.1 Building-Block Design Examples

6.1.1 Barrel Shifter

barrel shifter

A *barrel shifter* is a combinational logic circuit with *n* data inputs, *n* data outputs, and a set of control inputs that specify how to shift the data between input and output. A barrel shifter that is part of a microprocessor CPU can typically specify the direction of shift (left or right), the type of shift (circular, arithmetic, or logical), and the amount of shift (typically 0 to *n*–1 bits, but sometimes 1 to *n* bits).

In this subsection, we'll look at the design of a simple 16-bit barrel shifter that does left circular shifts only, using a 4-bit control input S[3:0] to specify the amount of shift. For example, if the input word is ABCDEFGHGIHKLMNOP (where each letter represents one bit), and the control input is 0101 (5), then the output word is FGHGIHKLMNOPABCDE.

From one point of view, this problem is deceptively simple. Each output bit can be obtained from a 16-input multiplexer controlled by the shift-control inputs, which each multiplexer data input connected to the appropriate On the other hand, when you look at the details of the design, you'll see that there are trade-offs in the speed and size of the circuit.

Let us first consider a design that uses off-the-shelf MSI multiplexers. A 16-input, one-bit multiplexer can be built using two 74x151s, by applying S3 and its complement to the EN_L inputs and combining the Y_L data outputs with a NAND gate, as we showed in Figure 5-66 for a 32-input multiplexer. The low-order shift-control inputs, S2–S0. connect to the like-named select inputs of the '151s.

We complete the design by replicating this 16-input multiplexer 16 times and hooking up the data inputs appropriately, as shown in Figure 6-1. The top '151 of each pair is enabled by S3_L, and the bottom one by S3; the remaining select bits are connected to all 32 '151s. Data inputs D0–D7 of each '151 are connected to the DIN inputs in the listed order from left to right.

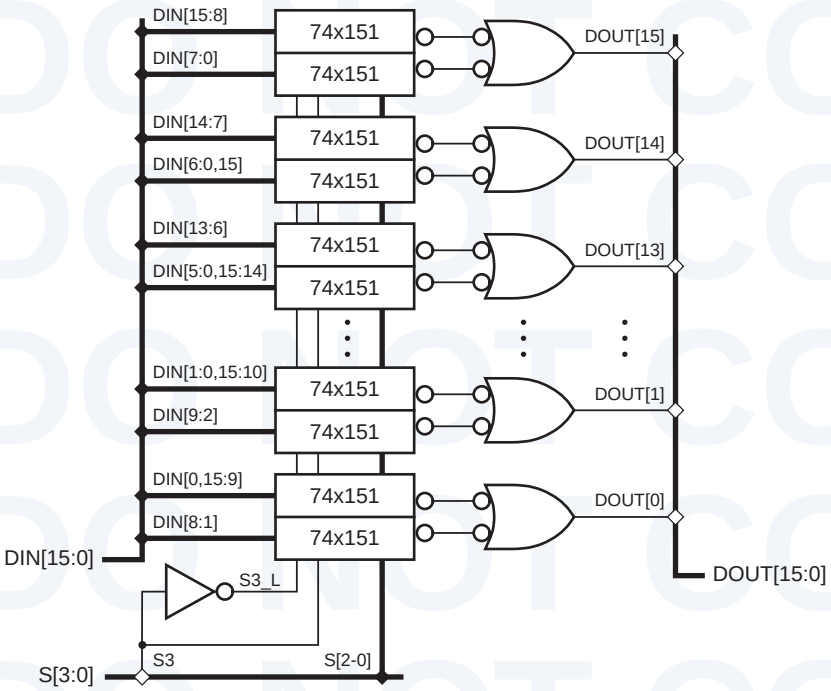

Figure 6-1
One approach to building a 16-bit barrel shifter.

The first row of Table 6-1 shows the characteristics of this first approach. About 36 chips (32 74x151s, 4 74x00s, and 1/6 74x04) are used in the MSI/SSI realization. We can reduce this to 32 chips by replacing the 74x151s with 74x251s and tying their three-state Y outputs together, as tabulated in the second row. Both of these designs have very heavy loading on the control inputs; each of the control bits S[2:0] must be connected to the like-named select input of all 32 multiplexers. The data inputs are also fairly heavily loaded; each data bit must connect to 16 different multiplexer data inputs, corresponding to the 16 possible shift amounts. However, assuming that the heavy control and data loads don't slow things down too much, the 74x251-based approach yields the shortest data delay, with each data bit passing through just one multiplexer.

Alternatively, we could build the barrel shifter using 16 74x157 2-input, 4-bit multiplexers, as tabulated in the last row of the table. We start by using four 74x157s to make a 2-input, 16-bit multiplexer. Then, we can hook up a first set

Table 6-1
Properties of four different barrel-shifter design approaches.

Multiplexer Component	*Data Loading*	*Data Delay*	*Control Loading*	*Total ICs*
74x151	16	2	32	36
74x251	16	1	32	32
74x153	4	2	8	16
74x157	2	4	4	16

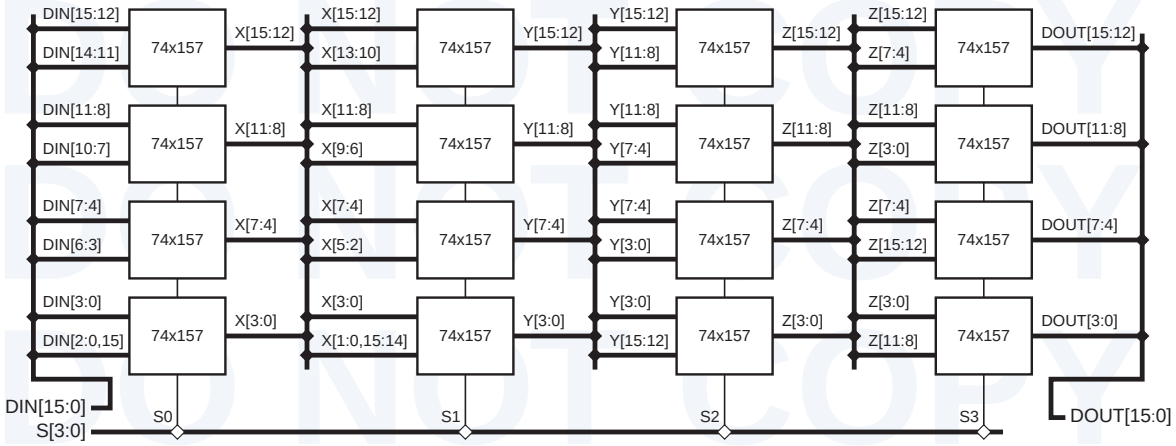

Figure 6-2 A second approach to building a 16-bit barrel shifter.

of four ’157s controlled by S0 to shift the input word left by 0 or 1 bit. The data outputs of this set are connected to the inputs of a second set, controlled by S1, which shifts its input word left by 0 or 2 bits. Continuing the cascade, a third and fourth set are controlled by S2 and S3 to shift selectively by 4 and 8 bits, as shown in Figure 6-2. Here, the 1A-4A and 1B-4B inputs and the 1Y-4Y outputs of each ’157 are connected to the indicated signals in the listed order from left to right.

The ’157-based approach requires only half as many MSI packages and has far less loading on the control and data inputs. On the other hand, it has the longest data-path delay, since each data bit must pass through four 74x157s.

Halfway between the two approaches, we can use eight 74x153 4-input, 2-bit multiplexers two build a 4-input, 16-bit multiplexer. Cascading two sets of these, we can use S[3:2] to shift selectively by 0, 4, 8, or 12 bits, and S[1:0] to shift by 0–3 bits. This approach has the performance characteristics shown in the third row of Table 6-1, and would appear to be the best compromise if you don’t need to have the absolutely shortest possible data delay.

The same kind of considerations would apply if you were building the barrel shifter out of ASIC cells instead of MSI parts, except you’d be counting chip area instead of MSI/SSI packages.

Typical ASIC cell libraries have 1-bit-wide multiplexers, usually realized with CMOS transmission gates, with 2 to 8 inputs. To build a larger multiplexer, you have to put together the appropriate combination of smaller cells. Besides the kind of choices we encountered in the MSI example, you have the further complication that CMOS delays are highly dependent on loading. Thus, depending on the approach, you must decide where to add buffers to the control lines, the data lines, or both to minimize loading-related delays. An approach that looks good on paper, before analyzing these delays and adding buffers, may actually turn out to have poorer delay or more chip area than another approach.