9.1 Implement 10-output (decimal) decoders using NAND gates for
   a. 2-out-of-5 code;
   b. 4-bit Gray code;
   c. 2-4-2-1 code.
   The corresponding codes are defined in Chapter 2.

9.17 Implement
   a. an eight-bit simple shifter using multiplexers;
   b. an eight-bit bidirectional 3-shifter using multiplexers.

9.22 Analyze the network shown in Figure 9.38 and design a gate network using AND, OR, XOR, and NOR gates that implements the same function. (Hint: \( z = 1 \) if the inputs to the decoder and the multiplexer are identical. Implement an equality comparator using XOR gates and one NOR.)

10.1 Compare the three implementations of the full adder given in Figure 10.3 in terms of delay and number of equivalent gates. Use the gate characteristics given in Chapter 3 and 4.

10.9 Consider a 64-bit adder. Draw the logic diagram of the network for the following implementations:
   a. A carry-ripple adder using 4-bit carry-lookahead adder modules.
   b. A carry-lookahead adder using 4-bit carry-lookahead adder modules (CLA-4) and 4-bit carry-lookahead generator modules (CLG-4).
   Compare the two implementations in terms of number of modules and delay. Use the characteristics of the family of gates given in Chapters 3 and 4.

10.12 For the following pairs of 8-bit vectors \( x \) and \( y \) representing integers in the two’s
complement systems, obtain the eight-bit vectors \( z \) and \( d \) representing \( z=x+y \) and \( d=x-y \), respectively. Perform the operation directly on the bit-vectors using the two’s-complement arithmetic unit (Figure 10.12) presented in this chapter. That is, show the values of control signals, \( c_0 \), and the bit-vectors at the output of the complementer and at the output of the adder, as well as the conditions \( o\text{vf} \), \( z\text{ero} \), and \( s\text{ign} \).

\[
\begin{array}{|c|c|}
\hline
x & y \\
\hline
01010011 & 00100111 \\
01010011 & 01000001 \\
10101010 & 10100000 \\
10101010 & 11110001 \\
10110110 & 00110011 \\
10110110 & 01100111 \\
\hline
\end{array}
\]

10.17 Design a four-bit ALU that performs one arithmetic operation (ADD) and one logic operation (NAND).

10.21 Design a 32-bit tree comparator using 4-bit modules.

10.24 Design a 8×4 binary multiplier using AND gates and four-bit adder modules. Use the AND gates to produce the partial products and the adders to add them. Determine the worst-case multiplier delay in terms of the module delays.