ELEE 2320
Chapter 2.

Chapter 2

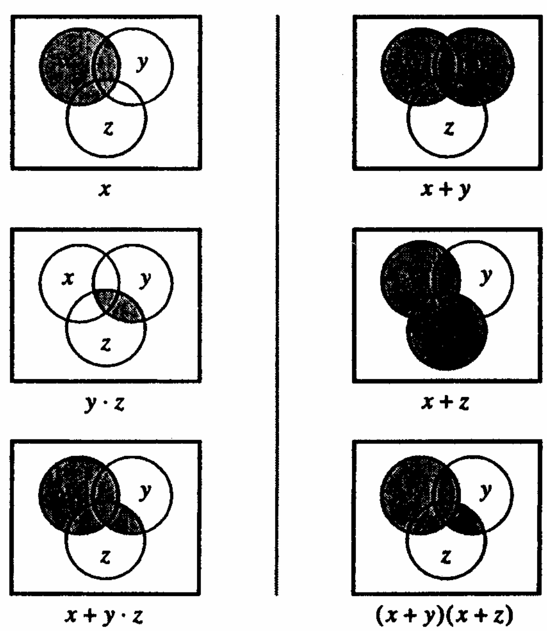
2.1. The proof is as follows:

$$
\begin{aligned}
(x+y)\cdot(x+z) &= xx + xz + xy + yz \\
&= x + xz + xy + yz \\
&= x(1 + z + y) + yz \\
&= x \cdot 1 + yz \\
&= x + yz
\end{aligned}
$$

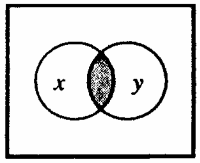
2.2. The proof is as follows:

$$
\begin{aligned}
(x+y)\cdot(x+\overline{y}) &= xx + xy + x\overline{y} + y\overline{y} \\
&= x + xy + x\overline{y} + 0 \\
&= x(1 + y + \overline{y}) \\
&= x \cdot 1 \\
&= x
\end{aligned}
$$

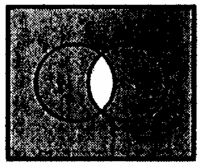
2.3. Proof using Venn diagrams:



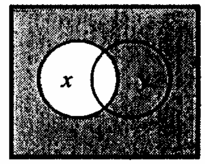
2.4. **Proof of 15*a* using Venn diagrams:**



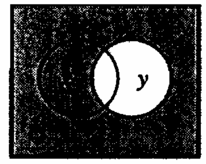
$x \cdot y$



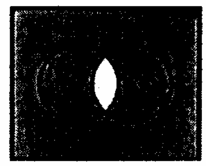
$\overline{x \cdot y}$



$\bar{x}$



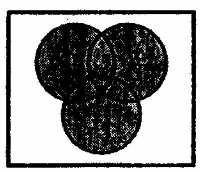
$\bar{y}$



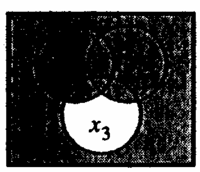
$\bar{x} \cdot \bar{y}$

A similar proof is constructed for 15*b*.

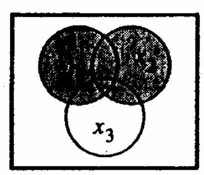
2.5. **Proof using Venn diagrams:**



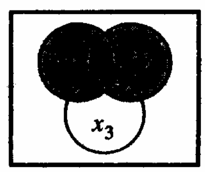
$x_1 + x_2 + x_3$



$x_1 + x_2 + \bar{x}_3$



$(x_1 + x_2 + x_3) \cdot (x_1 + x_2 + \bar{x}_3)$

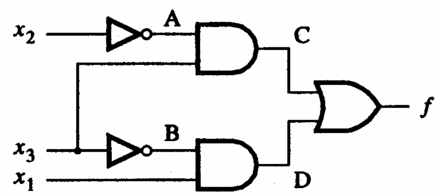


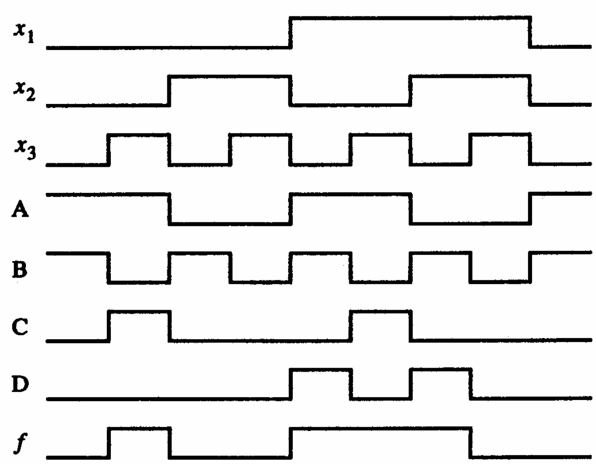
$x_1 + x_2$

2.6. A possible approach for determining whether or not the expressions are valid is to try to manipulate the left and right sides of an expression into the same form, using the theorems and properties presented in section 2.5. While this may seem simple, it is an awkward approach, because it is not obvious what target form one should try to reach. A much simpler approach is to construct a truth table for each side of an expression. If the truth tables are identical, then the expression is valid. Using this approach, we can show that the answers are:

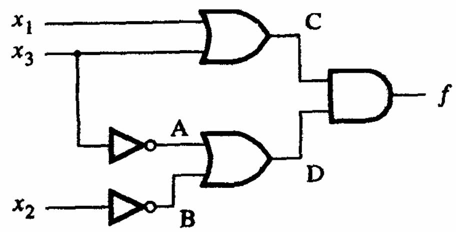
(a) Yes
(b) Yes
(c) No

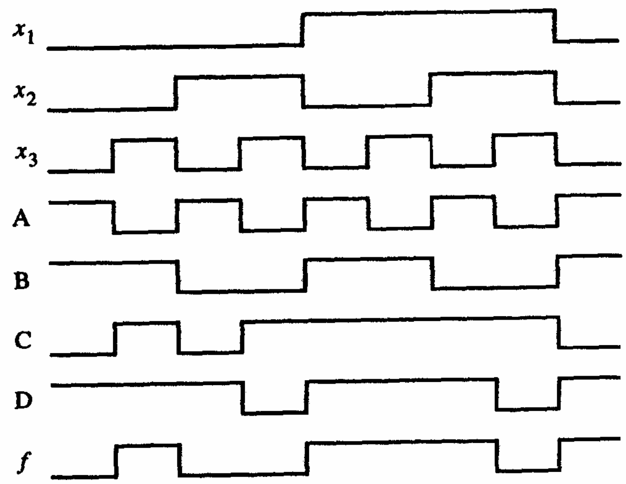
2.7. Timing diagram of the waveforms that can be observed on all wires of the circuit:





2.8. Timing diagram of the waveforms that can be observed on all wires of the circuit:





2.9. Starting with the canonical sum-of-products for f get

$$\begin{aligned}
f &= \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2\bar{x}_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3 + x_1x_2x_3 \\
&= x_1(\bar{x}_2\bar{x}_3 + \bar{x}_2x_3 + x_2\bar{x}_3 + x_2x_3) + x_2(\bar{x}_1\bar{x}_3 + \bar{x}_1x_3 + x_1\bar{x}_3 + x_1x_3) \\
&\quad + x_3(\bar{x}_1\bar{x}_2 + \bar{x}_1x_2 + x_1\bar{x}_2 + x_1x_2) \\
&= x_1(\bar{x}_2(\bar{x}_3 + x_3) + x_2(\bar{x}_3 + x_3)) + x_2(\bar{x}_1(\bar{x}_3 + x_3) + x_1(\bar{x}_3 + x_3)) \\
&\quad + x_3(\bar{x}_1(\bar{x}_2 + x_2) + x_1(\bar{x}_2 + x_2)) \\
&= x_1(\bar{x}_2 \cdot 1 + x_2 \cdot 1) + x_2(\bar{x}_1 \cdot 1 + x_1 \cdot 1) + x_3(\bar{x}_1 \cdot 1 + x_1 \cdot 1) \\
&= x_1(\bar{x}_2 + x_2) + x_2(\bar{x}_1 + x_1) + x_3(\bar{x}_1 + x_1) \\
&= x_1 \cdot 1 + x_2 \cdot 1 + x_3 \cdot 1 \\
&= x_1 + x_2 + x_3
\end{aligned}$$

2.10. The canonical product-of-sums for f is

$$\begin{aligned}
f &= (x_1 + x_2 + x_3)(x_1 + x_2 + \bar{x}_3)(x_1 + \bar{x}_2 + x_3)(x_1 + \bar{x}_2 + \bar{x}_3) \cdot \\
&\quad (\bar{x}_1 + x_2 + x_3)(\bar{x}_1 + x_2 + \bar{x}_3)(\bar{x}_1 + \bar{x}_2 + x_3)
\end{aligned}$$

It can be manipulated as follows:

$$\begin{aligned}
f &= (x_1(1 + x_2 + x_3)(1 + x_2 + \bar{x}_3)(1 + \bar{x}_2 + x_3)(1 + \bar{x}_2 + \bar{x}_3)) \cdot \\
&\quad (x_2(x_1 + 1 + x_3)(x_1 + 1 + \bar{x}_3)(\bar{x}_1 + 1 + x_3)(\bar{x}_1 + 1 + \bar{x}_3)) \cdot \\
&\quad (x_3(x_1 + x_2 + 1)(x_1 + \bar{x}_2 + 1)(\bar{x}_1 + x_2 + 1)(\bar{x}_1 + \bar{x}_2 + 1))
\end{aligned}$$

$$
\begin{aligned}
&= (x_1 \cdot 1 \cdot 1 \cdot 1 \cdot 1)(x_2 \cdot 1 \cdot 1 \cdot 1 \cdot 1)(x_3 \cdot 1 \cdot 1 \cdot 1 \cdot 1) \\
&= x_1 x_2 x_3
\end{aligned}
$$

2.11. Derivation of the minimum sum-of-products expression:

$$
\begin{aligned}
f &= x_1x_3 + x_1\overline{x}_2 + \overline{x}_1x_2x_3 + \overline{x}_1\overline{x}_2\overline{x}_3 \\
&= x_1(\overline{x}_2 + x_2)x_3 + x_1\overline{x}_2(\overline{x}_3 + x_3) + \overline{x}_1x_2x_3 + \overline{x}_1\overline{x}_2\overline{x}_3 \\
&= x_1\overline{x}_2x_3 + x_1x_2x_3 + x_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2x_3 + \overline{x}_1\overline{x}_2\overline{x}_3 \\
&= x_1x_3 + (x_1 + \overline{x}_1)x_2x_3 + (x_1 + \overline{x}_1)\overline{x}_2\overline{x}_3 \\
&= x_1x_3 + x_2x_3 + \overline{x}_2\overline{x}_3
\end{aligned}
$$

2.12. Derivation of the minimum sum-of-products expression:

$$
\begin{aligned}
f &= x_1\overline{x}_2\overline{x}_3 + x_1x_2x_4 + x_1\overline{x}_2x_3\overline{x}_4 \\
&= x_1\overline{x}_2\overline{x}_3(\overline{x}_4 + x_4) + x_1x_2x_4 + x_1\overline{x}_2x_3\overline{x}_4 \\
&= x_1\overline{x}_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3x_4 + x_1x_2x_4 + x_1\overline{x}_2x_3\overline{x}_4 \\
&= x_1\overline{x}_2\overline{x}_3 + x_1\overline{x}_2(\overline{x}_3 + x_3)\overline{x}_4 + x_1x_2x_4 \\
&= x_1\overline{x}_2\overline{x}_3 + x_1\overline{x}_2\overline{x}_4 + x_1x_2x_4
\end{aligned}
$$

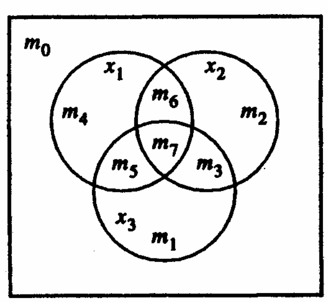
2.13. The simplest POS expression is derived as

$$
\begin{aligned}
f &= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + \overline{x}_3 + x_4) \\
&= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + x_3 + x_4)(x_1 + \overline{x}_2 + \overline{x}_3 + x_4) \\
&= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)((x_1 + \overline{x}_2 + x_4)(x_3 + \overline{x}_3)) \\
&= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + x_4) \cdot 1 \\
&= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + x_4)
\end{aligned}
$$

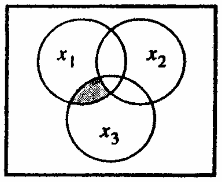
2.14. Derivation of the minimum product-of-sums expression:

$$
\begin{aligned}
f &= (x_1 + x_2 + x_3)(x_1 + \overline{x}_2 + x_3)(\overline{x}_1 + \overline{x}_2 + x_3)(x_1 + x_2 + \overline{x}_3) \\
&= ((x_1 + x_2) + x_3)((x_1 + x_2) + \overline{x}_3)(x_1 + (\overline{x}_2 + x_3))(\overline{x}_1 + (\overline{x}_2 + x_3)) \\
&= (x_1 + x_2)(\overline{x}_2 + x_3)
\end{aligned}
$$

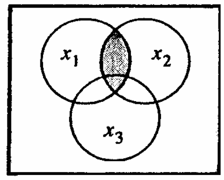
2.15. (*a*) Location of all minterms in a 3-variable Venn diagram:



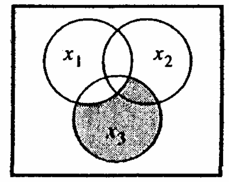
(b) For $f = x_1\overline{x}_2x_3 + x_1x_2 + \overline{x}_1x_3$ have:



$x_1 \cdot \bar{x}_2 \cdot x_3$

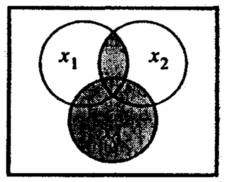


$x_1 \cdot x_2$



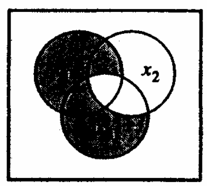
$\bar{x}_1 \cdot x_3$

Therefore, f is represented as:



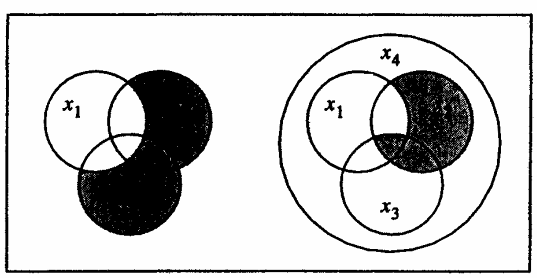
$f = x_3 + x_1x_2$

2.16. The function in Figure 2.18 in Venn diagram form is:



2.17. In Figure P2.1a it is possible to represent only 14 minterms. It is impossible to represent the minterms $\overline{x}_1\overline{x}_2x_3x_4$ and $x_1x_2\overline{x}_3\overline{x}_4$.
In Figure P2.1b, it is impossible to represent the minterms $x_1x_2\overline{x}_3\overline{x}_4$ and $x_1x_2x_3\overline{x}_4$.

2.18. Venn diagram for $f = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + x_1x_2x_3x_4 + \overline{x}_1x_2$ is



2.19. The simplest SOP implementation of the function is

$$\begin{aligned} f &= \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 \overline{x}_3 + x_1 x_2 \overline{x}_3 + x_1 x_2 x_3 \\ &= (\overline{x}_1 + x_1) x_2 x_3 + x_1 (\overline{x}_2 + x_2) \overline{x}_3 \\ &= x_2 x_3 + x_1 \overline{x}_3 \end{aligned}$$

2.20. The simplest SOP implementation of the function is

$$\begin{aligned} f &= \overline{x}_1 \overline{x}_2 x_3 + \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 \overline{x}_3 + x_1 x_2 \overline{x}_3 + x_1 x_2 x_3 \\ &= \overline{x}_1 (\overline{x}_2 + x_2) x_3 + x_1 (\overline{x}_2 + x_2) \overline{x}_3 + (\overline{x}_1 + x_1) x_2 x_3 \\ &= \overline{x}_1 x_3 + x_1 \overline{x}_3 + x_2 x_3 \end{aligned}$$

Another possibility is

$$f = \overline{x}_1 x_3 + x_1 \overline{x}_3 + x_1 x_2$$

2.21. The simplest POS implementation of the function is

$$\begin{aligned} f &= (x_1 + x_2 + x_3)(x_1 + \overline{x}_2 + x_3)(\overline{x}_1 + x_2 + \overline{x}_3) \\ &= ((x_1 + x_3) + x_2)((x_1 + x_3) + \overline{x}_2)(\overline{x}_1 + x_2 + \overline{x}_3) \\ &= (x_1 + x_3)(\overline{x}_1 + x_2 + \overline{x}_3) \end{aligned}$$

2.22. The simplest POS implementation of the function is

$$\begin{aligned} f &= (x_1 + x_2 + x_3)(x_1 + x_2 + \overline{x}_3)(\overline{x}_1 + x_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3) \\ &= ((x_1 + x_2) + x_3)((x_1 + x_2) + \overline{x}_3)((\overline{x}_1 + x_3) + x_2)((\overline{x}_1 + x_3) + \overline{x}_2) \\ &= (x_1 + x_2)(\overline{x}_1 + \overline{x}_3) \end{aligned}$$

2.23. The lowest cost circuit is defined by

$$f(x_1, x_2, x_3) = x_1 x_2 + x_1 x_3 + x_2 x_3$$

2.24. The truth table that corresponds to the timing diagram in Figure P2.3 is

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The simplest SOP expression is $f = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1\overline{x}_2 x_3 + x_1 x_2\overline{x}_3$.

2.25. The truth table that corresponds to the timing diagram in Figure P2.4 is

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The simplest SOP expression is derived as follows:

$$\begin{aligned}
f &= \overline{x}_1\overline{x}_2 x_3 + \overline{x}_1 x_2\overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1\overline{x}_2\overline{x}_3 + x_1 x_2 x_3 \\
&= \overline{x}_1(\overline{x}_2 + x_2)x_3 + \overline{x}_1\overline{x}_2(\overline{x}_3 + x_3) + (\overline{x}_1 + x_1)x_2 x_3 + x_1\overline{x}_2\overline{x}_3 \\
&= \overline{x}_1 \cdot 1 \cdot x_3 + \overline{x}_1 x_2 \cdot 1 + 1 \cdot x_2 x_3 + x_1\overline{x}_2\overline{x}_3 \\
&= \overline{x}_1 x_3 + \overline{x}_1 x_2 + x_2 x_3 + x_1\overline{x}_2\overline{x}_3
\end{aligned}$$

2.26. (*a*)

x_1	x_0	y_1	y_0	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

(*b*) $f = (x_1 + \overline{y}_1)(\overline{x}_1 + y_1)(x_0 + \overline{y}_0)(\overline{x}_0 + y_0)$

2.27. (*a*)

x_1	x_0	y_1	y_0	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

(*b*) The canonical SOP expression is

$$\begin{aligned} f &= \overline{x}_1\overline{x}_0\overline{y}_1\overline{y}_0 + \overline{x}_1x_0\overline{y}_1\overline{y}_0 + \overline{x}_1x_0\overline{y}_1y_0 + x_1\overline{x}_0\overline{y}_1\overline{y}_0 + x_1\overline{x}_0\overline{y}_1y_0 + x_1\overline{x}_0y_1\overline{y}_0 \\ &\quad + x_1x_0\overline{y}_1\overline{y}_0 + x_1x_0\overline{y}_1y_0 + x_1x_0y_1\overline{y}_0 + x_1x_0y_1y_0 \end{aligned}$$

(c) The simplest SOP expression is

$$f = x_1 x_0 + \overline{y}_1 \overline{y}_0 + x_1 \overline{y}_0 + x_0 \overline{y}_1$$

2.30.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob2_30 IS
    PORT ( x1, x2, x3, x4  : IN     STD_LOGIC ;
               f1, f2           : OUT  STD_LOGIC ) ;
END prob2_30 ;

ARCHITECTURE LogicFunc OF prob2_30 IS
BEGIN
    f1 <= (x1 AND NOT x3) OR (x2 AND NOT x3) OR
           NOT x3 AND NOT x4) OR (x1 AND x2) OR
           x1 AND NOT x4) ;
    f2 <= (x1 OR NOT x3) AND (x1 OR x2 OR NOT x4) AND
           x2 OR NOT x3 OR NOT x4) ;
END LogicFunc ;
```

2.31. For the functions given in this question, it is not true that $f_1 = \overline{f}_2$. The function f_1 is given in the form (SOP-term) AND (SOP-term). If these same two SOP terms are used for the *different* function f_1 = (SOP-term) OR (SOP-term) then for this new f_1 it is true that $f_1 = \overline{f}_2$. Complete VHDL code using this new function f_1 is shown below.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob2_31 IS
    PORT ( x1, x2, x3, x4  : IN     STD_LOGIC ;
               f1, f2           : OUT  STD_LOGIC ) ;
END prob2_31 ;

ARCHITECTURE LogicFunc OF prob2_31 IS
BEGIN
    f1 <= ((x1 AND x3) OR (NOT x1 AND NOT x3)) OR
           ((x2 AND x4) OR (NOT x2 AND NOT x4)) ;
    f2 <= (x1 AND x2 AND NOT x3 AND NOT x4) OR
           (NOT x1 AND NOT x2 AND x3 AND x4) OR
           (x1 AND NOT x2 AND NOT x3 AND x4) OR
           (NOT x1 AND x2 AND x3 AND NOT x4);
END LogicFunc ;
```

Thank you
Abe Aounallah.

Chapter 3

3.1. (*a*)

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

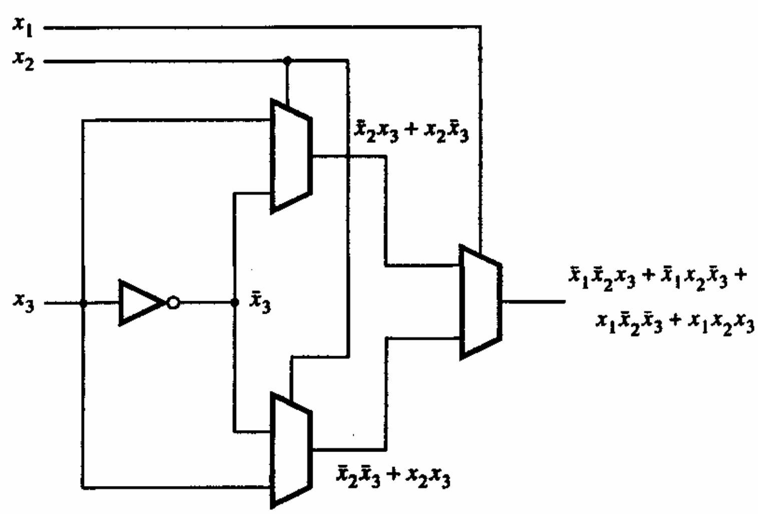
(*b*)

$$\begin{aligned}\#\text{transistors} &= \text{NOT\_gates} \times 2 + \text{AND\_gates} \times 8 + \text{OR\_gates}\\ &= 3 \times 2 + 4 \times 8 + 1 \times 10 = 48\end{aligned}$$

3.2. (*a*) In problem 3.1 the canonical SOP for f is

$$f = \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2\overline{x}_3 + x_1\overline{x}_2\overline{x}_3 + x_1x_2x_3$$

This expression is equivalent to f in Figure P3.2, as derived below.



(*b*) Assuming the multiplexers are implemented using transmission gates

$$\begin{aligned}\#\text{transistors} &= \text{NOT\_gates} \times 2 + \text{MUXes} \times 6\\ &= 1 \times 2 + 3 \times 6 = 20\end{aligned}$$

3.3. (*a*) A SOP expression for f in Figure P3.3 is:

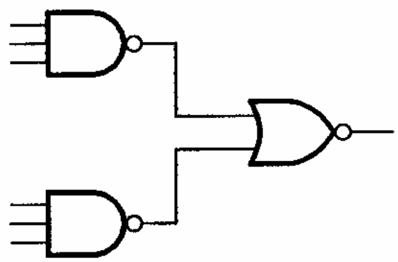
$$\begin{aligned} f &= (x_1 \oplus x_2) \oplus x_3 \\ &= (x_1 \oplus x_2)\overline{x}_3 + \overline{(x_1 \oplus x_2)}x_3 \\ &= x_1\overline{x}_2\overline{x}_3 + \overline{x}_1 x_2 \overline{x}_3 + \overline{x}_1\overline{x}_2 x_3 + x_1 x_2 x_3 \end{aligned}$$

which is equivalent to the expression derived in problem 3.2.

(b) Assuming the XOR gates are implemented as shown in Figure 3.61*b*

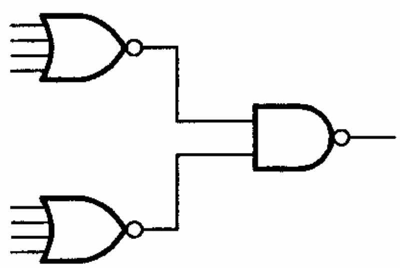
$$\begin{aligned} \#\text{transistors} &= \text{XOR\_gates} \times 8 \\ &= 2 \times 8 = 16 \end{aligned}$$

3.4. Using the circuit



The number of transistors needed is 16.

3.5. Using the circuit



The number of transistors needed is 20.

3.6. (*a*)

x_1	x_2	x_3	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

(*b*) The canonical SOP expression is

$$f = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2\overline{x}_3 + \overline{x}_1x_2x_3 + x_1\overline{x}_2\overline{x}_3$$

The number of transistors required using only AND, OR, and NOT gates is

$$\begin{aligned}\#\text{transistors} &= \text{NOT\_gates} \times 2 + \text{AND\_gates} \times 8 + \text{OR\_gates} \times 12 \\ &= 3 \times 2 + 5 \times 8 + 1 \times 12 = 58\end{aligned}$$

3.7. (*a*)

x_1	x_2	x_3	x_4	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0

x_1	x_2	x_3	x_4	f
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

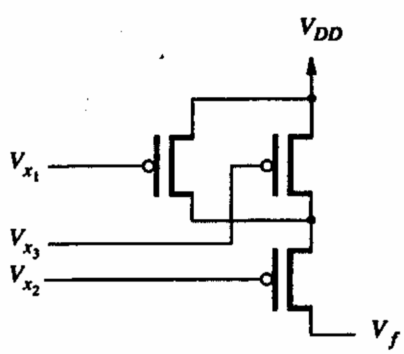
(*b*)

$$\begin{aligned}f &= \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3\overline{x}_4 \\ &= \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_2\overline{x}_3\overline{x}_4\end{aligned}$$

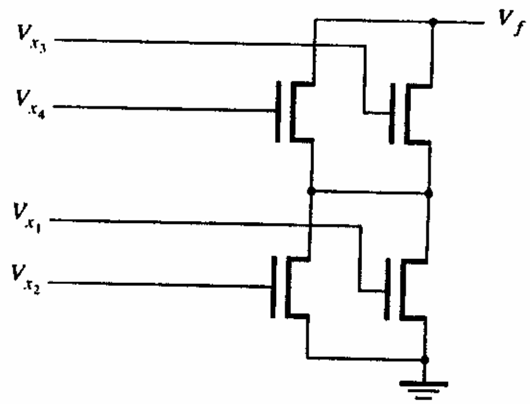
The number of transistors required using only AND, OR, and NOT gates is

$$\begin{aligned}\#\text{transistors} &= \text{NOT\_gates} \times 2 + \text{AND\_gates} \times 8 + \text{OR\_gates} \times 4 \\ &= 4 \times 2 + 2 \times 8 + 1 \times 4 = 28\end{aligned}$$

3.8.



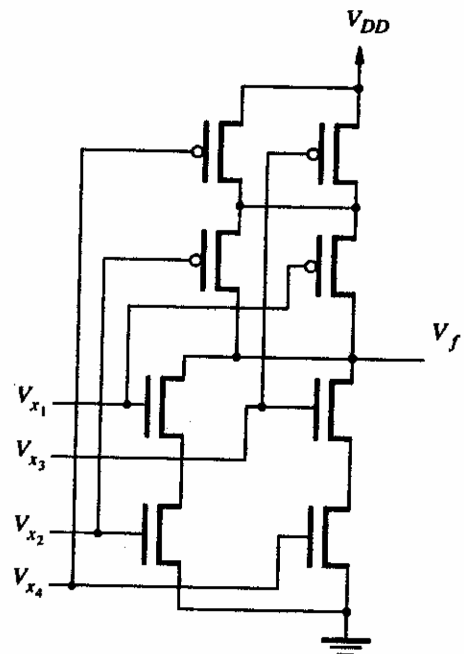
3.9.



3.10. Minimum SOP expression for f is

$$
\begin{aligned}
f &= \overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_3 + \overline{x}_2\overline{x}_4 + \overline{x}_1\overline{x}_4 \\
&= (\overline{x}_1 + \overline{x}_2)(\overline{x}_3 + \overline{x}_4)
\end{aligned}
$$

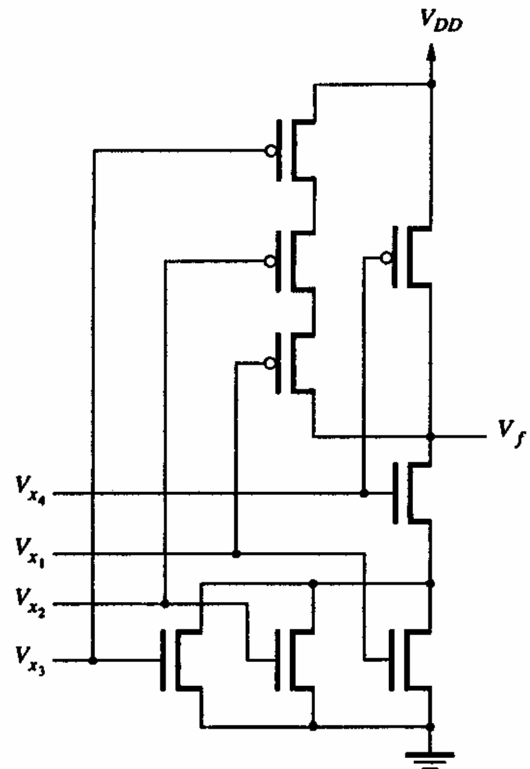
which leads to the circuit



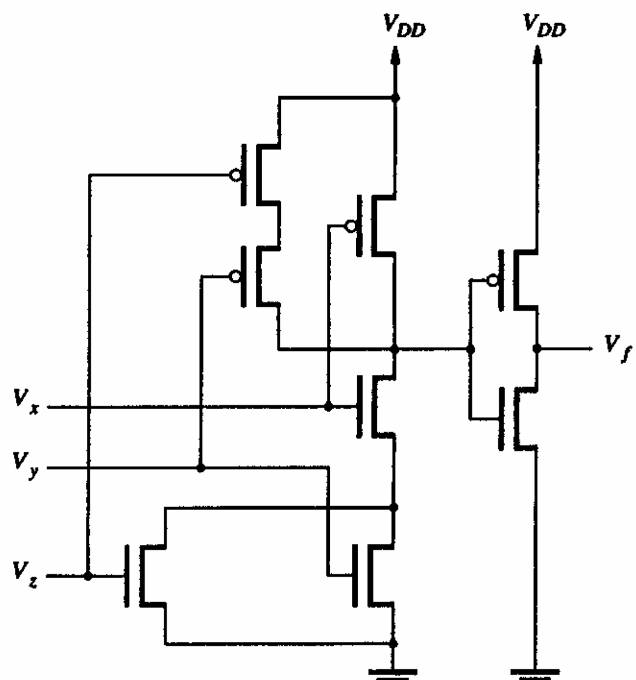
3.11. Minimum SOP expression for f is

$$
f = \overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3
$$

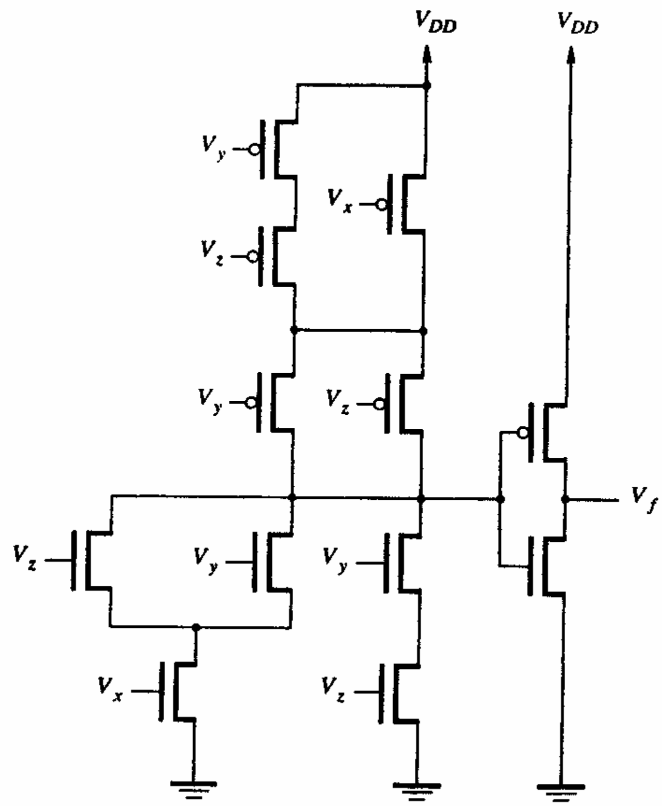
which leads to the circuit



3.12.



3.13.



3.14. (*a*) Since $V_{DS} \geq V_{GS} - V_T$ the NMOS transistor is operating in the saturation region:

$$\begin{aligned} I_D &= \frac{1}{2}k_n'\frac{W}{L}(V_{GS} - V_T)^2 \\ &= 10\,\frac{\mu\text{A}}{\text{V}^2} \times 5 \times (5\text{ V} - 1\text{ V})^2 = 800\,\mu\text{A} \end{aligned}$$

(*b*) In this case $V_{DS} < V_{GS} - V_T$, thus the NMOS transistor is operating in the triode region:

$$\begin{aligned} I_D &= k_n'\frac{W}{L}\left[(V_{GS} - V_T)V_{DS} - \frac{1}{2}V_{DS}^2\right] \\ &= 20\,\frac{\mu\text{A}}{\text{V}^2} \times 5 \times \left[(5\text{ V} - 1\text{ V}) \times 0.2\text{ V} - \frac{1}{2} \times (0.2\text{ V})^2\right] = 78\,\mu\text{A} \end{aligned}$$

3.15. (*a*) Since $V_{DS} \leq V_{GS} - V_T$ the PMOS transistor is operating in the saturation region:

$$\begin{aligned} I_D &= \frac{1}{2}k_p'\frac{W}{L}(V_{GS} - V_T)^2 \\ &= 5\,\frac{\mu\text{A}}{\text{V}^2} \times 5 \times (-5\text{ V} + 1\text{ V})^2 = 400\,\mu\text{A} \end{aligned}$$

(*b*) In this case $V_{DS} > V_{GS} - V_T$, thus the PMOS transistor is operating in the triode region:

$$\begin{aligned} I_D &= k_p'\frac{W}{L}\left[(V_{GS} - V_T)V_{DS} - \frac{1}{2}V_{DS}^2\right] \\ &= 10\,\frac{\mu\text{A}}{\text{V}^2} \times 5 \times \left[(-5\text{ V} + 1\text{ V}) \times (-0.2)\text{ V} - \frac{1}{2} \times (-0.2\text{ V})^2\right] = 39\,\mu\text{A} \end{aligned}$$

3.16.

$$\begin{aligned} R_{DS} &= 1/\left[k_n'\frac{W}{L}(V_{GS}-V_T)\right] \\ &= 1/\left[0.020\,\frac{\text{mA}}{\text{V}^2}\times 10\times(5\,\text{V}-1\,\text{V})\right] = 1.25\,\text{k}\Omega \end{aligned}$$

3.17.

$$\begin{aligned} R_{DS} &= 1/\left[k_n'\frac{W}{L}(V_{GS}-V_T)\right] \\ &= 1/\left[0.040\,\frac{\text{mA}}{\text{V}^2}\times 10\times(3.3\,\text{V}-0.66\,\text{V})\right] = 947\,\Omega \end{aligned}$$

3.18. Since $V_{DS} < (V_{GS} - V_T)$, the PMOS transistor is operating in the saturation region:

$$\begin{aligned} I_{SD} &= \frac{1}{2}k_p'\frac{W}{L}(V_{GS}-V_T)^2 \\ &= 50\,\frac{\mu\text{A}}{\text{V}^2}\times(-5\,\text{V}+1\,\text{V})^2 = 800\,\mu\text{A} \end{aligned}$$

Hence the value of R_{DS} is

$$\begin{aligned} R_{DS} &= V_{DS}/I_{DS} \\ &= 4.8\,\text{V}/800\,\mu\text{A} = 6\,\text{k}\Omega \end{aligned}$$

3.19. Since $V_{DS} < (V_{GS} - V_T)$, the PMOS transistor is operating in the saturation region:

$$\begin{aligned} I_{SD} &= \frac{1}{2}k_p'\frac{W}{L}(V_{GS}-V_T)^2 \\ &= 80\,\frac{\mu\text{A}}{\text{V}^2}\times(-3.3\,\text{V}+0.66\,\text{V})^2 = 558\,\mu\text{A} \end{aligned}$$

Hence the value of R_{DS} is

$$\begin{aligned} R_{DS} &= V_{DS}/I_{DS} \\ &= 3.2\,\text{V}/558\,\mu\text{A} = 5.7\,\text{k}\Omega \end{aligned}$$

3.20. The low output voltage of the pseudo-NMOS inverter can be obtained by setting $V_x = V_{DD}$ and evaluating the voltage V_f. First we assume that the NMOS transistor is operating in the triode region while the PMOS is operating in the saturation region. For simplicity we will assume that the magnitude of the threshold voltages for both the NMOS and PMOS transistors are equal, so that

$$V_T = V_{TN} = -V_{TP}$$

The current flowing through the PMOS transistor is

$$\begin{aligned} I_D &= \frac{1}{2}k_p'\frac{W_p}{L_p}(-V_{DD}-V_{TP})^2 \\ &= \frac{1}{2}k_p(-V_{DD}-V_{TP})^2 \\ &= \frac{1}{2}k_p(V_{DD}-V_T)^2 \end{aligned}$$

Similarly, the current going through the NMOS transistor is

$$
\begin{aligned}
I_D &= k_n' \frac{W_n}{L_n}\left[(V_x - V_{TN})V_f - \frac{1}{2}V_f^2\right] \\
&= k_n \left[(V_x - V_{TN})V_f - \frac{1}{2}V_f^2\right] \\
&= k_n \left[(V_{DD} - V_T)V_f - \frac{1}{2}V_f^2\right]
\end{aligned}
$$

Since there is only one path for current to flow, we can equate the currents flowing through the NMOS and PMOS transistors and solve for the voltage V_f.

$$
k_p(V_{DD} - V_T)^2 = 2k_n\left[(V_{DD} - V_T)V_f - \frac{1}{2}V_f^2\right]
$$

$$
k_p(V_{DD} - V_T)^2 - 2k_n(V_{DD} - V_T)V_f + k_n V_f^2 = 0
$$

This quadratic equation can be solved using the standard formula, with the parameters

$$
a = k_n,\ b = -2k_n(V_{DD} - V_T),\ c = k_p(V_{DD} - V_T)^2
$$

which gives

$$
\begin{aligned}
V_f &= \frac{-b}{2a} \pm \sqrt{\frac{b^2}{4a^2} - \frac{c}{a}} \\
&= (V_{DD} - V_T) \pm \sqrt{(V_{DD} - V_T)^2 - \frac{k_p}{k_n}(V_{DD} - V_T)^2} \\
&= (V_{DD} - V_T)\left[1 \pm \sqrt{1 - \frac{k_p}{k_n}}\right]
\end{aligned}
$$

Only one of these two solutions is valid, because we started with the assumption that the NMOS transistor is in the triode region while the PMOS is in the saturation region. Thus

$$
V_f = (V_{DD} - V_T)\left[1 - \sqrt{1 - \frac{k_p}{k_n}}\right]
$$

3.21. (*a*)

$$
\begin{aligned}
I_{stat} &= \frac{1}{2}k_p' \frac{W_p}{L_p}(V_{DD} - V_T)^2 \\
&= 12\,\frac{\mu\text{A}}{\text{V}^2} \times 1 \times (5\,\text{V} - 1\,\text{V})^2 = 192\,\mu\text{A}
\end{aligned}
$$

(*b*)

$$
\begin{aligned}
R_{DS} &= 1/\left[k_n' \frac{W_n}{L_n}(V_{GS} - V_T)\right] \\
&= 1/\left[0.060\,\frac{\text{mA}}{\text{V}^2} \times 4 \times (5\,\text{V} - 1\,\text{V})\right] = 1.04\,\text{k}\Omega
\end{aligned}
$$

(*c*) Using the expression derived in problem 3.20

$$
k_p = k_p' \frac{W_p}{L_p} = 24\,\frac{\mu\text{A}}{\text{V}^2}
$$

$$
k_n = k_n' \frac{W_n}{L_n} = 240\,\frac{\mu\text{A}}{\text{V}^2}
$$

$$\begin{aligned} V_{OL} = V_f &= (5\text{ V} - 1\text{ V})\left[1 - \sqrt{1 - \frac{24}{240}}\right] \\ &= 0.21\text{ V} \end{aligned}$$

(*d*)

$$\begin{aligned} P_D &= I_{stat} V_{DD} \\ &= 192\,\mu\text{A} \times 5\text{ V} = 960\,\mu\text{W} \approx 1\text{m}W \end{aligned}$$

(*e*)

$$\begin{aligned} R_{SDP} &= V_{SD}/I_{SD} \\ &= (V_{DD} - V_f)/I_{stat} \\ &= (5\text{ V} - 0.21\text{ V})/0.192\,\text{mA} = 24.9\,\text{k}\Omega \end{aligned}$$

(*f*) The low-to-high propagation delay is

$$\begin{aligned} t_{p_{LH}} &= \frac{1.7C}{k_p' \frac{W_p}{L_p} V_{DD}} \\ &= \frac{1.7 \times 70\text{ fF}}{24\,\frac{\mu\text{A}}{\text{V}^2} \times 1 \times 5\text{ V}} = 0.99\text{ ns} \end{aligned}$$

The high-to-low propagation delay is

$$\begin{aligned} t_{p_{HL}} &= \frac{1.7C}{k_n' \frac{W_n}{L_n} V_{DD}} \\ &= \frac{1.7 \times 70\text{ fF}}{60\,\frac{\mu\text{A}}{\text{V}^2} \times 4 \times 5\text{ V}} = 0.1\text{ ns} \end{aligned}$$

3.22. (*a*)

$$\begin{aligned} I_{stat} &= \frac{1}{2} k_p' \frac{W_p}{L_p} (V_{DD} - V_T)^2 \\ &= 48\,\frac{\mu\text{A}}{\text{V}^2} \times 1 \times (5\text{ V} - 1\text{ V})^2 = 768\,\mu\text{A} \end{aligned}$$

(*b*)

$$\begin{aligned} R_{DS} &= 1/\left[k_n' \frac{W_n}{L_n}(V_{GS} - V_T)\right] \\ &= 1/\left[0.060\,\frac{\text{mA}}{\text{V}^2} \times 4 \times (5\text{ V} - 1\text{ V})\right] = 1.04\,\text{k}\Omega \end{aligned}$$

(*c*) Using the expression derived in problem 3.20

$$k_p = k_p' \frac{W_p}{L_p} = 96\,\frac{\mu\text{A}}{\text{V}^2} k_n = k_n' \frac{W_n}{L_n} = 240\,\frac{\mu\text{A}}{\text{V}^2}$$

$$V_{OL} = V_f = (5\,\mathrm{V} - 1\,\mathrm{V})\left[1 - \sqrt{1 - \frac{96}{240}}\right] = 0.90\,\mathrm{V}$$

(*d*)

$$\begin{aligned} P_D &= I_{stat} V_{DD} \\ &= 768\,\mu\mathrm{A} \times 5\,\mathrm{V} = 3840\,\mu\mathrm{W} \approx 3.8\mathrm{m}W \end{aligned}$$

(*e*)

$$\begin{aligned} R_{SDP} &= V_{SD}/I_{SD} \\ &= (V_{DD} - V_f)/I_{stat} \\ &= (5\,\mathrm{V} - 0.90\,\mathrm{V})/0.768\,\mathrm{mA} = 5.34\,\mathrm{k}\Omega \end{aligned}$$

(*f*) The low-to-high propagation delay is

$$\begin{aligned} t_{PLH} &= \frac{1.7C}{k_p' \frac{W_p}{L_p} V_{DD}} \\ &= \frac{1.7 \times 70\,\mathrm{fF}}{96\,\frac{\mu\mathrm{A}}{\mathrm{V}^2} \times 1 \times 5\,\mathrm{V}} = 0.25\,\mathrm{ns} \end{aligned}$$

The high-to-low propagation delay is

$$\begin{aligned} t_{PHL} &= \frac{1.7C}{k_n' \frac{W_n}{L_n} V_{DD}} \\ &= \frac{1.7 \times 70\,\mathrm{fF}}{60\,\frac{\mu\mathrm{A}}{\mathrm{V}^2} \times 4 \times 5\,\mathrm{V}} = 0.1\,\mathrm{ns} \end{aligned}$$

3.23. (*a*)

$$\begin{aligned} I_{stat} &= \frac{1}{2} k_p' \frac{W_p}{L_p} (V_{DD} - V_T)^2 \\ &= 12\,\frac{\mu\mathrm{A}}{\mathrm{V}^2} \times 1 \times (5\,\mathrm{V} - 1\,\mathrm{V})^2 = 192\,\mu\mathrm{A} \end{aligned}$$

(*b*) The two NMOS transistors in series can be considered equivalent to a single transistor with twice the length. Thus

$$\begin{aligned} R_{DS} &= 1/\left[k_n' \frac{W_n}{L_n} (V_{GS} - V_T)\right] \\ &= 1/\left[0.060\,\frac{\mathrm{mA}}{\mathrm{V}^2} \times 2 \times (5\,\mathrm{V} - 1\,\mathrm{V})\right] = 2.08\,\mathrm{k}\Omega \end{aligned}$$

(*c*) Using the expression derived in problem 3.20

$$k_p = k_p' \frac{W_p}{L_p} = 24\,\frac{\mu\mathrm{A}}{\mathrm{V}^2}$$

$$k_n = k_n' \frac{W_n}{L_n} = 120\,\frac{\mu\mathrm{A}}{\mathrm{V}^2}$$

$$V_{OL} = V_f = (5\,\mathrm{V} - 1\,\mathrm{V})\left[1 - \sqrt{1 - \frac{24}{120}}\right] = 0.42\,\mathrm{V}$$

(*d*)

$$\begin{aligned} P_D &= I_{stat}V_{DD} \\ &= 192\,\mu\mathrm{A} \times 5\,\mathrm{V} = 960\,\mu\mathrm{W} \approx 1\mathrm{m}W \end{aligned}$$

(*e*)

$$\begin{aligned} R_{SDP} &= V_{SD}/I_{SD} \\ &= (V_{DD} - V_f)/I_{stat} \\ &= (5\,\mathrm{V} - 0.42\,\mathrm{V})/0.192\,\mathrm{mA} = 23.9\,\mathrm{k\Omega} \end{aligned}$$

(*f*) The low-to-high propagation delay is

$$\begin{aligned} t_{PLH} &= \frac{1.7C}{k_p' \frac{W_p}{L_p} V_{DD}} \\ &= \frac{1.7 \times 70\,\mathrm{fF}}{24\,\frac{\mu\mathrm{A}}{\mathrm{V}^2} \times 1 \times 5\,\mathrm{V}} = 0.99\,\mathrm{ns} \end{aligned}$$

The high-to-low propagation delay is

$$\begin{aligned} t_{PHL} &= \frac{1.7C}{k_n' \frac{W_n}{L_n} V_{DD}} \\ &= \frac{1.7 \times 70\,\mathrm{fF}}{60\,\frac{\mu\mathrm{A}}{\mathrm{V}^2} \times 2 \times 5\,\mathrm{V}} = 0.2\,\mathrm{ns} \end{aligned}$$

3.24. (*a*)

$$\begin{aligned} I_{stat} &= \frac{1}{2}k_p' \frac{W_p}{L_p}(V_{DD} - V_T)^2 \\ &= 12\,\frac{\mu\mathrm{A}}{\mathrm{V}^2} \times 1 \times (5\,\mathrm{V} - 1\,\mathrm{V})^2 = 192\,\mu\mathrm{A} \end{aligned}$$

(*b*) The two NMOS transistors in parallel can be considered equivalent to a single transistor with twice the width. Thus

$$\begin{aligned} R_{DS} &= 1/\left[k_n' \frac{W_n}{L_n}(V_{GS} - V_T)\right] \\ &= 1/\left[0.060\,\frac{\mathrm{mA}}{\mathrm{V}^2} \times 8 \times (5\,\mathrm{V} - 1\,\mathrm{V})\right] = 520\,\Omega \end{aligned}$$

(*c*) Using the expression derived in problem 3.20

$$k_p = k_p' \frac{W_p}{L_p} = 24\,\frac{\mu\mathrm{A}}{\mathrm{V}^2}$$

$$k_n = k_n' \frac{W_n}{L_n} = 480\,\frac{\mu\mathrm{A}}{\mathrm{V}^2}$$

$$
\begin{aligned}
V_{OL} = V_f &= (5\,\text{V} - 1\,\text{V})\left[1 - \sqrt{1 - \frac{24}{480}}\right] \\
&= 0.10\,\text{V}
\end{aligned}
$$

(*d*)

$$
\begin{aligned}
P_D &= I_{stat} V_{DD} \\
&= 192\,\mu\text{A} \times 5\,\text{V} = 960\,\mu\text{W} \approx 1\text{m}W
\end{aligned}
$$

(*e*)

$$
\begin{aligned}
R_{SDP} &= V_{SD}/I_{SD} \\
&= (V_{DD} - V_f)/I_{stat} \\
&= (5\,\text{V} - 0.10\,\text{V})/0.192\,\text{mA} = 25.5\,\text{k}\Omega
\end{aligned}
$$

(*f*) The low-to-high propagation delay is

$$
\begin{aligned}
t_{PLH} &= \frac{1.7C}{k_p' \frac{W_p}{L_p} V_{DD}} \\
&= \frac{1.7 \times 70\,\text{fF}}{24\,\frac{\mu\text{A}}{\text{V}^2} \times 1 \times 5\,\text{V}} = 0.99\,\text{ns}
\end{aligned}
$$

The high-to-low propagation delay is

$$
\begin{aligned}
t_{PHL} &= \frac{1.7C}{k_n' \frac{W_n}{L_n} V_{DD}} \\
&= \frac{1.7 \times 70\,\text{fF}}{60\,\frac{\mu\text{A}}{\text{V}^2} \times 8 \times 5\,\text{V}} = 0.05\,\text{ns}
\end{aligned}
$$

3.25. (*a*)

$$
\begin{aligned}
NM_H = V_{OH} - V_{IH} = 0.5\,\text{V} \\
NM_L = V_{IL} - V_{OL} = 0.7\,\text{V}
\end{aligned}
$$

(*b*)

$$
\begin{aligned}
V_{OL} = 8 \times 0.1\,\text{V} = 0.8\,\text{V} \\
NM_L = 1\,\text{V} - 0.8\,\text{V} = 0.2\,\text{V}
\end{aligned}
$$

3.26. Under steady-state conditions, for an n-input CMOS NAND gate the voltage levels V_{OL} and V_{OH} are 0 V and V_{DD}, respectively. No current flows in a CMOS gate in the steady-state. Thus there can be no voltage drop across any of the transistors.

3.27. (*a*)

$$
\begin{aligned}
P_{NOT\_gate} &= fCV^2 \\
&= 75\,\text{MHz} \times 150\,\text{fF} \times (5\,\text{V})^2 = 281\,\mu\text{W}
\end{aligned}
$$

(*b*)

$$
P_{total} = 0.2 \times 250{,}000 \times 281\,\mu\text{W} = 14\,\text{W}
$$

3.28. (*a*)

$$
\begin{aligned}
P_{NOT\_gate} &= fCV^2 \\
&= 125\,\text{MHz} \times 120\,\text{fF} \times (3.3\,\text{V})^2 = 163\,\mu\text{W}
\end{aligned}
$$

(*b*)

$$P_{total} = 0.2 \times 250{,}000 \times 163\,\mu\text{W} = 8.2\,\text{W}$$

3.29. (*a*) The high-to-low propagation delay is

$$t_{PHL} = \frac{1.7C}{k_n' \frac{W_n}{L_n} V_{DD}} = \frac{1.7 \times 150\,\text{fF}}{20\,\frac{\mu\text{A}}{\text{V}^2} \times 10 \times 5\,\text{V}} = 0.255\,\text{ns}$$

(*b*) The low-to-high propagation delay is

$$t_{PLH} = \frac{1.7C}{k_p' \frac{W_p}{L_p} V_{DD}} = \frac{1.7 \times 150\,\text{fF}}{8\,\frac{\mu\text{A}}{\text{V}^2} \times 10 \times 5\,\text{V}} = 0.638\,\text{ns}$$

(*c*) For equivalent high-to-low and low-to-high delays

$$
\begin{aligned}
t_{PHL} &= t_{PLH} \\
\frac{1.7C}{k_n' \frac{W_n}{L_n} V_D D} &= \frac{1.7C}{k_p' \frac{W_p}{L_p} V_D D} \\
\frac{W_p}{L_p} &= \frac{\frac{k_n'}{k_p'} W_n}{L_n} \\
&= \frac{12.5\,\mu\text{m}}{0.5\,\mu\text{m}}
\end{aligned}
$$

3.30. (*a*) The high-to-low propagation delay is

$$t_{PHL} = \frac{1.7C}{k_n' \frac{W_n}{L_n} V_{DD}} = \frac{1.7 \times 150\,\text{fF}}{40\,\frac{\mu\text{A}}{\text{V}^2} \times 10 \times 3.3\,\text{V}} = 0.193\,\text{ns}$$

(*b*) The low-to-high propagation delay is

$$t_{PLH} = \frac{1.7C}{k_p' \frac{W_p}{L_p} V_{DD}} = \frac{1.7 \times 150\,\text{fF}}{16\,\frac{\mu\text{A}}{\text{V}^2} \times 10 \times 3.3\,\text{V}} = 0.483\,\text{ns}$$

(*c*) For equivalent high-to-low and low-to-high delays

$$
\begin{aligned}
t_{PHL} &= t_{PLH} \\
\frac{1.7C}{k_n' \frac{W_n}{L_n} V_D D} &= \frac{1.7C}{k_p' \frac{W_p}{L_p} V_D D} \\
\frac{W_p}{L_p} &= \frac{\frac{k_n'}{k_p'} W_n}{L_n} \\
&= \frac{8.75\,\mu\text{m}}{0.35\,\mu\text{m}}
\end{aligned}
$$

3.31. The two PMOS transistors in a CMOS NAND gate are connected in parallel. The worst case current to drive the output high happens when only one of these transistors is turned "ON". Thus each transistor has to have the same dimensions as the PMOS transistor in the inverter, namely $\frac{W_p}{L_p} = 4$.

The two NMOS transistors are connected in series. If each one had the ratio $\frac{W_n}{L_n}$, then the two transistors could be thought of as one equivalent transistor with a $\frac{W_n}{2L_n}$ ratio. Thus each NMOS transistor must have twice the width of that in the inverter, namely $\frac{W_n}{L_n} = 4$.

3.32. The two NMOS transistors in a CMOS NOR gate are connected in parallel. The worst case current to drive the output low happens when only one of these transistors is turned "ON". Thus each transistor has to have the same dimensions as the NMOS transistor in the inverter, namely $\frac{W_n}{L_n} = 2$.

The two PMOS transistors are connected in series. If each of these transistors had the ratio $\frac{W_p}{L_p}$, then the two transistors could be thought of as one transistor with a $\frac{W_p}{2L_p}$ ratio. Thus each PMOS transistor must be made twice as wide as that in the inverter, namely $\frac{W_n}{L_n} = 8$.

3.33. The worst case path in the PMOS network contains two transistors in series. Thus each PMOS transistor must be twice as wide the transistors in the inverter. The worst case path in the NMOS network also contains two transistors in series. Similarly, each NMOS transistor must be twice as wide as those in the inverter.

3.34. The worst case PMOS path contains three transistors in series so each transistor must be three times as wide as the PMOS transistors in the inverter. The worst case NMOS path contains two transistors in series. Thus the NMOS transistors must be two times as wide.

3.35. (*a*) The current flowing through the inverter is equal to the current flowing through the PMOS transistor. We shall assume that the PMOS transistor is operating in the saturation region.

$$\begin{aligned} I_{stat} &= \frac{1}{2}k'_p\frac{W_p}{L_p}(V_{GS} - V_{Tp})^2 \\ &= 120\,\frac{\mu\text{A}}{\text{V}^2} \times ((3.5\,\text{V} - 5\,\text{V}) + 1\,\text{V})^2 = 30\,\mu\text{A} \end{aligned}$$

(*b*) The current flowing through the NMOS transistor is equal to the static current I_{stat}. Assume that the NMOS transistor is operating in the triode region.

$$\begin{aligned} I_{stat} &= k'_n\frac{W_n}{L_n}\left[(V_{GS} - V_{Tn})V_{DS} - \frac{1}{2}V_{DS}^2\right] \\ 30\,\mu\text{A} &= 240\,\frac{\mu\text{A}}{\text{V}^2} \times \left[2.5\,\text{V} \times V_f - \frac{1}{2}V_f^2\right] \\ 1 &= 20V_f - 4V_f^2 \end{aligned}$$

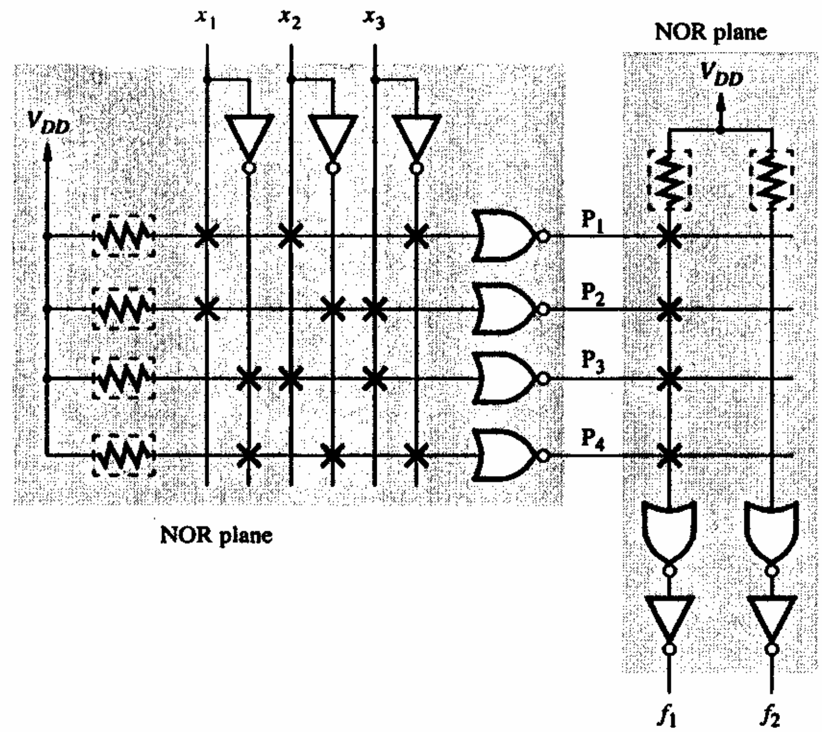
Solving this quadratic equation yields $V_f = 0.05$ V. Note that the output voltage V_f satisfies the assumption that the PMOS transistor is operating in the saturation region while the NMOS transistor is operating in the triode region. (*c*) The static power dissipated in the inverter is

$$P_S = I_{stat}V_{DD} = 30\,\mu\text{A} \times 5\,\text{V} = 150\,\mu\text{W}$$

(*d*) The static power dissipated by 250,000 inverters.

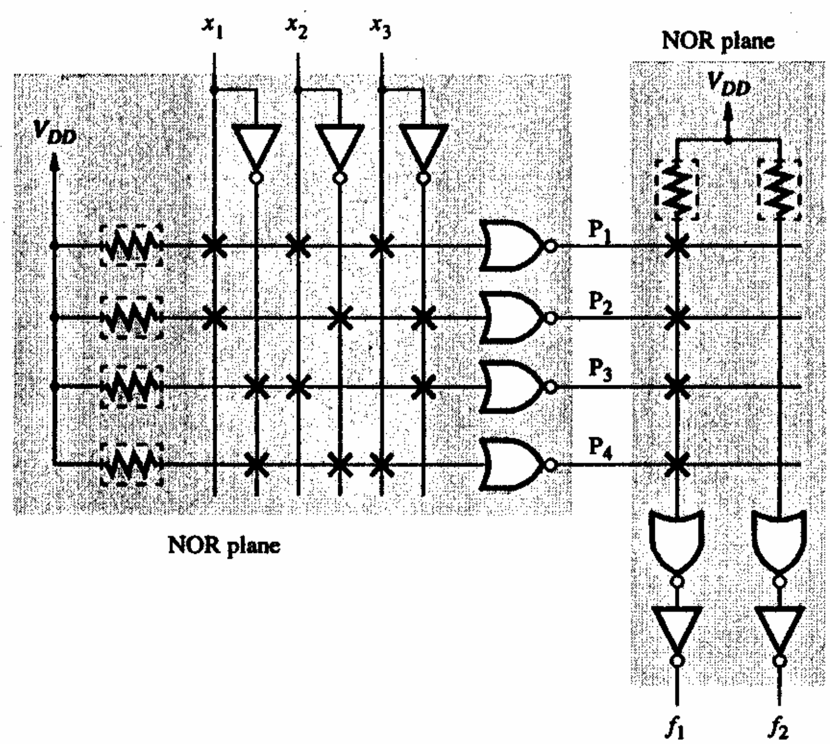
$$250,000 \times P_s = 37.5\,\text{W}$$

3.36.



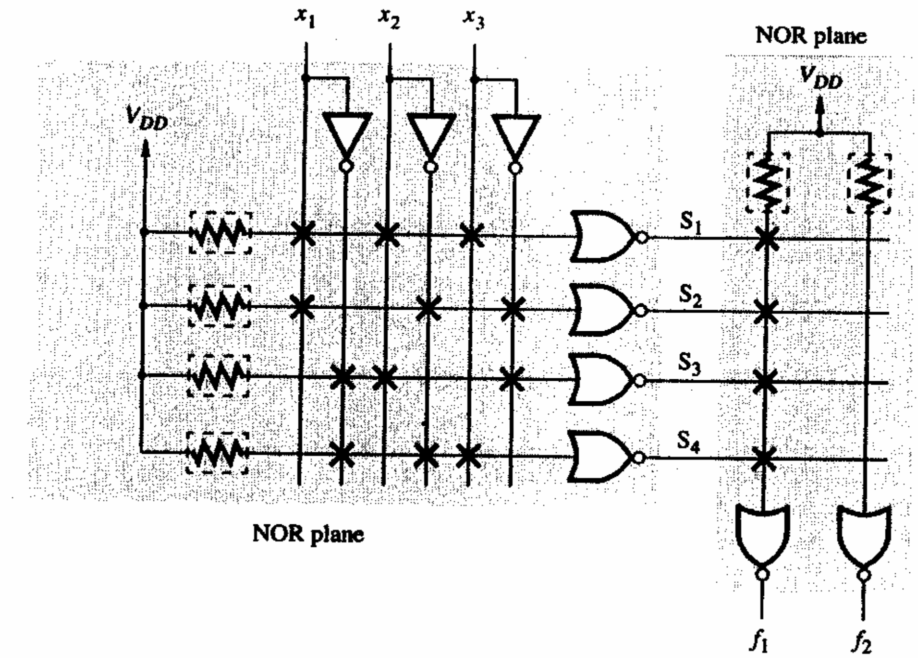
x_1
x_2
x_3
NOR plane
V_{DD}
V_{DD}
P_1
P_2
P_3
P_4
NOR plane
f_1
f_2

3.37.

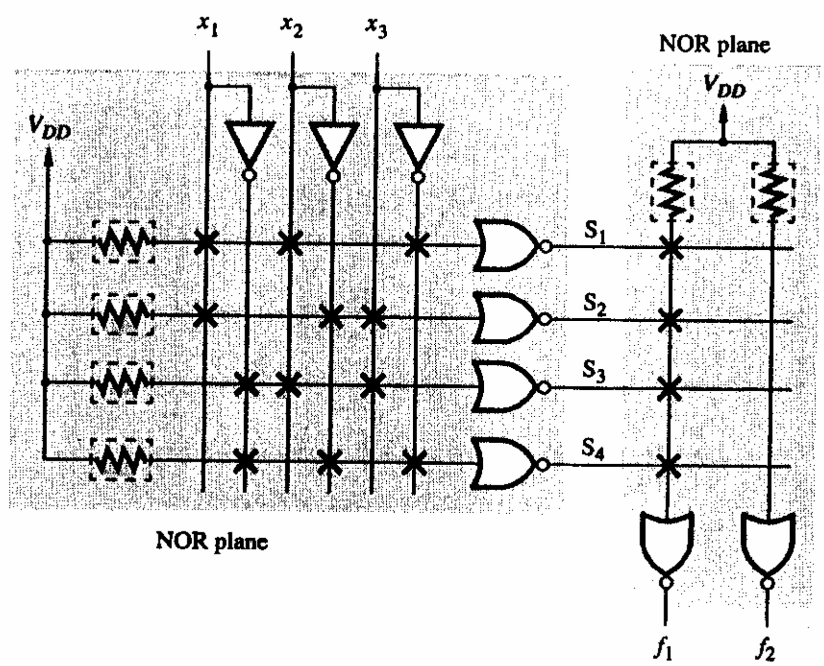


x_1
x_2
x_3
NOR plane
V_{DD}
V_{DD}
P_1
P_2
P_3
P_4
NOR plane
f_1
f_2

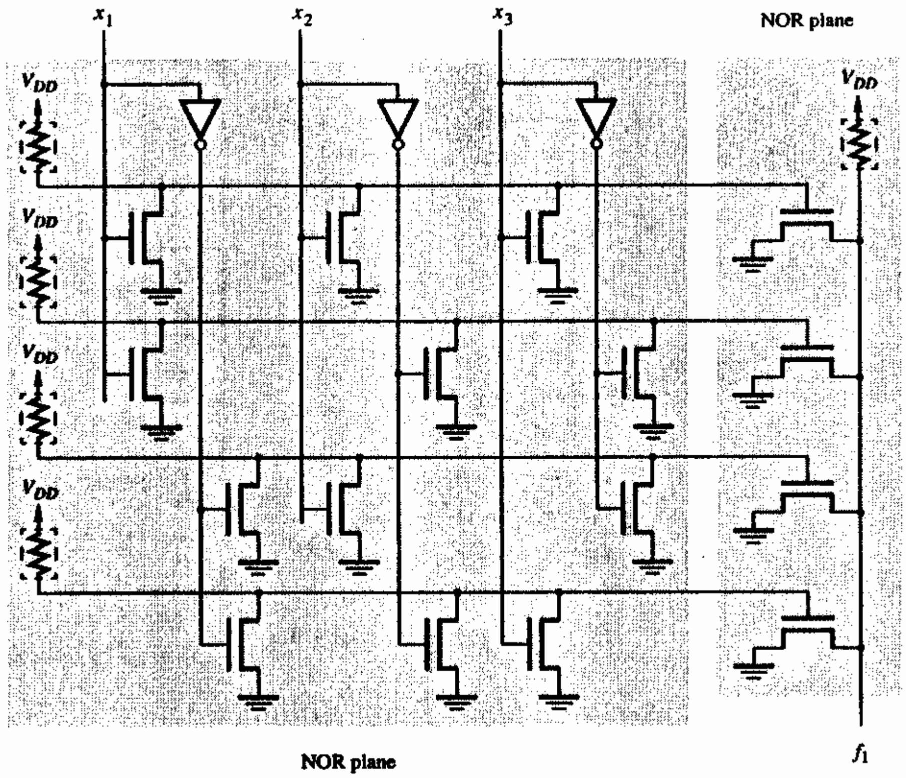
3.38.


x_1
x_2
x_3
NOR plane
V_{DD}
V_{DD}
S_1
S_2
S_3
S_4
NOR plane
f_1
f_2

3.39.

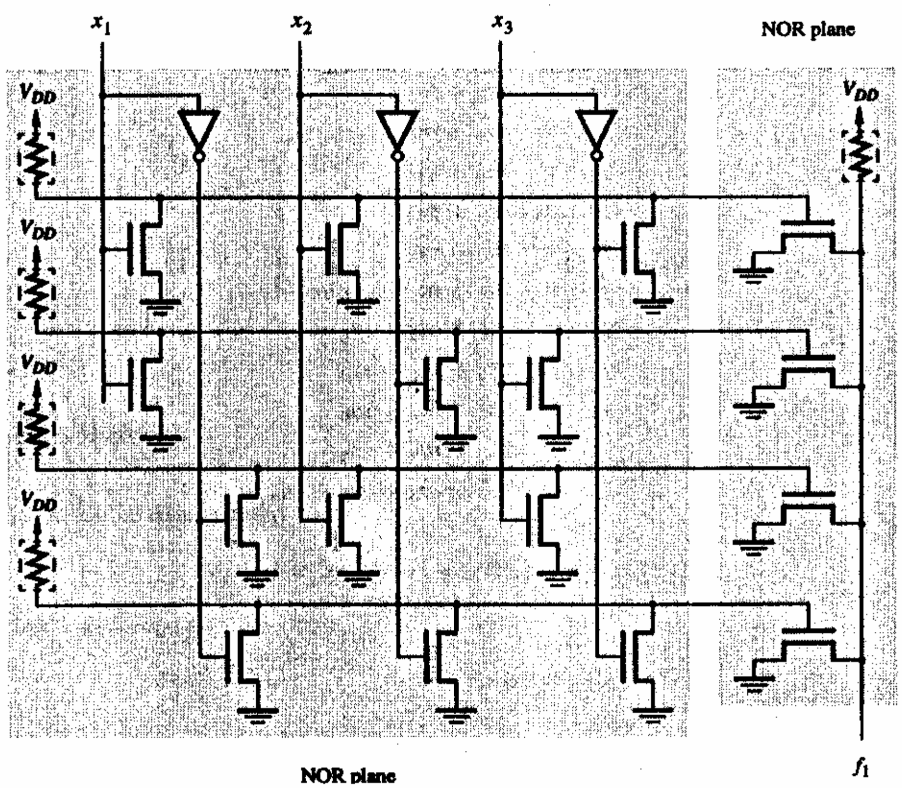

x_1
x_2
x_3
NOR plane
V_{DD}
V_{DD}
S_1
S_2
S_3
S_4
NOR plane
f_1
f_2

3.40.



x_1
x_2
x_3
NOR plane
V_{DD}
V_{DD}
V_{DD}
V_{DD}
V_{DD}
NOR plane
f_1

3.41.



x_1
x_2
x_3
NOR plane
V_{DD}
V_{DD}
V_{DD}
V_{DD}
V_{DD}
NOR plane
f_1

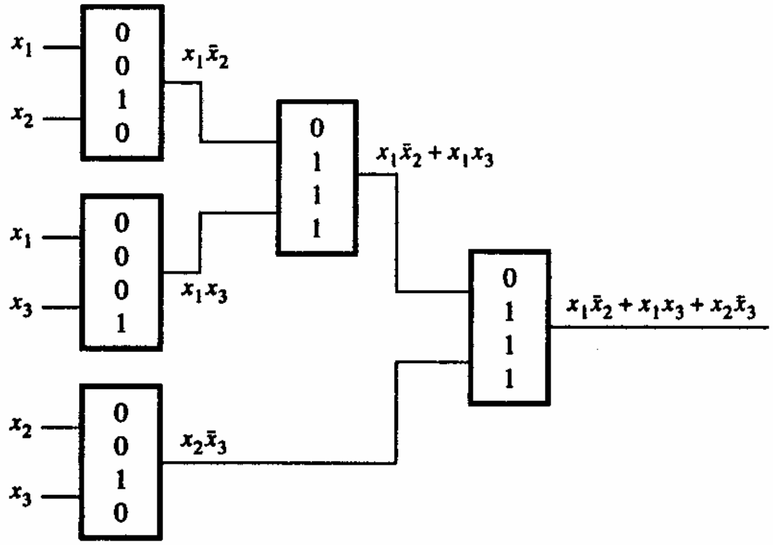
3.42.

$$
\begin{aligned}
f_2 &= m_1 \\
f_2 &= m_2 \\
f_2 &= m_4 \\
f_2 &= m_7 \\
f_2 &= m_1 + m_2 \\
f_2 &= m_1 + m_4 \\
f_2 &= m_1 + m_7 \\
f_2 &= m_2 + m_4 \\
f_2 &= m_2 + m_7 \\
f_2 &= m_4 + m_7 \\
f_2 &= m_1 + m_2 + m_4 \\
f_2 &= m_1 + m_2 + m_7 \\
f_2 &= m_1 + m_4 + m_7 \\
f_2 &= m_2 + m_4 + m_7 \\
f_2 &= m_1 + m_2 + m_4 + m_7
\end{aligned}
$$

3.43.

$$
\begin{aligned}
f_2 &= m_0 \\
f_2 &= m_3 \\
f_2 &= m_5 \\
f_2 &= m_6 \\
f_2 &= m_0 + m_3 \\
f_2 &= m_0 + m_5 \\
f_2 &= m_0 + m_6 \\
f_2 &= m_3 + m_4 \\
f_2 &= m_3 + m_6 \\
f_2 &= m_5 + m_6 \\
f_2 &= m_0 + m_3 + m_5 \\
f_2 &= m_0 + m_3 + m_6 \\
f_2 &= m_0 + m_5 + m_6 \\
f_2 &= m_3 + m_5 + m_6 \\
f_2 &= m_0 + m_3 + m_5 + m_6
\end{aligned}
$$

3.44.



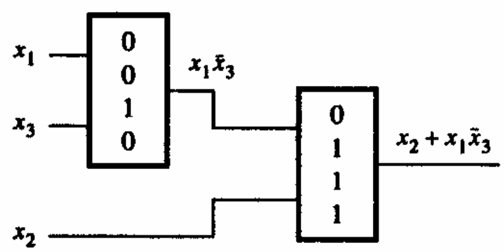
3.45. The canonical SOP for f is

$$f = \overline{x}_1x_2\overline{x}_3 + \overline{x}_1x_2x_3 + x_1\overline{x}_2\overline{x}_3 + x_1x_2\overline{x}_3 + x_1x_2x_3$$

This expression can be manipulated into

$$\begin{aligned} f &= \overline{x}_1x_2 + x_1\overline{x}_3 + x_1x_2 \\ &= x_2 + x_1\overline{x}_3 \end{aligned}$$

The circuit is



3.46. The canonical SOP for f is

$$f = x_1x_2x_4 + x_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3$$

This expression can be manipulated into

$$f = x_2 \cdot (x_1x_4 + x_3\overline{x}_4) + \overline{x}_2 \cdot (\overline{x}_1\overline{x}_3)$$

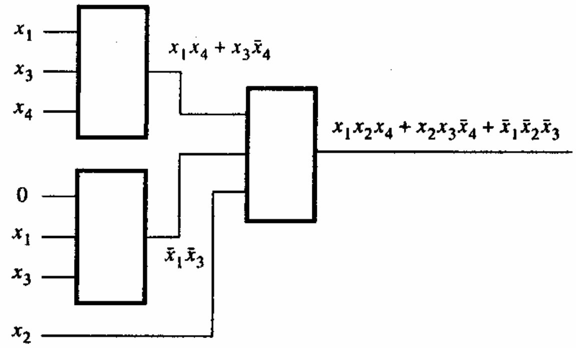
Using functional decomposition we have

$$f = x_2f_1 + \overline{x}_2f_2$$

where

$$\begin{aligned} f_1 &= x_1x_4 + x_3\overline{x}_4 \\ f_2 &= \overline{x}_1\overline{x}_3 \end{aligned}$$

The circuit is



3.47. The canonical SOP for f is

$$f = x_1x_2x_4 + x_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3$$

This expression can be manipulated into

$$f = x_2 \cdot (x_1x_4 + x_3\overline{x}_4) + \overline{x}_2 \cdot (\overline{x}_1\overline{x}_3)$$

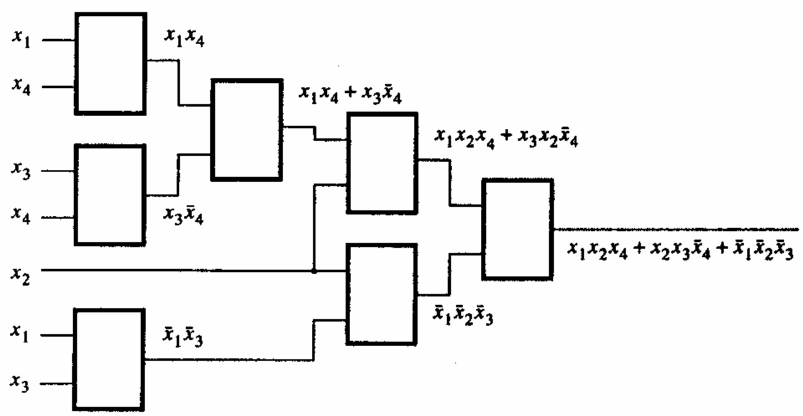
Using functional decomposition we have

$$f = x_2f_1 + \overline{x}_2f_2$$

where

$$\begin{aligned} f_1 &= x_1x_4 + x_3\overline{x}_4 \\ f_2 &= \overline{x}_1\overline{x}_3 \end{aligned}$$

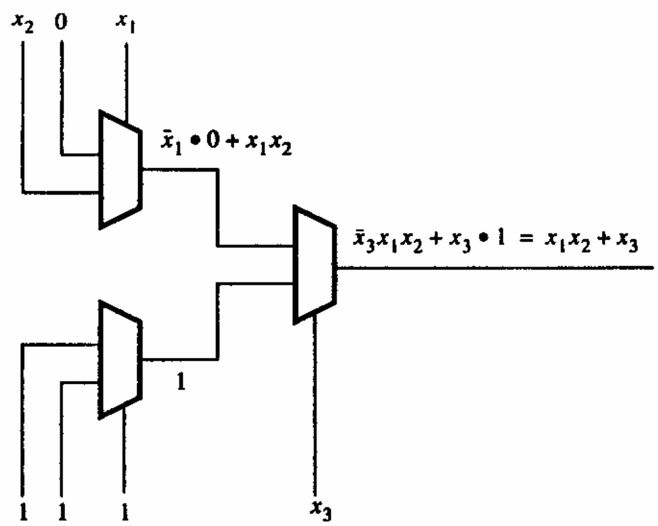
The function f_1 requires one 2-LUT, while f_2 requires three 2-LUTs. We then need three additional 3-LUTs to realize f, as illustrated in the circuit



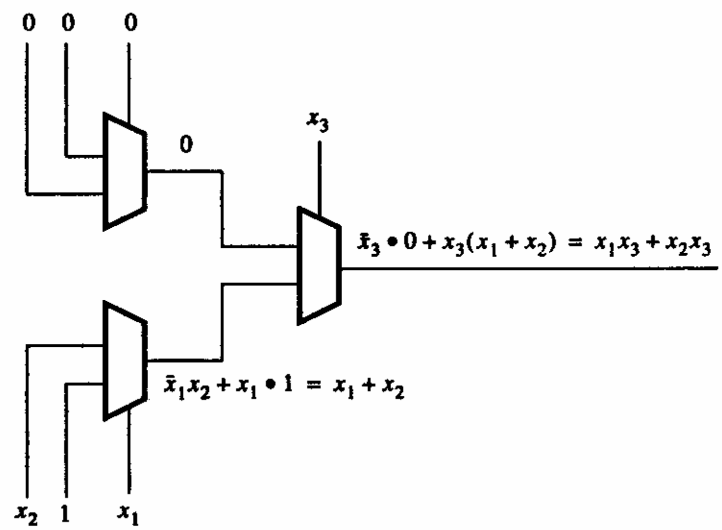
3.48.

$$\begin{aligned} g &= \overline{x}_2x_3 \\ h &= x_1 \\ j &= x_2 \\ k &= x_3 \end{aligned}$$

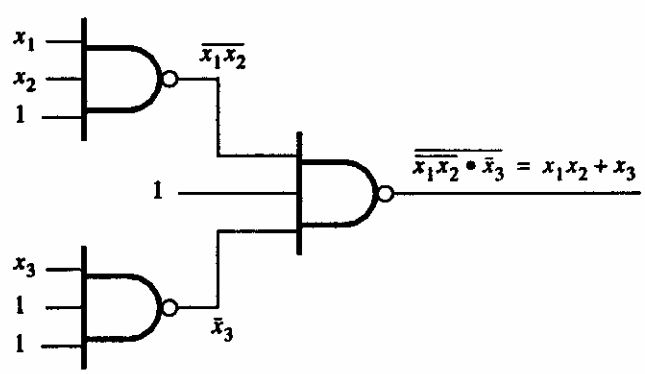
3.49. (*a*)


x_2 0 x_1
$\bar{x}_1 \bullet 0 + x_1 x_2$
$\bar{x}_3 x_1 x_2 + x_3 \bullet 1 = x_1 x_2 + x_3$
1
1 1 1
x_3

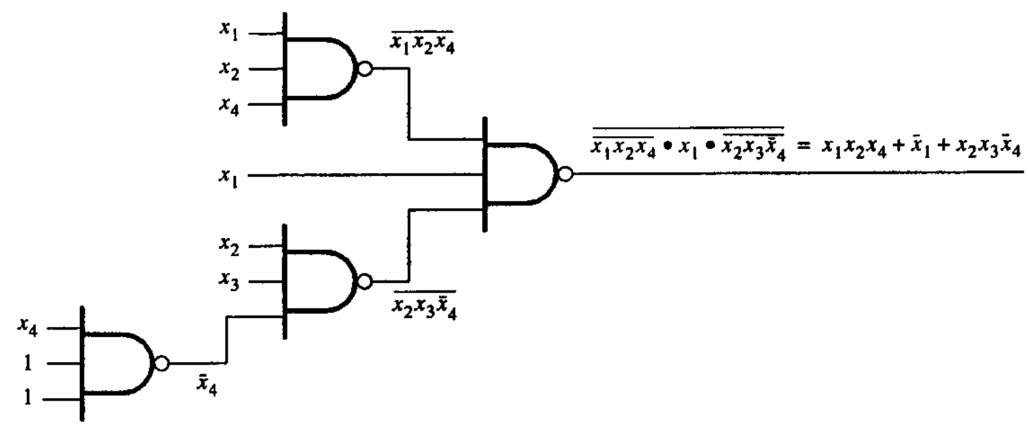
(*b*)


0 0 0
0
x_3
$\bar{x}_3 \bullet 0 + x_3(x_1 + x_2) = x_1 x_3 + x_2 x_3$
$\bar{x}_1 x_2 + x_1 \bullet 1 = x_1 + x_2$
x_2 1 x_1

3.50. (*a*)


x_1
x_2
1
$\overline{x_1 x_2}$
1
$\overline{\overline{x_1 x_2} \bullet \bar{x}_3} = x_1 x_2 + x_3$
x_3
1
1
$\bar{x}_3$

(*b*)



3.51.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob3_51 IS
    PORT ( x1, x2, x3, x4  : IN    STD_LOGIC ;
           f               : OUT   STD_LOGIC ) ;
END prob3_51 ;

ARCHITECTURE LogicFunc OF prob3_51 IS
BEGIN
    f <= (x2 AND NOT x3 AND NOT x4) OR
         (NOT x1 AND x2 AND x4) OR
         (NOT x1 AND x2 AND x3) OR (x1 AND x2 AND x3) ;
END LogicFunc ;
```

3.52.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob3_52 IS
    PORT ( x1, x2, x3, x4  : IN    STD_LOGIC ;
           f               : OUT   STD_LOGIC ) ;
END prob3_52 ;

ARCHITECTURE LogicFunc OF prob3_52 IS
BEGIN
    f <= (x1 OR x2 OR NOT x4) AND
          (NOT x2 OR x3 OR NOT x4) AND
         (NOT x1 OR x3 OR NOT x4) AND
         (NOT x1 OR NOT x3 OR NOT x4) ;
END LogicFunc ;
```

3.53.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob3_53 IS
    PORT ( x1, x2, x3, x4, x5, x6, x7  : IN    STD_LOGIC ;
               f                        : OUT  STD_LOGIC ) ;
END prob3_53 ;

ARCHITECTURE LogicFunc OF prob3_53 IS
BEGIN
    f <= (x1 AND x3 AND NOT x6) OR
         (x1 AND x4 AND x5 AND NOT x6) OR
         (x2 AND x3 AND x7) OR
         (x2 AND x4 AND x5 AND x7) ;
END LogicFunc ;
```

3.54. The circuit in Figure P3.10 is a two-input XOR gate. Since NMOS transistors are used only to pass logic 0 and PMOS transistors are used only to pass logic 1, the circuit does nor suffer from any major drawbacks.

3.55. The circuit in Figure P3.11 is a two-input XOR gate. This circuit has two drawbacks: when both inputs are 0 the PMOS transistor must drive f to 0, resulting in $f = V_T$ volts. Also, when $x_1 = 1$ and $x_2 = 0$, the NMOS transistor must drive the output high, resulting in $f = V_{DD} - V_T$.

Thank you, Abe

Please report any mistake to the instructor.

Chapter 4

4.1. SOP form: $f = \overline{x}_1 x_2 + \overline{x}_2 x_3$
POS form: $f = (\overline{x}_1 + \overline{x}_2)(x_2 + x_3)$

4.2. SOP form: $f = x_1\overline{x}_2 + x_1 x_3 + \overline{x}_2 x_3$
POS form: $f = (x_1 + x_3)(x_1 + \overline{x}_2)(\overline{x}_2 + x_3)$

4.3. SOP form: $f = \overline{x}_1 x_2 x_3 \overline{x}_4 + x_1 x_2 \overline{x}_3 x_4 + \overline{x}_2 x_3 x_4$
POS form: $f = (\overline{x}_1 + x_4)(x_2 + x_3)(\overline{x}_2 + \overline{x}_3 + \overline{x}_4)(x_2 + x_4)(x_1 + x_3)$

4.4. SOP form: $f = \overline{x}_2\overline{x}_3 + \overline{x}_2\overline{x}_4 + x_2 x_3 x_4$
POS form: $f = (\overline{x}_2 + x_3)(x_2 + \overline{x}_3 + \overline{x}_4)(\overline{x}_2 + x_4)$

4.5. SOP form: $f = \overline{x}_3\overline{x}_5 + \overline{x}_3 x_4 + x_2 x_4 \overline{x}_5 + \overline{x}_1 x_3 \overline{x}_4 x_5 + x_1 x_2 \overline{x}_4 x_5$
POS form: $f = (\overline{x}_3 + x_4 + x_5)(\overline{x}_3 + \overline{x}_4 + \overline{x}_5)(x_2 + \overline{x}_3 + \overline{x}_4)(x_1 + x_3 + x_4 + \overline{x}_5)(\overline{x}_1 + x_2 + x_4 + \overline{x}_5)$

4.6. SOP form: $f = \overline{x}_2 x_3 + \overline{x}_1 x_5 + \overline{x}_1 x_3 + \overline{x}_3\overline{x}_4 + \overline{x}_2 x_5$
POS form: $f = (\overline{x}_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_4)(x_3 + \overline{x}_4 + x_5)$

4.7. SOP form: $f = x_3\overline{x}_4\overline{x}_5 + \overline{x}_3\overline{x}_4 x_5 + x_1 x_4 x_5 + x_1 x_2 x_4 + x_3 x_4 x_5 + \overline{x}_2 x_3 x_4 + x_2\overline{x}_3 x_4 \overline{x}_5$
POS form: $f = (x_3 + x_4 + x_5)(\overline{x}_3 + x_4 + \overline{x}_5)(x_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4 + x_5)$

4.8. $f = \sum m(0,7)$
$f = \sum m(1,6)$
$f = \sum m(2,5)$
$f = \sum m(0,1,6)$
$f = \sum m(0,2,5)$
etc.

4.9. $f = x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_3 x_4 + x_2 x_3 x_4$

4.10. SOP form: $f = x_1 x_2 \overline{x}_3 + x_1 \overline{x}_2 x_4 + x_1 x_3 \overline{x}_4 + \overline{x}_1 x_2 x_3 + \overline{x}_1 x_3 x_4 + x_2 \overline{x}_3 x_4$
POS form: $f = (x_1 + x_2 + x_3)(x_1 + x_2 + x_4)(x_1 + x_3 + x_4)(x_2 + x_3 + x_4)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4)$
The POS form has lower cost.

4.11. The statement is false. As a counter example consider $f(x_1, x_2, x_3) = \sum m(0,5,7)$.
Then, the minimum-cost SOP form $f = x_1 x_3 + \overline{x}_1\overline{x}_2\overline{x}_3$ is unique.
But, there are two minimum-cost POS forms:
$f = (x_1 + \overline{x}_3)(\overline{x}_1 + x_3)(x_1 + \overline{x}_2)$ and
$f = (x_1 + \overline{x}_3)(\overline{x}_1 + x_3)(\overline{x}_2 + x_3)$

4.12. If each circuit is implemented separately:
$f = \overline{x}_1\overline{x}_4 + \overline{x}_1x_2x_3 + x_1\overline{x}_2x_4$ Cost= 15
$g = \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_2x_3\overline{x}_4 + x_1\overline{x}_3x_4 + x_1x_2x_4$ Cost = 21

In a combined circuit:
$f = \overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3x_4 + \overline{x}_1x_2x_3$
$g = \overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3x_4 + x_1x_2x_4$
The first 3 product terms are shared, hence the total cost is 31.

4.13. If each circuit is implemented separately:
$f = \overline{x}_1x_2x_4 + x_2x_4x_5 + x_3\overline{x}_4\overline{x}_5 + \overline{x}_1\overline{x}_2\overline{x}_4x_5$ Cost = 22
$g = \overline{x}_3\overline{x}_5 + \overline{x}_4\overline{x}_5 + \overline{x}_1\overline{x}_2\overline{x}_4 + \overline{x}_1x_2x_4 + x_2x_4x_5$ Cost = 24

In a combined circuit:
$f = \overline{x}_1x_2x_4 + x_2x_4x_5 + x_3\overline{x}_4\overline{x}_5 + \overline{x}_1\overline{x}_2\overline{x}_4x_5$
$g = \overline{x}_1x_2x_4 + x_2x_4x_5 + x_3\overline{x}_4\overline{x}_5 + \overline{x}_1\overline{x}_2\overline{x}_4x_5 + \overline{x}_3\overline{x}_5$

The first 4 product terms are shared, hence the total cost is 31. Note that in this implementation $f \subseteq g$, thus g can be realized as $g = f + \overline{x}_3\overline{x}_5$, in which case the total cost is lowered to 28.

4.14. $f = (x_3 \uparrow g) \uparrow ((g \uparrow g) \uparrow x_4)$ where $g = (x_1 \uparrow (x_2 \uparrow x_2)) \uparrow ((x_1 \uparrow x_1) \uparrow x_2)$

4.15. $\overline{f} = (((x_3 \downarrow x_3) \downarrow g) \downarrow ((g \downarrow g) \downarrow (x_4 \downarrow x_4))$, where
$g = ((x_1 \downarrow x_1) \downarrow x_2) \downarrow (x_1 \downarrow (x_2 \downarrow x_2))$. Then, $f = \overline{f} \downarrow \overline{f}$.

4.16. $f = (g \uparrow k) \uparrow ((g \uparrow g) \uparrow (k \uparrow k))$, where $g = (x_1 \uparrow x_1) \uparrow (x_2 \uparrow x_2) \uparrow (x_5 \uparrow x_5)$
and $k = (x_3 \uparrow (x_4 \uparrow x_4)) \uparrow ((x_3 \uparrow x_3) \uparrow x_4)$

4.17. $\overline{f} = (g \downarrow k) \downarrow ((g \downarrow g) \downarrow (k \downarrow k))$, where $g = x_1 \downarrow x_2 \downarrow x_5$
and $k = ((x_3 \downarrow x_3) \downarrow x_4) \downarrow (x_3 \downarrow (x_4 \downarrow x_4))$. Then, $f = \overline{f} \downarrow \overline{f}$.

4.18. $f = \overline{x}_1(x_2 + x_3)(x_4 + x_5) + x_1(\overline{x}_2 + x_3)(\overline{x}_4 + x_5)$

4.19. $f = x_1\overline{x}_3\overline{x}_4 + x_2\overline{x}_3\overline{x}_4 + x_1x_3x_4 + x_2x_3x_4 = (x_1 + x_2)\overline{x}_3\overline{x}_4 + (x_1 + x_2)x_3x_4$
This requires 2 OR and 2 AND gates.

4.20. $f = x_1 \cdot g + \overline{x}_1 \cdot \overline{g}$, where $g = \overline{x}_3x_4 + x_3\overline{x}_4$

4.21 $f = g \cdot h + \overline{g} \cdot \overline{h}$, where $g = x_1x_2$ and $h = x_3 + x_4$

4.22. Let $D(0, 20)$ be 0 and $D(15, 26)$ be 1. Then decomposition yields:
$g = x_5(\overline{x}_1 + x_2)$
$f = (\overline{x}_3\overline{x}_4 + x_3x_4)g + \overline{x}_3x_4\overline{g} = x_3x_4g + \overline{x}_3\overline{x}_4g + \overline{x}_3x_4\overline{g}$
Cost $= 9 + 18 = 27$

The optimal SOP form is:
$f = \overline{x}_3 x_4 \overline{x}_5 + \overline{x}_1 x_3 x_4 x_5 + x_1 \overline{x}_2 \overline{x}_3 x_4 + \overline{x}_1 \overline{x}_3 \overline{x}_4 x_5 + x_2 \overline{x}_3 \overline{x}_4 x_5 + x_2 x_3 x_4 x_5$
Cost $= 7 + 29 = 36$

4.23. Note that $X \# Y = X \cdot \overline{Y}$. Therefore,

$$\begin{aligned}(A \cdot B)\#C &= A \cdot B \cdot \overline{c} \\ (A\#C) \cdot (B\#C) &= A \cdot \overline{C} \cdot B \cdot \overline{C} \\ &= A \cdot B \cdot \overline{C}\end{aligned}$$

Similarly,

$$\begin{aligned}(A + B)\#C &= (A + B) \cdot \overline{C} \\ &= A \cdot \overline{C} + B \cdot \overline{C} \\ (A\#C) + (B\#C) &= A \cdot \overline{C} + B \cdot \overline{C}\end{aligned}$$

4.24. The initial cover is $C^0 = \{0000, 0011, 0100, 0101, 0111, 1000, 1001, 1111\}$.
Using the $*$-product get the prime implicants
$P = \{00x0, 0x00, x000, 010x, 01x1, 100x, x111\}$.
The minimum cover is $C_{minimum} = \{00x0, 010x, 100x, x111\}$, which corresponds to $f = \overline{x}_1 \overline{x}_2 \overline{x}_4 + \overline{x}_1 x_2 \overline{x}_3 + x_1 \overline{x}_2 \overline{x}_3 + x_2 x_3 x_4$.

4.25. The initial cover is $C^0 = \{0x0x0, 110xx, x1101, 1001x, 11110, 01x10, 0x011\}$.
Using the $*$-product get the prime implicants
$P = \{0x0x0, xx01x, x1x10, 110xx, x10x0, 11x01, x1101\}$.
The minimum cover is $C_{mimimum} = \{0x0x0, xx01x, x1x10, 110xx, x1101\}$, which corresponds to $f = \overline{x}_1 \overline{x}_3 \overline{x}_5 + \overline{x}_3 x_4 + x_2 x_4 \overline{x}_5 + x_1 x_2 \overline{x}_3 + x_2 x_3 \overline{x}_4 x_5$.

4.26. The initial cover is $C^0 = \{00x0, 100x, x010, 1111, 00x1, 011x\}$.
Using the $*$-product get the prime implicants $P = \{00xx, 0x1x, x00x, x0x0, x111\}$.
The minimum-cost cover is $C_{minimum} = \{x00x, x0x0, x111\}$, which corresponds to $f = \overline{x}_2 \overline{x}_3 + \overline{x}_2 \overline{x}_4 + x_2 x_3 x_4$.

4.27. Expansion of $\overline{x}_1 \overline{x}_2 \overline{x}_3$ gives $\overline{x}_1$.
Expansion of $\overline{x}_1 \overline{x}_2 x_3$ gives $\overline{x}_1$.
Expansion of $\overline{x}_1 x_2 \overline{x}_3$ gives $\overline{x}_1$.
Expansion of $x_1 x_2 x_3$ gives $x_2 x_3$.
The set of prime implicants comprises $\overline{x}_1$ and $x_2 x_3$.

4.28. Expansion of $\overline{x}_1 x_2 \overline{x}_3 x_4$ gives $x_2 \overline{x}_3 x_4$ and $\overline{x}_1 x_2 x_4$.
Expansion of $x_1 x_2 \overline{x}_3 x_4$ gives $x_2 \overline{x}_3 x_4$.
Expansion of $x_1 x_2 x_3 \overline{x}_4$ gives $x_3 \overline{x}_4$.
Expansion of $\overline{x}_1 x_2 x_3$ gives $\overline{x}_1 x_3$.
Expansion of $\overline{x}_2 x_3$ gives $\overline{x}_2 x_3$.
The set of prime implicants comprises $x_2 \overline{x}_3 x_4$, $\overline{x}_1 x_2 x_4$, $x_3 \overline{x}_4$, $\overline{x}_1 x_3$, and $\overline{x}_2 x_3$.

4.29. Representing both functions in the form of Karnaugh map, it is easy to show that $f = g$. The minimum cost SOP expression is
$f = g = \overline{x}_2\overline{x}_3\overline{x}_5 + \overline{x}_2x_3\overline{x}_4 + x_1x_3x_4 + x_1x_2x_4x_5$.

4.30. The cost of the circuit in Figure P4.2 is 11 gates and 30 inputs, for a total of 41. The functions implemented by the circuit can also be realized as

$$\begin{aligned} f &= \overline{x}_1\overline{x}_2\overline{x}_4 + x_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_3x_4 + x_1x_4 \\ g &= \overline{x}_1\overline{x}_2\overline{x}_4 + x_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_3x_4 + \overline{x}_2x_4 + x_3\overline{x}_4 \end{aligned}$$

The first three product terms in f and g are the same; therefore, they can be shared. Then, the cost of implementing f and g is 8 gates and 24 inputs, for a total of 32.

4.31. The cost of the circuit in Figure P4.3 is 11 gates and 26 inputs, for a total of 37. The functions implemented by the circuit can also be realized as

$$\begin{aligned} f &= (\overline{x}_2 \uparrow x_4) \uparrow (\overline{x}_1 \uparrow x_2 \uparrow x_3) \uparrow (x_1 \uparrow \overline{x}_2 \uparrow x_3) \uparrow (\overline{x}_2 \uparrow \overline{x}_3) \\ g &= (\overline{x}_2 \uparrow x_4) \uparrow (\overline{x}_1 \uparrow x_2 \uparrow x_3) \uparrow (x_1 \uparrow \overline{x}_2 \uparrow x_3) \uparrow (\overline{x}_1 \uparrow \overline{x}_1) \end{aligned}$$

The first three NAND terms in f and g are the same; therefore, they can be shared. Then, the cost of implementing f and g is 7 gates and 20 inputs, for a total of 27.

Chapter 5

5.1. (*a*) 478
(*b*) 743
(*c*) 2025
(*d*) 41567
(*e*) 61680

5.2. (*a*) 478
(*b*) −280
(*c*) −1

5.3. (*a*) 478
(*b*) −281
(*c*) −2

5.4. The numbers are represented as follows:

Decimal	Sign and Magnitude	1's Complement	2's Complement
73	000001001001	000001001001	000001001001
1906	011101110010	011101110010	011101110010
−95	100001011111	111110100000	111110100001
−1630	111001011110	100110100001	100110100010

5.5. The results of the operations are:

(*a*):	00110110	54	(*b*):	01110101	117	(*c*):	11011111	(−33)
	+01000101	+69		+11011110	− 34		+10111000	+(−72)
	01111011	123		01010011	83		10010111	(−105)
(*d*):	00110110	54	(*e*):	01110101	(117)	(*f*):	11010011	(−45)
	−00101011	−43		−11010110	−(− 42)		−11101100	−(−20)
	00001011	11		10011111	(159)		11100111	(−25)

Arithmetic overflow occurs in example *e*; note that the pattern 10011111 represents −97 rather than +159.

5.6. The associativity of the XOR operation can be shown as follows:

$$\begin{aligned}
x \oplus (y \oplus z) &= x \oplus (\overline{y}z + y\overline{z}) \\
&= \overline{x}(\overline{y}z + y\overline{z}) + x(\overline{y} \cdot \overline{z} + yz) \\
&= \overline{x} \cdot \overline{y}z + \overline{x}y\overline{z} + x\overline{y} \cdot \overline{z} + xyz
\end{aligned}$$

$$\begin{aligned}
(x \oplus y) \oplus z &= (\overline{x}y + x\overline{y}) \oplus z \\
&= (\overline{x} \cdot \overline{y} + xy)z + (\overline{x}y + x\overline{y})\overline{z} \\
&= \overline{x} \cdot \overline{y}z + xyz + \overline{x}y\overline{z} + x\overline{y} \cdot \overline{z}
\end{aligned}$$

The two SOP expressions are the same.

5.7. In the circuit of Figure 5.5b, we have:

$$\begin{aligned}
s_i &= (x_i \oplus y_i) \oplus c_i \\
&= x_i \oplus y_i \oplus c_i
\end{aligned}$$

$$\begin{aligned}
c_{i+1} &= (x_i \oplus y_i)c_i + x_i y_i \\
&= (\overline{x}_i y_i + x_i \overline{y}_i)c_i + x_i y_i \\
&= \overline{x}_i y_i c_i + x_i \overline{y}_i c_i + x_i y_i \\
&= y_i c_i + x_i c_i + x_i y_i
\end{aligned}$$

The expressions for s_i and c_{i+1} are the same as those derived in Figure 5.4*b*.

5.8. We will give a descriptive proof for ease of understanding. The 2's complement of a given number can be found by adding 1 to the 1's complement of the number. Suppose that the number has k 0s in the least-significant bit positions, $b_{k-1} \ldots b_0$, and it has $b_k = 1$. When this number is converted to its 1's complement, each of these k bits has the value 1. Adding 1 to this string of 1s produces $b_k b_{k-1} b_{k-2} \ldots b_0 = 100 \ldots 0$. This result is equivalent to copying the k 0s and the first 1 (in bit position b_k) encountered when the number is scanned from right to left. Suppose that the most-significant $n-k$ bits, $b_{n-1}b_{n-2} \ldots b_k$, have some pattern of 0s and 1s, but $b_k = 1$. In the 1's complement this pattern will be complemented in each bit position, which will include $b_k = 0$. Now, adding 1 to the entire n-bit number will make $b_k = 1$, but no further carries will be generated; therefore, the complemented bits in positions $b_{n-1}b_{n-2} \ldots b_{k+1}$ will remain unchanged.

5.9. Construct the truth table

x_{n-1}	y_{n-1}	c_{n-1}	c_n	s_{n-1} (sign bit)	Overflow
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Note that overflow cannot occur when two numbers with opposite signs are added. From the truth table the overflow expression is

$$Overflow = \overline{c}_n c_{n-1} + c_n \overline{c}_{n-1} = c_n \oplus c_{n-1}$$

5.10. Since $s_k = x_k \oplus y_k \oplus c_k$, it follows that

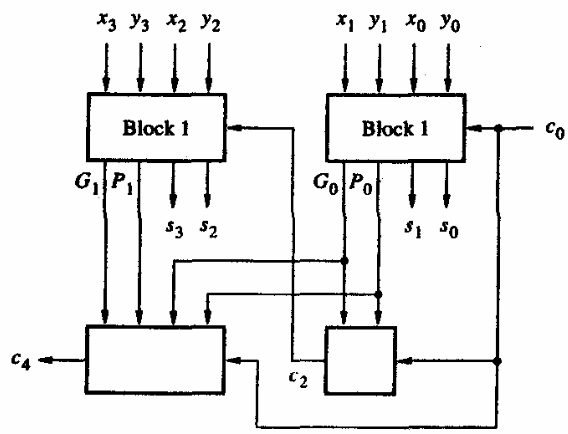
$$\begin{aligned}
x_k \oplus y_k \oplus s_k &= (x_k \oplus y_k) \oplus (x_k \oplus y_k \oplus c_k) \\
&= (x_k \oplus y_k) \oplus (x_k \oplus y_k) \oplus c_k \\
&= 0 \oplus c_k \\
&= c_k
\end{aligned}$$

5.11. Yes, it works. The NOT gate that produces c_i is not needed in stages where $i > 0$. The drawback is "poor" propagation of $\bar{c}_i = 1$ through the topmost NMOS transistor. The positive aspect is fewer transistors needed to produce $\bar{c}_{i+1}$.

5.12. From Expression 5.4, each c_i requires i AND gates and one OR gate. Therefore, to determine all c_i signals we need $\sum_{i=1}^{n}(i+1) = (n^2+3n)/2$ gates. In addition to this, we need $3n$ gates to generate all g, p, and s functions. Therefore, a total of $(n^2+9n)/2$ gates are needed.

5.13. 84 gates.

5.14. The circuit for a 4-bit version of the adder based on the hierarchical structure in Figure 5.18 is constructed as follows:

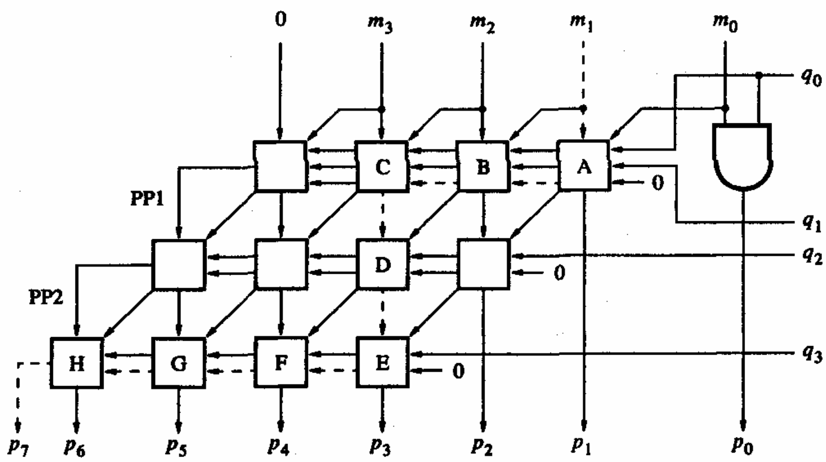


Blocks 0 and 1 have the structure similar to the circuit in Figure 5.16. The overall circuit is given by the expressions

$$\begin{aligned}
p_i &= x_i + y_i \\
g_i &= x_i y_i \\
P_0 &= p_1 p_0 \\
G_0 &= g_1 + p_1 g_0
\end{aligned}$$

$$
\begin{aligned}
P_1 &= p_3p_2 \\
G_1 &= g_3 + p_3g_2 \\
c_2 &= G_0 + P_0c_0 \\
c_4 &= G_1 + P_1G_0 + P_1P_0c_0
\end{aligned}
$$

5.15. The longest path, which causes the critical delay, is from the inputs m_0 and m_1 to the output p_7, indicated by the dashed path in the following copy of Figure 5.33*a*:



Propagation through the block A involves one gate delay in the AND gate shown in Figure 5.33*b* and two gate delays to generate the carry-out in the full-adder. Then, in each of the blocks B, C, D, E, F, G, and H, two more gate delays are needed to generate the carry-out signals in the circuits depicted by Figure 5.33*c*. Therefore, the total delay along the critical path is 17 gate delays.

5.16. (*a*)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY row0 IS
    PORT ( q0, q1, cin, mk, mkp1  : IN    STD_LOGIC ;
           s, cout                : OUT   STD_LOGIC ) ;
END row0;

ARCHITECTURE LogicFunc OF row0 IS
    SIGNAL a0, a1 : STD_LOGIC ;
BEGIN
    a0 <= q0 AND mkp1 ;
    a1 <= q1 AND mk ;
    s <= cin XOR a0 XOR a1 ;
    cout <= (cin AND a0) OR (cin AND a1) OR (a0 AND a1) ;
END LogicFunc ;
```

(*b*)
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY row1 IS
     PORT ( qj, cin, mk, BitPPi  : IN     STD_LOGIC ;
            s, cout              : OUT  STD_LOGIC ) ;
END row1 ;

ARCHITECTURE LogicFunc OF row1 IS
     SIGNAL a0 : STD_LOGIC ;
BEGIN
     a0 <= qj AND mk ;
     s <= cin XOR a0 XOR BitPPi ;
     cout <= (cin AND a0) OR (cin AND BitPPi) OR (a0 AND BitPPi) ;
END LogicFunc ;
```

(*c*)
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY mult4x4 IS
     PORT ( cin     : IN     STD_LOGIC ;
            M, Q    : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            P       : OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) );
END mult4x4 ;

ARCHITECTURE Structure OF mult4x4 IS
     COMPONENT row0
          PORT ( q0, q1, cin, mk, mkp1  : IN     STD_LOGIC ;
                 s, cout                : OUT  STD_LOGIC ) ;
     END COMPONENT ;
     COMPONENT row1
          PORT ( qj, cin, mk, BitPPi  : IN     STD_LOGIC ;
                 s, cout              : OUT  STD_LOGIC ) ;
     END COMPONENT ;
     SIGNAL PP1     : STD_LOGIC_VECTOR(5 DOWNTO 2) ;
     SIGNAL PP2     : STD_LOGIC_VECTOR(6 DOWNTO 3) ;
     SIGNAL Crow0, Crow1, Crow2 : STD_LOGIC_VECTOR(1 TO 3) ;
BEGIN
     P(0) <= Q(0) AND M(0) ;
     row0_1: row0 PORT MAP ( Q(0), Q(1), cin, M(0), M(1), P(1), Crow0(1) ) ;
     row0_2: row0 PORT MAP ( Q(0), Q(1), Crow0(1), M(1), M(2), PP1(2), Crow0(2) ) ;
     row0_3: row0 PORT MAP ( Q(0), Q(1), Crow0(2), M(2), M(3), PP1(3), Crow0(3) ) ;
     row0_4: row0 PORT MAP ( Q(0), Q(1), Crow0(3), M(3), cin, PP1(4), PP1(5) ) ;
     row1_2: row1 PORT MAP ( Q(2), cin, M(0), PP1(2), P(2), Crow1(1) ) ;
     row1_3: row1 PORT MAP ( Q(2), Crow1(1), M(1), PP1(3), PP2(3), Crow1(2) ) ;
     row1_4: row1 PORT MAP ( Q(2), Crow1(2), M(2), PP1(4), PP2(4), Crow1(3) ) ;
     row1_5: row1 PORT MAP ( Q(2), Crow1(3), M(3), PP1(5), PP2(5), PP2(6) ) ;
     row2_3: row1 PORT MAP ( Q(3), cin, M(0), PP2(3), P(3), Crow2(1) ) ;
     row2_4: row1 PORT MAP ( Q(3), Crow2(1), M(1), PP2(4), P(4), Crow2(2) ) ;
     row2_5: row1 PORT MAP ( Q(3), Crow2(2), M(2), PP2(5), P(5), Crow2(3) ) ;
     row2_6: row1 PORT MAP ( Q(3), Crow2(3), M(3), PP2(6), P(6), P(7) ) ;
END Structure ;
```

5.17. The code in Figure P5.2 represents a multiplier. It multiplies the lower two bits of *Input* by the upper two bits of *Input*, producing the four-bit *Output*. The style of code is poor, because it is not readily apparent what is being described.

5.18. Let $Y = y_3y_2y_1y_0$ be the 9's complement of the BCD digit $X = x_3x_2x_1x_0$. Then, Y is defined by the truth table

x_3	x_2	x_1	x_0	y_3	y_2	y_1	y_0
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0

This gives

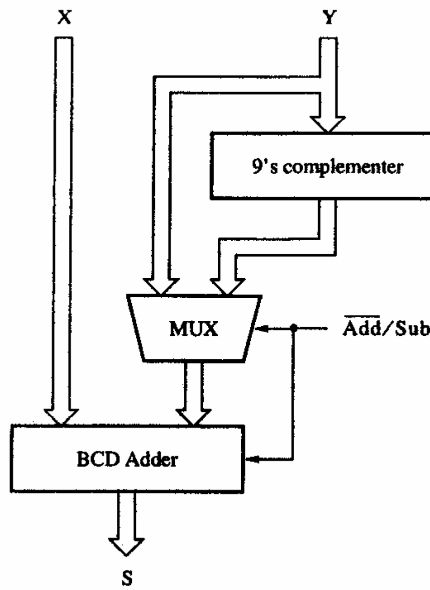
$$
\begin{aligned}
y_0 &= \overline{x}_0 \\
y_1 &= x_1 \\
y_2 &= \overline{x}_2x_1 + x_2\overline{x}_1 \\
y_3 &= \overline{x}_3\overline{x}_2\overline{x}_1
\end{aligned}
$$

5.19. BCD subtraction can be performed using 10's complement representation, using an approach that is similar to 2's complement subtraction. Note that 10's and 2's complements are the radix complements in number systems where the radices are 10 and 2, respectively. Let X and Y be BCD numbers given in 10's complement representation, such that the sign (left-most) BCD digit is 0 for positive numbers and 9 for negative numbers. Then, the subtraction operation $S = X - Y$ is performed by finding the 10's complement of Y and adding it to X, ignoring any carry-out from the sign-digit position.

For example, let $X = 068$ and $Y = 043$. Then, the 10's complement of Y is 957, and $S' = 068 + 957 = 1025$. Dropping the carry-out of 1 from the sign-digit position gives $S = 025$.

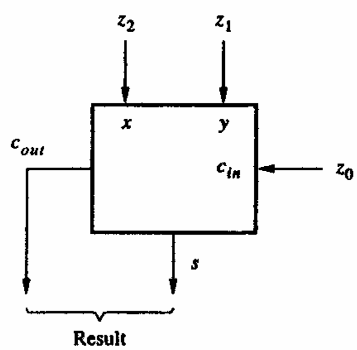
As another example, let $X = 032$ and $Y = 043$. Then, $S = 032 + 957 = 989$, which represents -11_{10}.

The 10's complement of Y can be formed by adding 1 to the 9's complement of Y. Therefore, a circuit that can add and subtract BCD operands can be designed as follows:

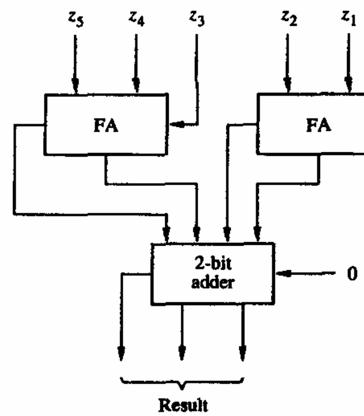


For the 9's complementer one can use the circuit designed in problem 5.18. The BCD adder is a circuit based on the approach illustrated in Figure 5.40.

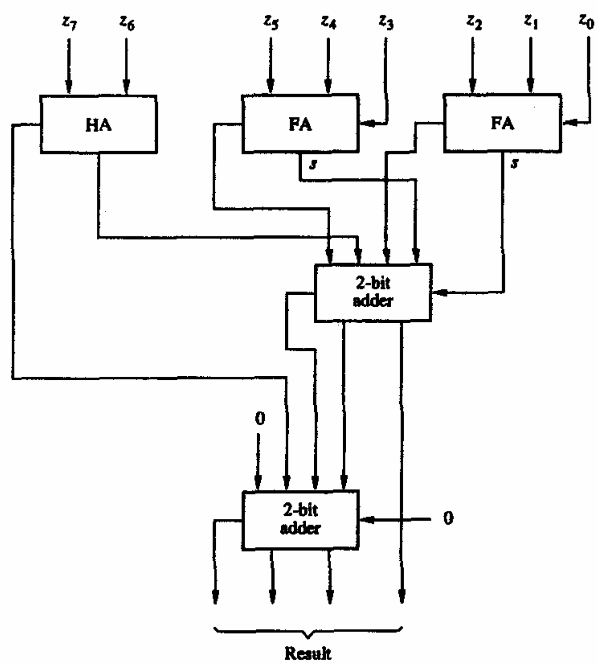
5.21. A full-adder circuit can be used, such that two of the bits of the number are connected as inputs x and y, while the third bit is connected as the carry-in. Then, the carry-out and sum bits will indicate how many input bits are equal to 1.



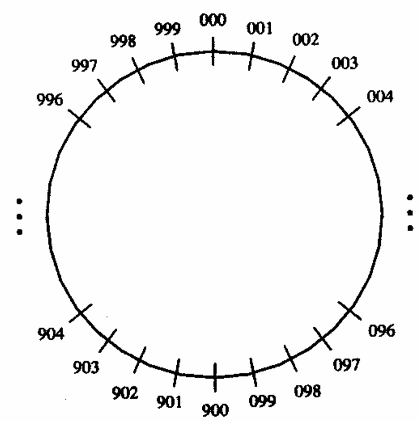
5.22. Using the approach explained in the solution to problem 5.21, the desired circuit can be built as follows:



5.23. Using the approach explained in the solutions to problems 5.21 and 5.22, the desired circuit can be built as follows:



5.24. The graphical representation is



For example, the addition $-3 + (+5) = 2$ involves starting at 997 (= −3) and going clockwise 5 numbers, which gives the result 002 (= +2). Similarly, the subtraction $4 - (+8) = -4$ involves starting at 004 (= +4) and going counterclockwise 8 numbers, which gives the result 996 (= −4).

5.25. The ternary half-adder in Figure P5.3 can be defined using binary-encoded signals as follows:

A		B		Carry	Sum	
a_1	a_0	b_1	b_0	c_{out}	s_1	s_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	0	0	1	0
1	0	0	1	1	0	0
1	0	1	0	1	0	1

The remaining 7 (out of 16) valuations, where either $a_1 = a_0 = 1$, or $b_1 = b_0 = 1$, can be treated as don't care conditions. Then, the minimum cost expressions are:

$$
\begin{aligned}
c_{out} &= a_0 b_1 + a_1 b_1 + a_1 b_0 \\
s_1 &= a_0 b_0 + \overline{a}_1 \overline{a}_0 b_1 + a_1 \overline{b}_1 \overline{b}_0 \\
s_0 &= a_1 b_1 + \overline{a}_1 \overline{a}_0 b_0 + a_0 \overline{b}_1 \overline{b}_0
\end{aligned}
$$

26. Ternary full-adder is defined by the truth table:

c_{in}	A	B	c_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	0	2	0	2
0	1	0	0	1
0	1	1	0	2
0	1	2	1	0
0	2	0	0	2
0	2	1	1	0
0	2	2	1	1
1	0	0	0	1
1	0	1	0	2
1	0	2	1	0
1	1	0	0	2
1	1	1	1	0
1	1	2	1	1
1	2	0	1	0
1	2	1	1	1
1	2	2	1	2

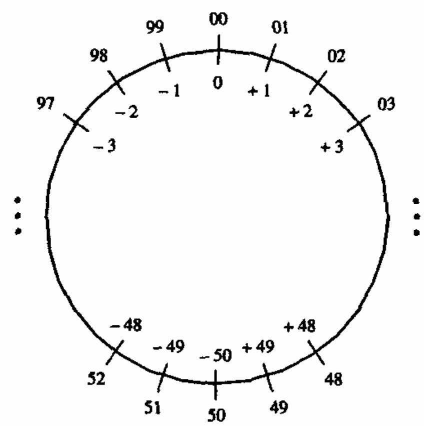
Using binary-encoded signals for this full-adder gives the following truth table:

	A		B			Sum	
c_{in}	a_1	a_0	b_1	b_0	c_{out}	s_1	s_0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	0	1	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	0	0	1	0
1	0	1	0	1	1	0	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	0	1	1	0

Treating the 14 (out of 32) valuations where either $a_1 = a_0 = 1$ or $b_1 = b_0 = 1$ as don't care conditions, leads to the minimum cost expressions

$$\begin{aligned}
c_{out} &= a_0b_1 + a_1b_0 + a_1b_1 + a_1c_{in} + b_1c_{in} + a_0b_0c_{in} \\
s_1 &= a_0b_0\bar{c}_{in} + \bar{a}_1\bar{a}_0b_1\bar{c}_{in} + a_1\bar{b}_1\bar{b}_0\bar{c}_{in} + a_1b_1c_{in} + \bar{a}_1\bar{a}_0b_0c_{in} + a_0\bar{b}_1\bar{b}_0c_{in} \\
s_0 &= a_1b_1\bar{c}_{in} + \bar{a}_1\bar{a}_0b_0\bar{c}_{in} + a_0\bar{b}_1\bar{b}_0\bar{c}_{in} + a_1b_0c_{in} + a_0b_1c_{in} + \bar{a}_1\bar{a}_0\bar{b}_1\bar{b}_0c_{in}
\end{aligned}$$

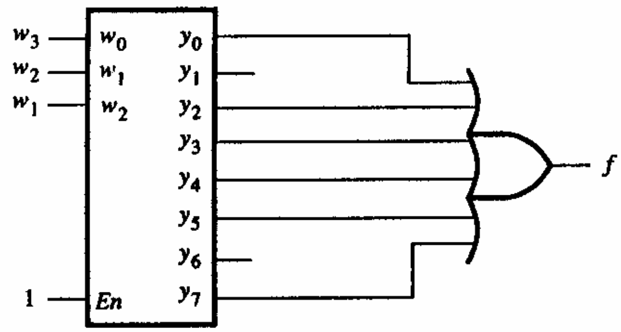
5.27. The subtractions $26 - 27 = 99$ and $18 - 34 = 84$ make sense if the two-digit numbers 00 to 99 are interpreted so that the numbers 00 to 49 are positive integers from 0 to +49, while the numbers 50 to 99 are negative integers from −50 to −1. This scheme can be illustrated graphically as follows:



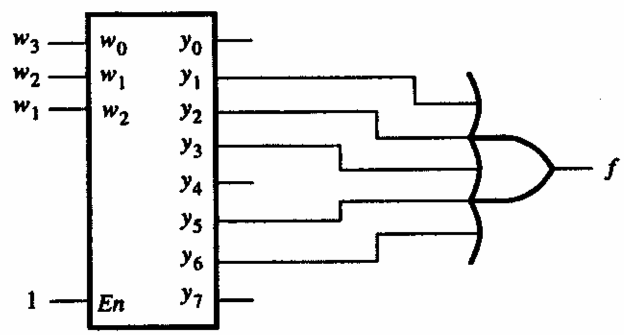
Thanks.

Chapter 6

6.1.



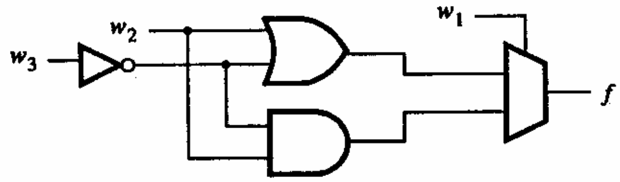
6.2.



6.3.

w_1	w_2	w_3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

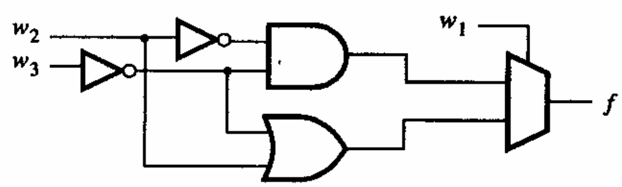
w_1	f
0	$w_2 + \overline{w}_3$
1	$w_2\overline{w}_3$



6.4.

w_1	w_2	w_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

w_1	f
0	$\overline{w}_2\overline{w}_3$
1	$w_2 + \overline{w}_3$



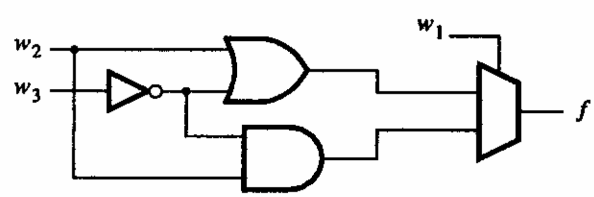
6.5. The function f can be expressed as

$$f = \overline{w}_1\overline{w}_2\overline{w}_3 + \overline{w}_1 w_2\overline{w}_3 + \overline{w}_1 w_2 w_3 + w_1 w_2\overline{w}_3$$

Expansion in terms of w_1 produces

$$f = \overline{w}_1(w_2 + \overline{w}_3) + w_1(w_2\overline{w}_3)$$

The corresponding circuit is



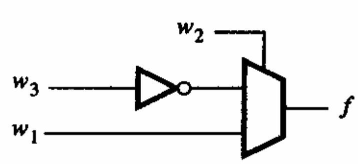
6.6. The function f can be expressed as

$$f = \overline{w}_1\overline{w}_2\overline{w}_3 + w_1\overline{w}_2\overline{w}_3 + w_1 w_2\overline{w}_3 + w_1 w_2 w_3$$

Expansion in terms of w_2 produces

$$f = \overline{w}_2(\overline{w}_3) + w_2(w_1)$$

The corresponding circuit is



6.7. Expansion in terms of w_2 gives

$$\begin{aligned} f &= \overline{w}_2(1 + \overline{w}_1\overline{w}_3 + w_1w_3) + w_2(\overline{w}_1\overline{w}_3 + w_1w_3) \\ &= \overline{w}_1\overline{w}_2\overline{w}_3 + w_1\overline{w}_2w_3 + \overline{w}_2 + \overline{w}_1w_2\overline{w}_3 + w_1w_2w_3 \end{aligned}$$

Further expansion in terms of w_1 gives

$$\begin{aligned} f &= \overline{w}_1(w_2\overline{w}_3 + \overline{w}_2\overline{w}_3 + \overline{w}_2) + w_1(w_2w_3 + \overline{w}_2w_3 + \overline{w}_2) \\ &= \overline{w}_1w_2\overline{w}_3 + \overline{w}_1\overline{w}_2\overline{w}_3 + \overline{w}_1\overline{w}_2 + w_1w_2w_3 + w_1\overline{w}_2w_3 + w_1\overline{w}_2 \end{aligned}$$

Further expansion in terms of w_3 gives

$$\begin{aligned} f &= \overline{w}_3(\overline{w}_1w_2 + \overline{w}_1\overline{w}_2 + \overline{w}_1\overline{w}_2 + w_1\overline{w}_2) + w_3(w_1w_2 + w_1\overline{w}_2 + w_1\overline{w}_2 + \overline{w}_1\overline{w}_2) \\ &= \overline{w}_1w_2\overline{w}_3 + \overline{w}_1\overline{w}_2\overline{w}_3 + w_1\overline{w}_2\overline{w}_3 + w_1w_2w_3 + w_1\overline{w}_2w_3 + \overline{w}_1\overline{w}_2w_3 \end{aligned}$$

6.8. Expansion in terms of w_1 gives

$$f = \overline{w}_1w_2 + \overline{w}_1\overline{w}_3 + w_1w_2$$

Further expansion in terms of w_2 gives

$$\begin{aligned} f &= \overline{w}_2(\overline{w}_1\overline{w}_3) + w_2(w_1 + \overline{w}_1 + \overline{w}_1\overline{w}_3) \\ &= \overline{w}_1w_2 + \overline{w}_1w_2\overline{w}_3 + \overline{w}_1\overline{w}_2\overline{w}_3 + w_1w_2 \end{aligned}$$

Further expansion in terms of w_3 gives

$$\begin{aligned} f &= \overline{w}_3(\overline{w}_1\overline{w}_2 + w_1w_2 + \overline{w}_1w_2 + \overline{w}_1w_2) + w_3(w_1w_2 + \overline{w}_1w_2) \\ &= \overline{w}_1\overline{w}_2\overline{w}_3 + w_1w_2\overline{w}_3 + \overline{w}_1w_2\overline{w}_3 + \overline{w}_1w_2w_3 + w_1w_2w_3 \end{aligned}$$

6.9. Proof of Shannon's expansion theorem

$$f(x_1, x_2, ..., x_n) = \overline{x}_1 \cdot f(0, x_2, ..., x_n) + x_1 \cdot f(1, x_2, ..., x_n)$$

This theorem can be proved using *perfect induction*, by showing that the expression is true for every possible value of x_1. Since x_1 is a boolean variable, we need to look at only two cases: $x_1 = 0$ and $x_1 = 1$.

Setting $x_1 = 0$ in the above expression, we have:

$$\begin{aligned} f(0, x_2, ..., x_n) &= 1 \cdot f(0, x_2, ..., x_n) + 0 \cdot f(1, x_2, ..., x_n) \\ &= f(0, x_2, ..., x_n) \end{aligned}$$

Setting $x_1 = 1$, we have:

$$\begin{aligned} f(1, x_2, ..., x_n) &= 0 \cdot f(0, x_2, ..., x_n) + 1 \cdot f(1, x_2, ..., x_n) \\ &= f(1, x_2, ..., x_n) \end{aligned}$$

This proof can be performed for any arbitrary x_i in the same manner.

6.10. Derivation using $\overline{f}$:

$$\begin{aligned} \overline{f} &= \overline{w}\overline{f}_{\overline{w}} + w\overline{f}_w \\ f &= \left(\overline{\overline{w}\overline{f}_{\overline{w}} + w\overline{f}_w}\right) \\ &= \left(\overline{\overline{w}\overline{f}_{\overline{w}}}\right) \cdot \left(\overline{w\overline{f}_w}\right) \\ &= (w + f_{\overline{w}})(\overline{w} + f_w) \end{aligned}$$

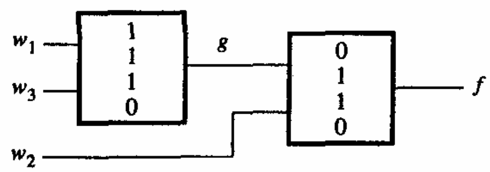
6.11. Expansion in terms of w_2 gives

$$f = \overline{w}_2(\overline{w}_1 + \overline{w}_3) + w_2(w_1 w_3)$$

Letting $g = \overline{w}_1 + \overline{w}_3$, we have

$$f = \overline{w}_2 g + w_2 \overline{g}$$

The corresponding circuit is



6.12. Expansion of f in terms of w_2 gives

$$\begin{aligned} f &= \overline{w}_2(\overline{w}_1 + \overline{w}_3) + w_2(w_1 w_3) \\ &= w_2 \oplus (\overline{w}_1 + \overline{w}_3) \\ &= w_2 \oplus \overline{w_1 w_3} \end{aligned}$$

The cost of this multilevel circuit is 2 gates + 4 inputs = 6.

6.13. Using Shannon's expansion in terms of w_2 we have

$$\begin{aligned} f &= \overline{w}_2(\overline{w}_3 + \overline{w}_1 w_4) + w_2(w_3 \overline{w}_4 + w_1 w_3) \\ &= \overline{w}_2(\overline{w}_3 + \overline{w}_1 w_4) + w_2(w_3(w_1 + \overline{w}_4)) \end{aligned}$$

If we let $g = \overline{w}_3 + \overline{w}_1 w_4$, then

$$f = \overline{w}_2 g + w_2 \overline{g}$$

Thus, two 3-LUTs are needed to implement f.

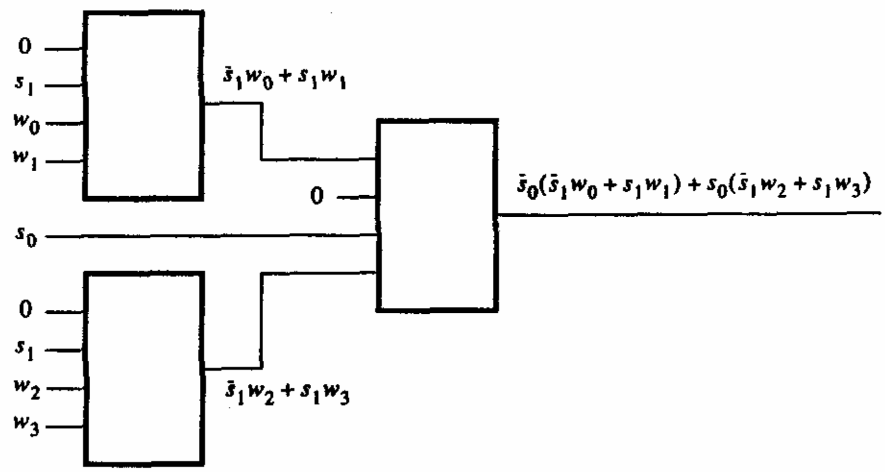
6.14. Any number of 5-variable functions can be implemented by using two 4-LUTs. For example, if we cascade the two 4-LUTs by connecting the output of one 4-LUT to an input of the other, then we can realize any function of the form

$$\begin{aligned} f &= f_1(w_1, w_2, w_3, w_4) + w_5 \\ f &= f_1(w_1, w_2, w_3, w_4) \cdot w_5 \end{aligned}$$

6.15. Expressing f in the form

$$\begin{aligned} f &= \overline{s}_1 \overline{s}_0 w_0 + s_1 \overline{s}_0 w_1 + \overline{s}_1 s_0 w_2 + s_1 s_0 w_3 \\ &= \overline{s}_0(\overline{s}_1 w_0 + s_1 w_1) + s_0(\overline{s}_1 w_2 + s_1 w_3) \end{aligned}$$

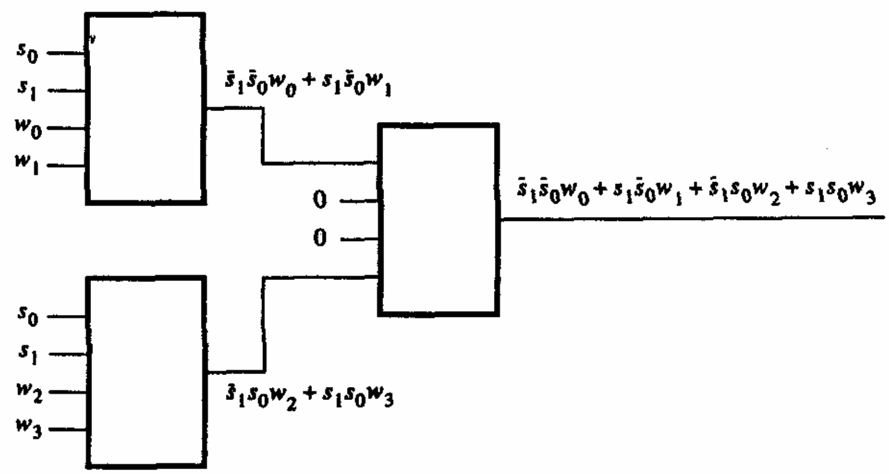
leads to the circuit.



Alternatively, directly using the expression

$$f = \overline{s}_1\overline{s}_0 w_0 + s_1\overline{s}_0 w_1 + \overline{s}_1 s_0 w_2 + s_1 s_0 w_3$$

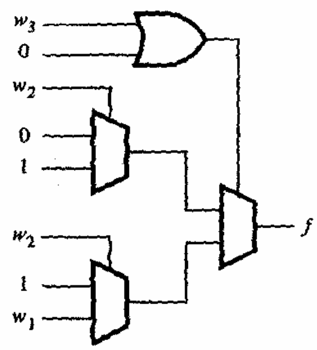
leads to the circuit.



6.16. Using Shannon's expansion in terms of w_3 we have

$$\begin{aligned} f &= \overline{w}_3(w_2) + w_3(w_1 + \overline{w}_2) \\ &= \overline{w}_3(w_2) + w_3(\overline{w}_2 + w_2 w_1) \end{aligned}$$

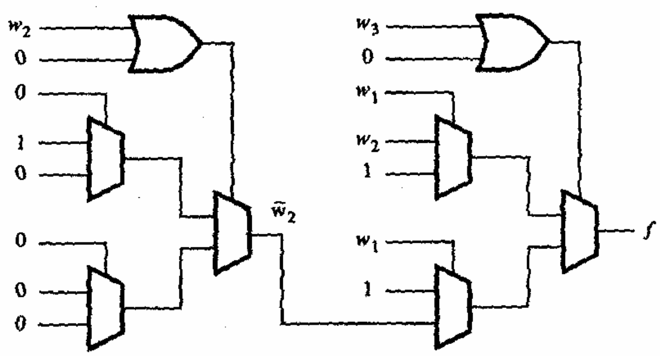
The corresponding circuit is



6.17. *Using Shannon's expansion in terms of* w_3 we have

$$f = w_3(\overline{w}_1 + w_1\overline{w}_2) + \overline{w}_3(w_1 + \overline{w}_1 w_2)$$

The corresponding circuit is



6.18. The code in Figure P6.2 is a 2-to-4 decoder with an enable input. This style of code is a poor choice because its meaning is not readily apparent. Better choices of code that represents a 2-to-4 decoder are shown in Figures 6.30 and 6.46.

6.19.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_19 IS
    PORT ( w  : IN    STD_LOGIC_VECTOR(1 TO 3) ;
           f  : OUT  STD_LOGIC ) ;
END prob6_19 ;

ARCHITECTURE Behavior OF prob6_19 IS
BEGIN
    WITH w SELECT
        f <= '0' WHEN "001",
             '0' WHEN "110",
             '1' WHEN OTHERS ;
END Behavior ;
```

6.20.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_20 IS
    PORT ( w  : IN    STD_LOGIC_VECTOR(1 TO 3) ;
           f  : OUT   STD_LOGIC ) ;
END prob6_20 ;

ARCHITECTURE Behavior OF prob6_20 IS
BEGIN
    WITH w SELECT
        f <= '0' WHEN "000",
             '0' WHEN "100",
             '0' WHEN "111",
             '1' WHEN OTHERS ;
END Behavior ;
```

6.21.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_21 IS
    PORT ( w  : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           y  : OUT   STD_LOGIC_VECTOR(1 DOWNTO 0) ) ;
END prob6_21 ;

ARCHITECTURE Behavior OF prob6_21 IS
BEGIN
    WITH w SELECT
        y <= "00" WHEN "0001",
             "01" WHEN "0010",
             "10" WHEN "0100",
             "11" WHEN OTHERS ;
END Behavior ;
```

6.22.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_22 IS
    PORT ( w  : IN    STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           y  : OUT   STD_LOGIC_VECTOR(2 DOWNTO 0) ) ;
END prob6_22 ;
```

. . . con't

```
ARCHITECTURE Behavior OF prob6_22 IS
BEGIN
    y <= "000" WHEN w = "00000001" ELSE
         "001" WHEN w = "00000010" ELSE
         "010" WHEN w = "00000100" ELSE
         "011" WHEN w = "00001000" ELSE
         "100" WHEN w = "00010000" ELSE
         "101" WHEN w = "00100000" ELSE
         "110" WHEN w = "01000000" ELSE
         "111" ;
END Behavior ;
```

6.23. First define a set of intermediate variables

$$
\begin{aligned}
i_0 &= \overline{w}_7\overline{w}_6\overline{w}_5\overline{w}_4\overline{w}_3\overline{w}_2\overline{w}_1 w_0 \\
i_1 &= \overline{w}_7\overline{w}_6\overline{w}_5\overline{w}_4\overline{w}_3\overline{w}_2 w_1 \\
i_2 &= \overline{w}_7\overline{w}_6\overline{w}_5\overline{w}_4\overline{w}_3 w_2 \\
i_3 &= \overline{w}_7\overline{w}_6\overline{w}_5\overline{w}_4 w_3 \\
i_4 &= \overline{w}_7\overline{w}_6\overline{w}_5 w_4 \\
i_5 &= \overline{w}_7\overline{w}_6 w_5 \\
i_6 &= \overline{w}_7 w_6 \\
i_7 &= w_7
\end{aligned}
$$

Now a traditional binary encoder can be used for the priority encoder

$$
\begin{aligned}
y_0 &= i_1 + i_3 + i_5 + i_7 \\
y_1 &= i_2 + i_3 + i_6 + i_7 \\
y_2 &= i_4 + i_5 + i_6 + i_7
\end{aligned}
$$

6.24.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_24 IS
    PORT ( w  : IN   STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           y  : OUT  STD_LOGIC_VECTOR(2 DOWNTO 0) ;
           z  : OUT  STD_LOGIC ) ;
END prob6_24 ;

ARCHITECTURE Behavior OF prob6_24 IS
BEGIN
    y <= "111" WHEN w(7) = '1' ELSE
         "110" WHEN w(6) = '1' ELSE
         "101" WHEN w(5) = '1' ELSE
         "100" WHEN w(4) = '1' ELSE
         "011" WHEN w(3) = '1' ELSE
         "010" WHEN w(2) = '1' ELSE
         "001" WHEN w(1) = '1' ELSE
         "000" ;
    z <= '0' WHEN w="00000000" ELSE '1' ;
END Behavior ;
```

6.25.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_25 IS
    PORT ( w  : IN   STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           y  : OUT  STD_LOGIC_VECTOR(2 DOWNTO 0) ;
           z  : OUT  STD_LOGIC ) ;
END prob6_25 ;

ARCHITECTURE Behavior OF prob6_25 IS
BEGIN
    PROCESS ( w )
    BEGIN
        IF w(7) = '1' THEN
            y <= "111" ;
        ELSIF w(6) = '1' THEN
            y <= "110" ;
        ELSIF w(5) = '1' THEN
            y <= "101" ;
        ELSIF w(4) = '1' THEN
            y <= "100" ;
        ELSIF w(3) = '1' THEN
            y <= "011" ;
        ELSIF w(2) = '1' THEN
            y <= "010" ;
        ELSIF w(1) = '1' THEN
            y <= "001" ;
        ELSE
            y <= "000" ;
        END IF ;
        IF w = "00000000" THEN
            z <= '0' ;
        ELSE
            z <= '1' ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

6.26.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY if2to4 IS
    PORT ( w   : IN   STD_LOGIC_VECTOR(1 DOWNTO 0) ;
           En  : IN   STD_LOGIC ;
           y   : OUT  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END if2to4 ;
```

. . . con't

```
ARCHITECTURE Behavior OF if2to4 IS
BEGIN
    PROCESS ( En, w )
    BEGIN
        IF En = '0' THEN
            y <= "0000" ;
        ELSE
            IF w = "00" THEN
                y <= "0001" ;
            ELSIF w = "01" THEN
                y <= "0010" ;
            ELSIF w = "10" THEN
                y <= "0100" ;
            ELSE
                y <= "1000" ;
            END IF ;
        END IF ;
    END PROCESS ;
END Behavior ;

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
PACKAGE if2to4_package IS
    COMPONENT if2to4
        PORT ( w    : IN    STD_LOGIC_VECTOR(1 DOWNTO 0) ;
               En   : IN    STD_LOGIC ;
               y    : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
    END COMPONENT ;
END if2to4_package ;


LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.if2to4_package.all ;

ENTITY h3to8 IS
    PORT ( w    : IN    STD_LOGIC_VECTOR(2 DOWNTO 0) ;
           En   : IN    STD_LOGIC ;
           y    : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END h3to8 ;

ARCHITECTURE Structure OF h3to8 IS
    SIGNAL EnableTop, EnableBot : STD_LOGIC ;
BEGIN
    EnableTop <= w(2) AND En ;
    EnableBot <= (NOT w(2)) AND En ;

    Decoder1: if2to4 PORT MAP ( w( 1 DOWNTO 0 ), EnableBot, y( 3 DOWNTO 0 ) ) ;
    Decoder2: if2to4 PORT MAP ( w( 1 DOWNTO 0 ), EnableTop, y( 7 DOWNTO 4 ) ) ;
END Structure ;
```

. . . con't

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
PACKAGE h3to8_package IS
    COMPONENT h3to8
        PORT ( w    : IN    STD_LOGIC_VECTOR(2 DOWNTO 0) ;
               En   : IN    STD_LOGIC ;
               y    : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
    END COMPONENT ;
END h3to8_package ;
```

6.27.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.h3to8_package.all ;

ENTITY h6to64 IS
    PORT ( w    : IN    STD_LOGIC_VECTOR( 5 DOWNTO 0 ) ;
           En   : IN    STD_LOGIC ;
           y    : OUT   STD_LOGIC_VECTOR(63 DOWNTO 0 ) ) ;
END h6to64 ;

ARCHITECTURE Structure OF h6to64 IS
    SIGNAL Enables : STD_LOGIC_VECTOR( 7 DOWNTO 0 ) ;
BEGIN
    root : h3to8 PORT MAP ( w( 5 DOWNTO 3 ), En, Enables ) ;
    leaf0: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 0 ), y( 7 DOWNTO 0 ) ) ;
    leaf1: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 1 ), y( 15 DOWNTO 8 ) ) ;
    leaf2: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 2 ), y( 23 DOWNTO 16 ) ) ;
    leaf3: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 3 ), y( 31 DOWNTO 24 ) ) ;
    leaf4: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 4 ), y( 39 DOWNTO 32 ) ) ;
    leaf5: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 5 ), y( 47 DOWNTO 40 ) ) ;
    leaf6: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 6 ), y( 55 DOWNTO 48 ) ) ;
    leaf7: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 7 ), y( 63 DOWNTO 56 ) ) ;
END Structure ;
```

6.28.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_28 IS
    PORT ( s    : IN    STD_LOGIC_VECTOR( 1 DOWNTO 0 ) ;
           w    : IN    STD_LOGIC_VECTOR( 3 DOWNTO 0 ) ;
           f    : OUT   STD_LOGIC ) ;
END prob6_28 ;
```

. . . con't

```
ARCHITECTURE Structure OF prob6_28 IS
    COMPONENT dec2to4
        PORT ( w   : IN    STD_LOGIC_VECTOR(1 DOWNTO 0) ;
               En  : IN    STD_LOGIC ;
               y   : OUT   STD_LOGIC_VECTOR(0 TO 3) );
    END COMPONENT;
    SIGNAL High : STD_LOGIC ;
    SIGNAL y : STD_LOGIC_VECTOR( 3 DOWNTO 0 ) ;
BEGIN
    High <= '1' ;
    decoder: dec2to4 PORT MAP ( s, High, y ) ;
    f <= (w(0) AND y(0)) OR (w(1) AND y(1)) OR
        (w(2) AND y(2)) OR w(3) AND y(3) ) ;
END Structure ;
```

6.30.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_30 IS
    PORT ( bcd  : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           leds : OUT   STD_LOGIC_VECTOR(1 TO 7) ) ;
END prob6_30 ;

ARCHITECTURE Behavior OF prob6_30 IS
BEGIN
    WITH bcd SELECT
        --        abcdefg
        leds <= "1111110" WHEN "0000",
                "0110000" WHEN "0001",
                "1101101" WHEN "0010",
                "1111001" WHEN "0011",
                "0110011" WHEN "0100",
                "1011011" WHEN "0101",
                "1011111" WHEN "0110",
                "1110000" WHEN "0111",
                "1111111" WHEN "1000",
                "1111011" WHEN "1001",
                "-------" WHEN OTHERS ;
END Behavior ;
```

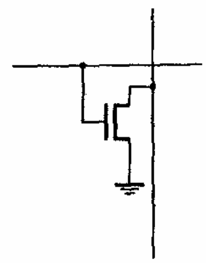
6.31.

$$
\begin{aligned}
a &= w_3 + w_2 w_0 + w_1 + \overline{w}_2 \overline{w}_0 \\
b &= w_3 + \overline{w}_1 \overline{w}_0 + w_1 w_0 + \overline{w}_2 \\
c &= w_2 + \overline{w}_1 + w_0
\end{aligned}
$$

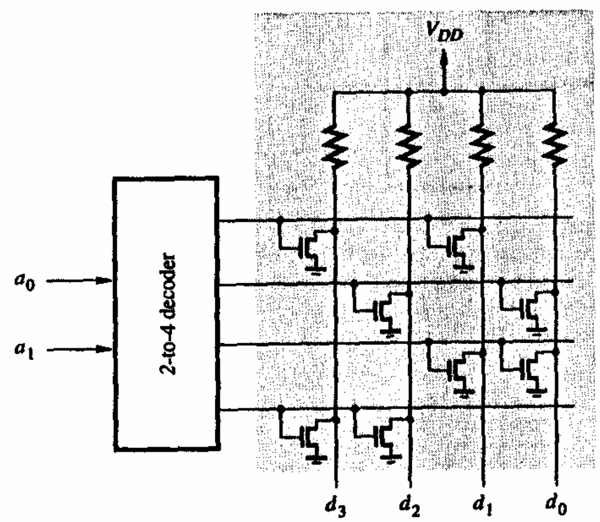
6.32.

$$
\begin{aligned}
d &= w_3 + \overline{w}_2 \overline{w}_0 + w_1 \overline{w}_0 + w_2 \overline{w}_1 w_0 + \overline{w}_2 w_1 \\
e &= \overline{w}_2 \overline{w}_0 + w_1 \overline{w}_0 \\
f &= w_3 + \overline{w}_1 \overline{w}_0 + w_2 \overline{w}_0 + w_2 \overline{w}_1 \\
g &= w_3 + w_1 \overline{w}_0 + w_2 \overline{w}_1 + \overline{w}_2 w_1
\end{aligned}
$$

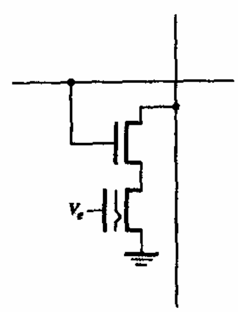
6.33. (*a*) Each ROM location that should store a 1 requires no circuitry, because the pull-up resistors provides the default value of 1. Each location that stores a 0 has the following cell



(*b*)

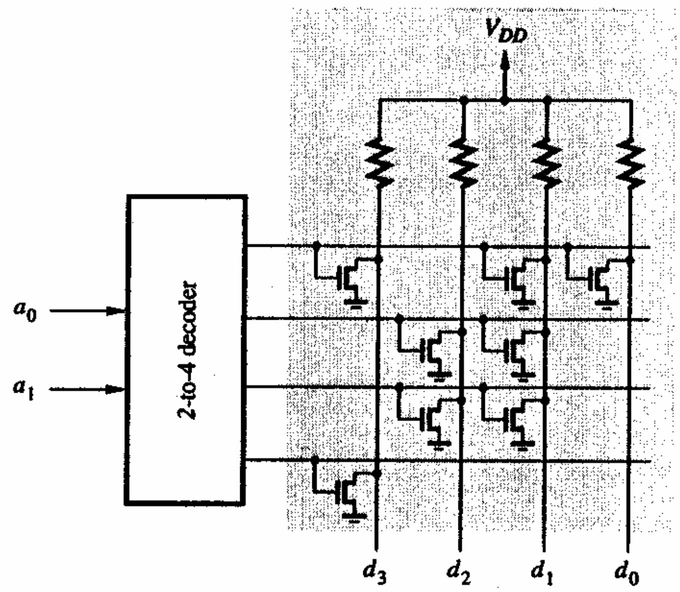


(c) Every location in the ROM contains the following cell



If a location should store a 1, then the corresponding EEPROM transistor is programmed to be turned off. But if the location should store a 0, then the EEPROM transistor is left unprogrammed.

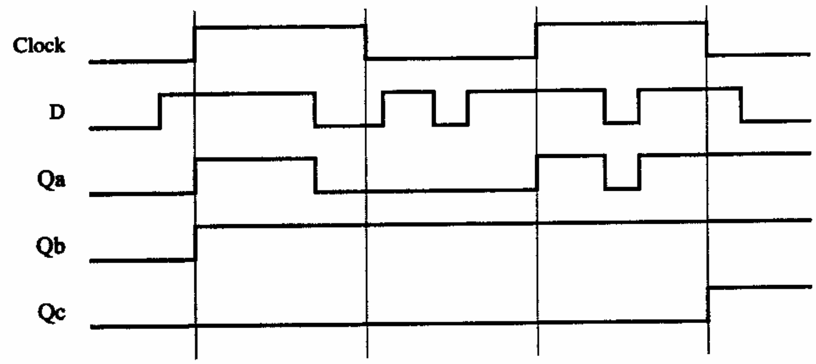
(*d*)



Thank you for visiting my web site. Make sure that you do contact me if you have any discrepancy about the material presented in my web site.

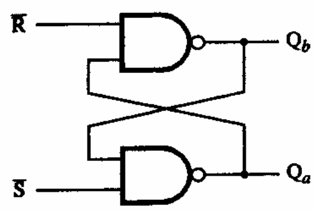
Chapter 7

7.1.



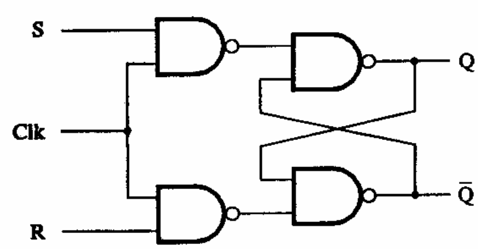
7.2. The circuit in Figure 7.3 can be modified to implement an SR latch by connecting S to the *Data* input and $S + R$ to the *Load* input. Thus the value of S is loaded into the latch whenever either S or R is asserted. Care must be taken to ensure that the *Data* signal remains stable while the *Load* signal is asserted.

7.3.

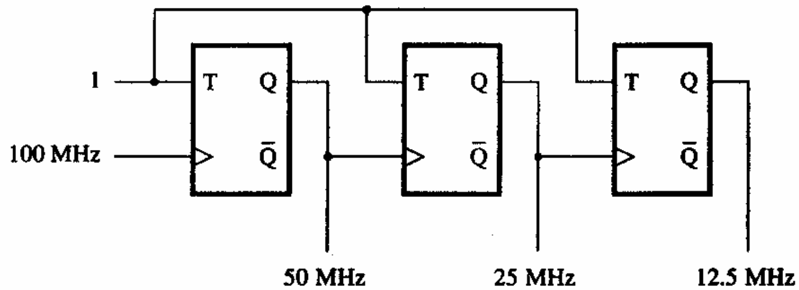


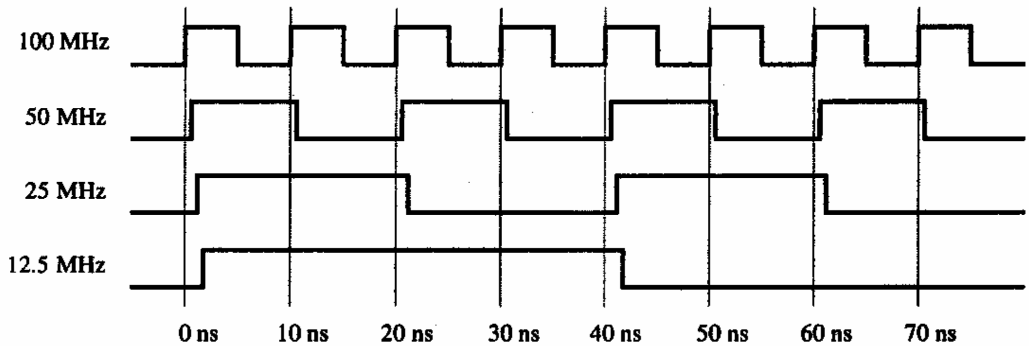
$\overline{S}$	$\overline{R}$	Q_a	Q_b	
1	1	0/1	1/0	(no change)
1	0	0	1	
0	1	1	0	
0	0	1	1	

7.4.

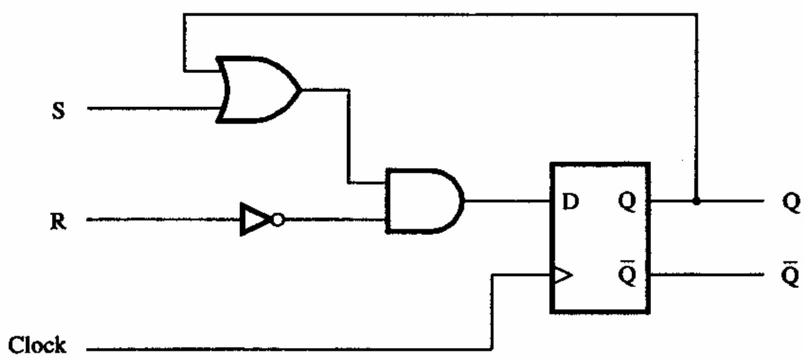


7.5.

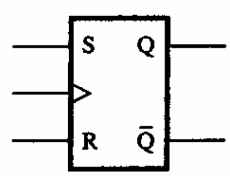




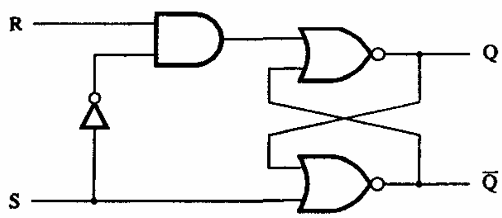
7.6.



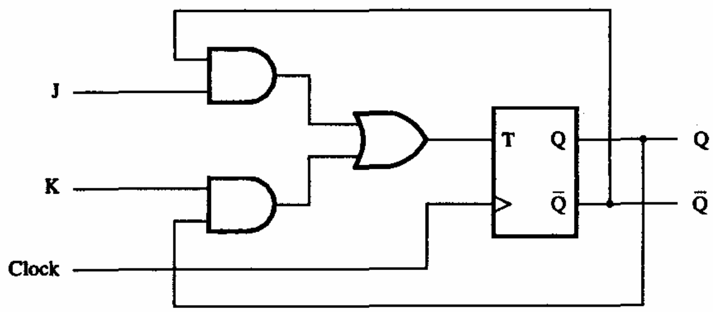
S	R	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	0



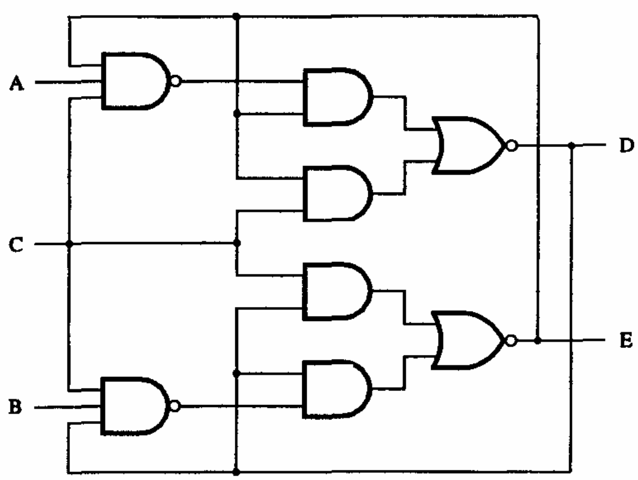
7.7.



7.8.



7.9. As the circuit in Figure P7.2 is drawn, it is not a useful flip-flop circuit, because setting $C = 0$ results in both of the circuit outputs being set to 0. Consider the slightly modified circuit shown below:



This modified circuit acts as a negative-edge-triggered JK flip-flop, in which $J = A, K = B$, *Clock* $=$ C, $\mathrm{Q} = D$, and $\overline{\mathrm{Q}} = E$. This circuit is found in the standard chip called 74LS107A (plus a *Clear* input, which is not shown).

7.10.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob7_10 IS
    PORT ( T, Resetn, Clock   : IN    STD_LOGIC ;
           Q                  : OUT   STD_LOGIC ) ;
END prob7_10 ;

ARCHITECTURE Behavior OF prob7_10 IS
    SIGNAL Qint : STD_LOGIC ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Qint <= '0' ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF T = '1' THEN
                Qint <= NOT Qint ;
            ELSE
                Qint <= Qint ;
            END IF ;
        END IF ;
    END PROCESS ;
    Q <= Qint ;
END Behavior ;
```

7.11.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob7_11 IS
    PORT ( J, K, Resetn, Clock   : IN    STD_LOGIC ;
           Q                     : OUT   STD_LOGIC ) ;
END prob7_11 ;

ARCHITECTURE Behavior OF prob7_11 IS
    SIGNAL Qint : STD_LOGIC ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Qint <= '0' ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Qint <= ( J AND NOT Qint ) OR ( NOT K AND Qint ) ;
        END IF ;
    END PROCESS ;
    Q <= Qint ;
END Behavior ;
```

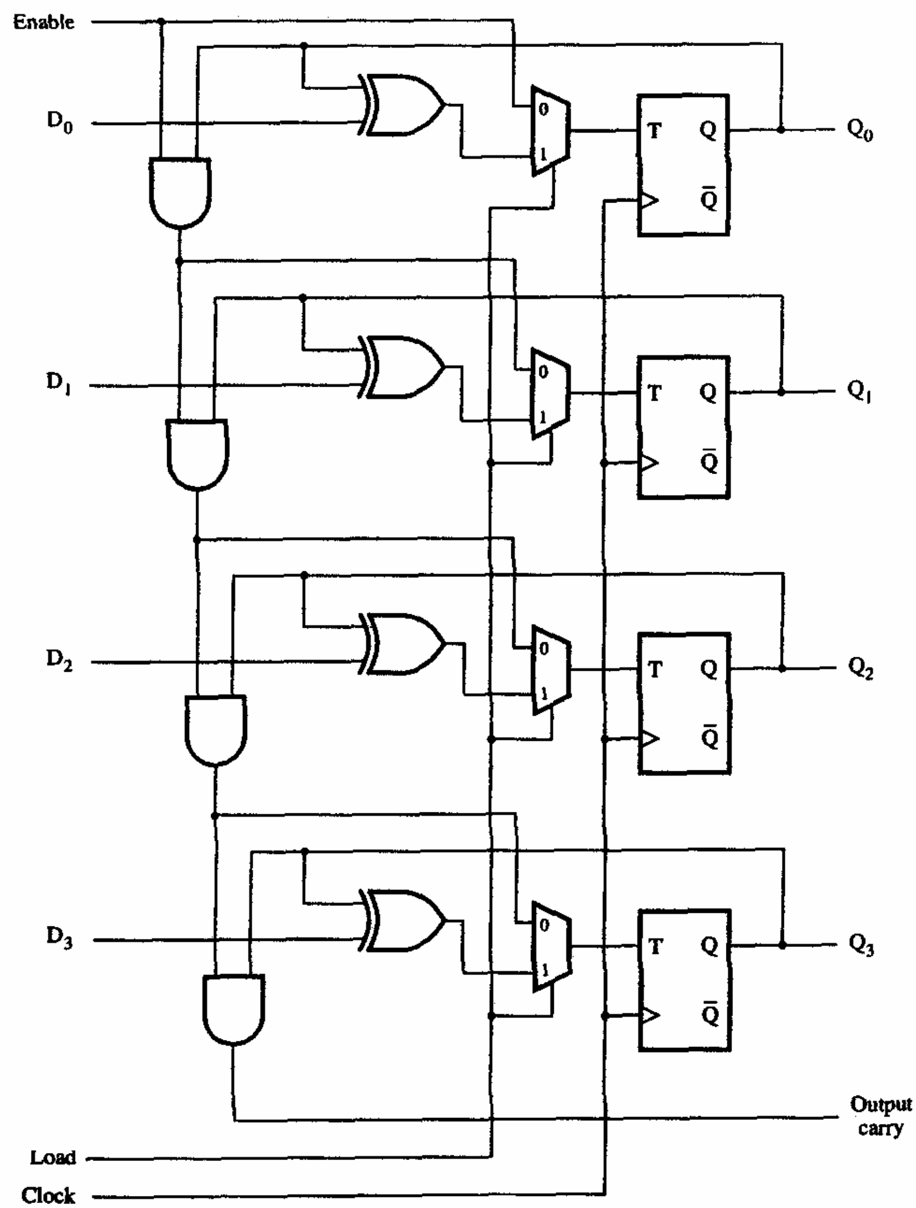
7.13. Let $S = s_1 s_0$ be a binary number that specifies the number of bit-positions to shift by. Also let L be a parallel-load input, and let $R = r_3 r_2 r_1 r_0$ be parallel data. If the inputs to the flip-flops are $D_0 \ldots D_3$ and the outputs are $Q_0 \ldots Q_3$, then the barrel-shifter can be represented by the logic expressions

$$\begin{aligned}
D_3 &= L \cdot R_3 + \overline{L} \cdot (\overline{s}_1 \overline{s}_0 q_3) \\
D_2 &= L \cdot R_2 + \overline{L} \cdot (\overline{s}_1 \overline{s}_0 q_2 + \overline{s}_1 s_0 q_3) \\
D_1 &= L \cdot R_1 + \overline{L} \cdot (\overline{s}_1 \overline{s}_0 q_1 + \overline{s}_1 s_0 q_2 + s_1 \overline{s}_0 q_3) \\
D_0 &= L \cdot R_0 + \overline{L} \cdot (\overline{s}_1 \overline{s}_0 q_0 + \overline{s}_1 s_0 q_1 + s_1 \overline{s}_0 q_2 + s_1 s_0 q_3)
\end{aligned}$$

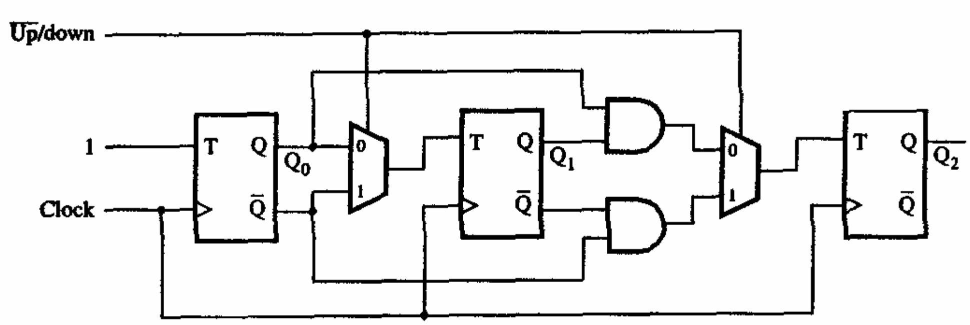
7.14.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob7_14 IS
    PORT ( R        : IN     STD_LOGIC_VECTOR (3 DOWNTO 0) ;
           Shift    : IN     STD_LOGIC_VECTOR (1 DOWNTO 0) ;
           L, Clock : IN     STD_LOGIC ;
           Q        : BUFFER STD_LOGIC_VECTOR (3 DOWNTO 0) ) ;
END prob7_14 ;

ARCHITECTURE Behavior OF prob7_14 IS
BEGIN
    PROCESS ( Clock )
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF L = '1' THEN
            Q <= R ;
        ELSE
            CASE Shift IS
                WHEN "10"     => Q <= "00" & Q(3 DOWNTO 2) ;
                WHEN "01"     => Q <= "0" & Q(3 DOWNTO 1) ;
                WHEN OTHERS => Q <= Q ;
            END CASE ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

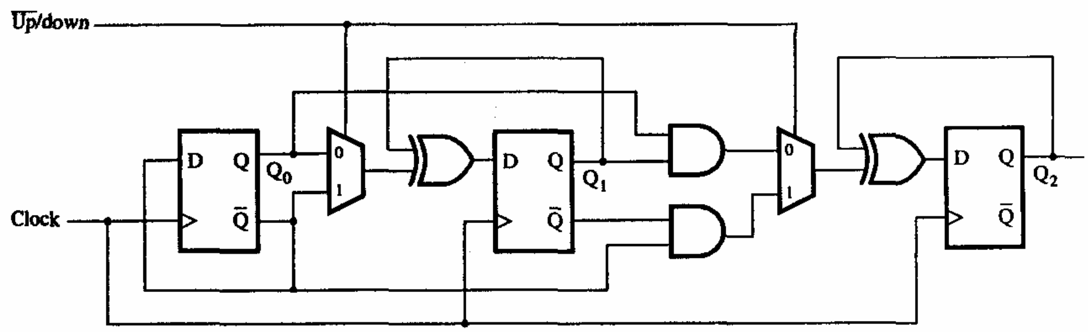
7.15.


Enable
D_0
D_1
D_2
D_3
Load
Clock
0
1
T
Q
$\bar{Q}$
Q_0
Q_1
Q_2
Q_3
Output carry

7.16.


$\overline{\text{Up}}$/down
1
Clock
T
Q
$\bar{Q}$
Q_0
Q_1
Q_2
0
1

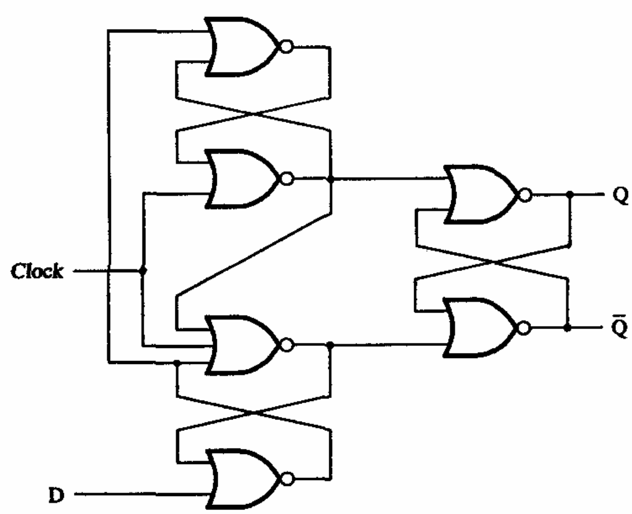
7.17.



7.18. The counting sequence is 000, 001, 010, 111.

7.19. The circuit in Figure P7.4 is a master-slave JK flip-flop. It suffers from a problem sometimes called *ones-catching*. Consider the situation where the Q output is low, *Clock* = 0, and $J = K = 0$. Now let *Clock* remain stable at 0 while J change from 0 to 1 and then back to 0. The master stage is now set to 1 and this value will be incorrectly transferred into the slave stage when the clock changes to 1.

7.20. Repeated application of DeMorgan's theorem can be used to change the positive-edge triggered D flip-flop in Figure 7.11 into the negative-edge D triggered flip-flop:



7.21.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY prob7_21 IS
    PORT ( R                  : IN      STD_LOGIC_VECTOR(23 DOWNTO 0) ;
           Clock, Resetn, L, U : IN      STD_LOGIC ;
           Q                  : BUFFER  STD_LOGIC_VECTOR(23 DOWNTO 0) ) ;
END prob7_21 ;

ARCHITECTURE Behavior OF prob7_21 IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0') ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF L = '1' THEN
                Q <= R ;
            ELSIF U = '1' THEN
                Q <= Q+1 ;
            ELSE
                Q <= Q-1 ;
            END IF ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.22.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY prob7_22 IS
    GENERIC ( N : INTEGER := 4 ) ;
    PORT ( Clock, Resetn, E  : IN   STD_LOGIC ;
           Q                 : OUT  STD_LOGIC_VECTOR ( N-1 DOWNTO 0) ) ;
END prob7_22 ;

ARCHITECTURE Behavior OF prob7_22 IS
    SIGNAL Count : STD_LOGIC_VECTOR ( N-1 DOWNTO 0 ) ;
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Count <= (OTHERS => '0') ;
```

. . . con't

```
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF E = '1' THEN
                Count <= Count + 1 ;
            ELSE
                Count <= Count ;
            END IF ;
        END IF ;
    END PROCESS ;
    Q <= Count ;
END Behavior ;
```

7.23.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob7_23 IS
    PORT ( R                  : IN      INTEGER RANGE 0 TO 11 ;
           Clock, Resetn, L   : IN      STD_LOGIC ;
           Q                  : BUFFER  INTEGER RANGE 0 TO 11 ) ;
END prob7_23 ;

ARCHITECTURE Behavior OF prob7_23 IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= 0 ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF L = '1' THEN
                Q <= R ;
            ELSE
                IF Q = 11 THEN
                    Q <= 0 ;
                ELSE
                    Q <= Q + 1 ;
                END IF ;
            END IF ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.24. The longest delay in the circuit is the from the output of FF_0 to the input of FF_3. This delay totals 5 ns. Thus the minimum period for which the circuit will operate reliably is

$$T_{min} = 5\text{ ns} + t_{su} = 8\text{ ns}$$

The maximum frequency is

$$F_{max} = 1/T_{min} = 125\text{ MHz}$$

7.25.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob7_25 IS
    PORT ( Clock, Clear   : IN        STD_LOGIC ;
           BCD0, BCD1 : BUFFER  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END prob7_25 ;

ARCHITECTURE Structure OF prob7_25 IS
    COMPONENT fig7_25
        PORT ( D                      : IN        STD_LOGIC_VECTOR(3 DOWNTO 0) ;
               Clock, Enable, Load    : IN        STD_LOGIC ;
               Q                      : BUFFER    STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
    END COMPONENT ;
    SIGNAL Load0, Load1 : STD_LOGIC ;
    SIGNAL Enab0, Enab1 : STD_LOGIC ;
    SIGNAL Zero         : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
BEGIN
    Zero   <= "0000" ;
    Enab0 <= '1' ;
    Enab1 <= BCD0(0) AND BCD0(3) ;

    Load0 <= Enab1 OR Clear ;
    Load1 <= (BCD1(0) AND BCD1(3)) OR Clear ;

    cnt0: fig7_25 PORT MAP ( Clock => Clock, Load => Load0, Enable => Enab0,
                             D => Zero, Q => BCD0 ) ;
    cnt1: fig7_25 PORT MAP ( Clock => Clock, Load => Load1, Enable => Enab1,
                             D => Zero, Q => BCD1 ) ;
END Structure ;
```

7.26.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob7_26 IS
    PORT ( Clock, Resetn   : IN        STD_LOGIC ;
           Q               : BUFFER    STD_LOGIC_VECTOR(0 TO 7) ) ;
END prob7_26 ;

ARCHITECTURE Behavior OF prob7_26 IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= "00000000" ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= (NOT Q(7)) & Q(0 TO 6) ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.27.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob7_27 IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT ( Clock, Start  : IN      STD_LOGIC ;
           Q             : BUFFER  STD_LOGIC_VECTOR(0 TO N-1) ) ;
END prob7_27 ;

ARCHITECTURE Behavior OF prob7_27 IS
BEGIN
    PROCESS ( Clock, Start )
    BEGIN
        IF Start = '1' THEN
            Q <= (OTHERS => '0') ;
            Q(0) <= '1' ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            GenBits: FOR i IN 1 TO N-1 LOOP
                Q(i) <= Q(i-1) ;
            END LOOP ;
            Q(0) <= Q(N-1) ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.28.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY prob7_28 IS
    PORT ( Clock, Reset  : IN      STD_LOGIC ;
           Data          : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Q             : BUFFER  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END prob7_28;

ARCHITECTURE Behavior OF prob7_28 IS
BEGIN
    PROCESS ( Clock, Reset )
    BEGIN
        IF Reset = '1' THEN
            Q <= "0000" ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= Q + Data ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.29.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
LIBRARY lpm ;
USE lpm.lpm_components.all ;

ENTITY prob7_29 IS
    PORT ( Clock, Reset  : IN     STD_LOGIC ;
           Q             : OUT    STD_LOGIC_VECTOR(31 DOWNTO 0) ) ;
END prob7_29 ;

ARCHITECTURE Structural OF prob7_29 IS
BEGIN
    cnt: lpm_counter
        GENERIC MAP ( lpm_width => 32 )
        PORT MAP ( clock => Clock, aclr => Reset, q => Q ) ;
END Structural ;
```

7.33.

	T_1	T_2	T_3
(Swap): I_4	$R_{out} = X, T_{in}$	$R_{out} = Y, R_{in} = X$	$T_{out}, R_{in} = Y,$ *Done*

Since the processor now has five operations a 3-to-8 decoder is needed to decode the signals f_2, f_1, f_0. The SWAP operation is represented by the code

$$I_4 = f_2\overline{f}_1\overline{f}_0$$

New expressions are needed for R_{in} and R_{out} to accommodate the SWAP operation:

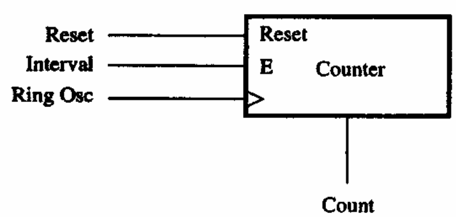
$$\begin{aligned} Rk_{in} &= (I_0 + I_1) \cdot T_1 \cdot X_k + (I_2 + I_3) \cdot T_3 \cdot X_k + I_4 \cdot T_2 \cdot X_k + I_4 \cdot T_3 \cdot Y_k \\ Rk_{out} &= I_1 \cdot T_1 \cdot Y_k + (I_2 + I_3) \cdot (T_1 X_k + T_2 Y_k) + I_4 \cdot T_1 X_k + I_4 \cdot T_2 Y_k \end{aligned}$$

The control signals for the temporary register, T, are

$$\begin{aligned} T_{in} &= T_1 I_4 \\ T_{out} &= T_3 I_4 \end{aligned}$$

7.34. (*a*) Period $= 2 \times n \times t_p$

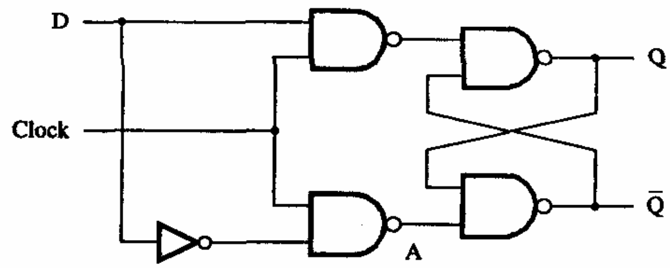
(*b*)

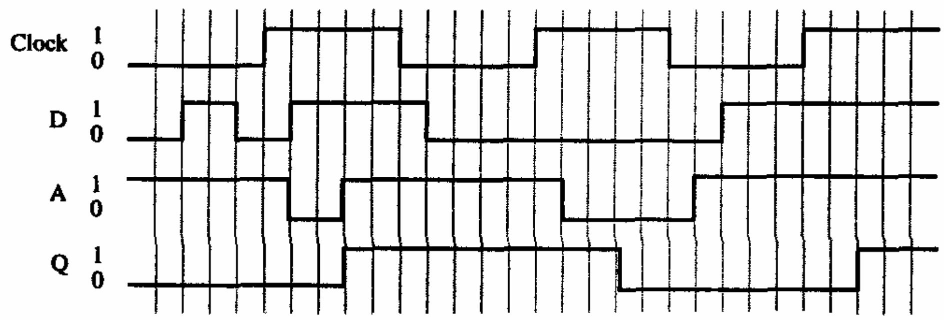


The counter tallies the number of pulses in the 100 ns time period. Thus

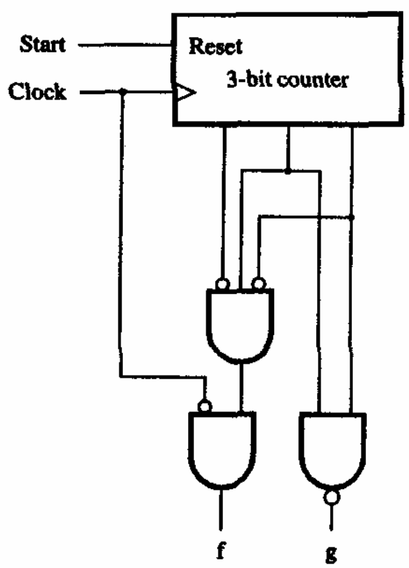
$$t_p = \frac{100 \text{ ns}}{2 \times Count \times n}$$

7.35.





7.36.



If you have any question, call or talk to your instructor.

Here is End Of Chapter 8
Solutions.

Chapter 8

8.1. The expressions for the inputs of the flip-flops are

$$D_2 = Y_2 = \overline{w}y_2 + \overline{y}_1\overline{y}_2$$
$$D_1 = Y_1 = w \oplus y_1 \oplus y_2$$

The output equation is

$$z = y_1 y_2$$

8.2. The excitation table for JK flip-flops is

Present state y_2y_1	Flip-flop inputs				Output z
	$w=0$		$w=1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	1d	0d	1d	1d	0
01	0d	d0	0d	d1	0
10	d0	1d	d1	0d	0
11	d0	d1	d1	d0	1

The expressions for the inputs of the flip-flops are

$$J_2 = \overline{y}_1$$
$$K_2 = w$$
$$J_1 = \overline{w}y_2 + w\overline{y}_2$$
$$K_1 = J_1$$

The output equation is

$$z = y_1 y_2$$

8.3. A possible state table is

Present state	Next state		Output z	
	$w=0$	$w=1$	$w=0$	$w=1$
A	A	B	0	0
B	E	C	0	0
C	E	D	0	0
D	E	D	0	1
E	F	B	0	0
F	A	B	0	1

8.4.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_4 IS
    PORT ( Clock  : IN   STD_LOGIC ;
           Resetn : IN   STD_LOGIC ;
           w      : IN   STD_LOGIC ;
           z      : OUT  STD_LOGIC ) ;
END prob8_4 ;

ARCHITECTURE Behavior OF prob8_4 IS
    TYPE State_type IS ( A, B, C, D, E, F ) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            y <= A ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            CASE y IS
                WHEN A =>
                    IF w = '0' THEN y <= A ;
                    ELSE y <= B ;
                    END IF ;
                WHEN B =>
                    IF w = '0' THEN y <= E ;
                    ELSE y <= C ;
                    END IF ;
                WHEN C =>
                    IF w = '0' THEN y <= E ;
                    ELSE y <= D ;
                    END IF ;
                WHEN D =>
                    IF w = '0' THEN y <= E ;
                    ELSE y <= D ;
                    END IF ;
                WHEN E =>
                    IF w = '0' THEN y <= F ;
                    ELSE y <= B ;
                    END IF ;
                WHEN F =>
                    IF w = '0' THEN y <= A ;
                    ELSE y <= B ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;

    . . . con't
```

```
    PROCESS ( y, w )
    BEGIN
       IF (y = D AND w = '1') OR (y = F AND w = '1') THEN
          z <= '1' ;
       ELSE
          z <= '0' ;
       END IF ;
    END PROCESS ;
END Behavior ;
```

8.5. A minimal state table is

Present state	Next State $w=0$	Next State $w=1$	Output z
A	A	B	0
B	E	C	0
C	D	C	0
D	A	F	1
E	A	F	0
F	E	C	1

8.6. An initial attempt at deriving a state table may be

Present state	Next state $w=0$	Next state $w=1$	Output z $w=0$	Output z $w=1$
A	A	B	0	0
B	D	C	0	0
C	D	C	1	0
D	A	E	0	1
E	D	C	0	0

States B and E are equivalent; hence the minimal state table is

Present state	Next state $w=0$	Next state $w=1$	Output z $w=0$	Output z $w=1$
A	A	B	0	0
B	D	C	0	0
C	D	C	1	0
D	A	B	0	1

8.7. For Figure 8.51 have (using the straightforward state assignment):

	Present state $y_3y_2y_1$	Next state $w=0$ $Y_3Y_2Y_1$	Next state $w=1$ $Y_3Y_2Y_1$	Output z
A	000	001	010	1
B	001	011	101	1
C	010	101	100	0
D	011	001	110	1
E	100	101	010	0
F	101	100	011	0
G	110	101	110	0

This leads to

$$\begin{aligned} Y_3 &= \overline{w}y_3 + \overline{y}_1 y_2 + w y_1 \overline{y}_3 \\ Y_2 &= w y_3 + w\overline{y}_1\overline{y}_2 + w y_1 y_2 + \overline{w} y_1 \overline{y}_2 \overline{y}_3 \\ Y_1 &= \overline{y}_3\overline{w} + \overline{y}_1\overline{w} + w y_1 \overline{y}_2 \\ z &= y_1\overline{y}_3 + \overline{y}_2\overline{y}_3 \end{aligned}$$

For Figure 8.52 have

	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
A	00	01	10	1
B	01	00	11	1
C	10	11	10	0
F	11	10	00	0

This leads to

$$\begin{aligned} Y_2 &= \overline{w}y_2 + \overline{y}_1 y_2 + w\overline{y}_2 \\ Y_1 &= \overline{y}_1\overline{w} + w y_1 \overline{y}_2 \\ z &= \overline{y}_2 \end{aligned}$$

Clearly, minimizing the number of states leads to a much simpler circuit.

8.8. For Figure 8.55 have (using straightforward state assignment):

	Present state $y_4y_3y_2y_1$	Next state $Y_4Y_3Y_2Y_1$				Output z
		DN=00	01	10	11	
S1	0000	0000	0010	0001	–	0
S2	0001	0001	0011	0100	–	0
S3	0010	0010	0101	0110	–	0
S4	0011	0000	–	–	–	1
S5	0100	0010	–	–	–	1
S6	0101	0101	0111	1000	–	0
S7	0110	0000	–	–	–	1
S8	0111	0000	–	–	–	1
S9	1000	0010	–	–	–	1

The next-state and output expressions are

$$
\begin{aligned}
Y_4 &= Dy_3 \\
Y_3 &= Dy_1 + Dy_2 + Ny_2 + \overline{D}y_3\overline{y}_2y_1 \\
Y_2 &= N\overline{y}_2 + y_3\overline{y}_1 + \overline{N}\overline{y}_3y_2\overline{y}_1 \\
Y_1 &= Ny_2 + D\overline{y}_2\overline{y}_1 + \overline{D}\overline{y}_2y_1 \\
z &= y_4 + y_1y_2 + \overline{y}_1y_3
\end{aligned}
$$

Using the same approach for Figure 8.56 gives

	Present state $y_3y_2y_1$	Next state $Y_3Y_2Y_1$				Output z
		DN=00	01	10	11	
S1	000	000	010	001	–	0
S2	001	001	011	100	–	0
S3	010	010	001	011	–	0
S4	011	000	–	–	–	1
S5	100	010	–	–	–	1

The next-state and output expressions are:

$$
\begin{aligned}
Y_3 &= D\overline{y}_2y_1 \\
Y_2 &= y_3 + \overline{N}y_2\overline{y}_1 + N\overline{y}_2 \\
Y_1 &= \overline{D}\overline{y}_2y_1 + Ny_2\overline{y}_1 + D\overline{y}_3\overline{y}_1 \\
z &= y_3 + y_2y_1
\end{aligned}
$$

These expressions define a circuit that has considerably lower cost that the circuit resulting from Figure 8.55.

8.9. To compare individual bits, let $k = w_1 \oplus w_2$. Then, a suitable state table is

Present state	Next state		Output z	
	$k=0$	$k=1$	$k=0$	$k=1$
A	B	A	0	0
B	C	A	0	0
C	D	A	0	0
D	D	A	1	0

The state-assigned table is

Present state	Next State		Output	
	$k=0$	$k=1$	$k=0$	$k=1$
y_2y_1	Y_2Y_1	Y_2Y_1	z	z
00	01	00	0	0
01	10	00	0	0
10	11	00	0	0
11	11	00	1	0

The next-state and output expressions are

$$
\begin{aligned}
Y_2 &= \overline{k}y_1 + \overline{k}y_2 \\
Y_1 &= \overline{k}\overline{y}_1 + \overline{k}y_2 \\
z &= \overline{k}y_1y_2
\end{aligned}
$$

8.10.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_10 IS
    PORT ( Clock   : IN    STD_LOGIC ;
           Resetn  : IN    STD_LOGIC ;
           w1, w2  : IN    STD_LOGIC ;
           z       : OUT   STD_LOGIC ) ;
END prob8_10 ;

ARCHITECTURE Behavior OF prob8_10 IS
    TYPE State_type IS ( A, B, C, D ) ;
    SIGNAL y : State_type ;
    SIGNAL k : STD_LOGIC ;

    . . . con't
```

```
BEGIN
    k <= w1 XOR w2 ;
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            y <= A ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            CASE y IS
                WHEN A =>
                    IF k = '0' THEN y <= B ;
                    ELSE y <= A ;
                    END IF ;
                WHEN B =>
                    IF k = '0' THEN y <= C ;
                    ELSE y <= A ;
                    END IF ;
                WHEN C =>
                    IF k = '0' THEN y <= D ;
                    ELSE y <= A ;
                    END IF ;
                WHEN D =>
                    IF k = '0' THEN y <= D ;
                    ELSE y <= A ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;

    z <= '1' WHEN y = D AND k = '0' ELSE '0' ;
END Behavior ;
```

8.11. A possible minimum state table for a Moore-type FSM is

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	B	C	0
B	D	E	0
C	E	D	0
D	F	G	0
E	F	F	0
F	A	A	0
G	A	A	1

8.12. A minimum state table is shown below. We assume that the 3-bit patterns do not overlap.

Present state	Next state $w=0$	Next state $w=1$	Output p
A	B	C	0
B	D	E	0
C	E	D	0
D	A	F	0
E	F	A	0
F	B	C	1

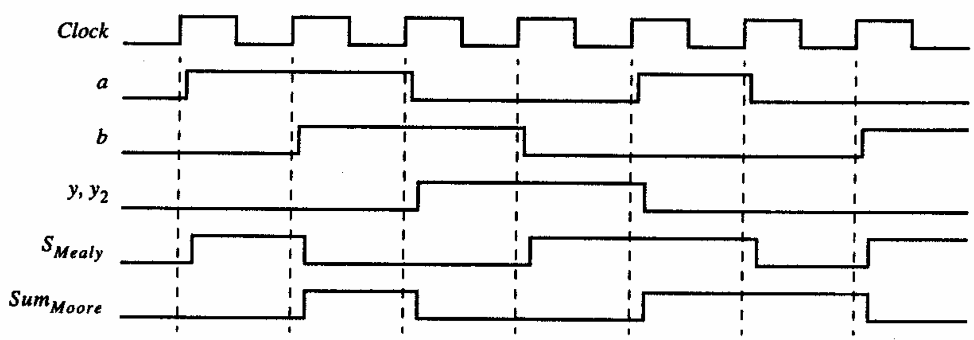
8.13.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_13 IS
    PORT ( Clock   : IN    STD_LOGIC ;
           Resetn  : IN    STD_LOGIC ;
           w       : IN    STD_LOGIC ;
           p       : OUT   STD_LOGIC ) ;
END prob8_13 ;

ARCHITECTURE Behavior OF prob8_13 IS
    TYPE State_type IS ( A, B, C, D, E, F ) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
       IF Resetn = '0' THEN
          y <= A ;
       ELSIF (Clock'EVENT AND Clock = '1') THEN
          CASE y IS
             WHEN A =>
                IF w = '0' THEN y <= B ;
                ELSE y <= C ;
                END IF ;
             WHEN B =>
                IF w = '0' THEN y <= D ;
                ELSE y <= E ;
                END IF ;
             WHEN C =>
                IF w = '0' THEN y <= E ;
                ELSE y <= D ;
                END IF ;
```

. . . con't

```
            WHEN D =>
                IF w = '0' THEN y <= A ;
                ELSE y <= F ;
                END IF ;
            WHEN E =>
                IF w = '0' THEN y <= F ;
                ELSE y <= A ;
                END IF ;
            WHEN F =>
                IF w = '0' THEN y <= B ;
                ELSE y <= C ;
                END IF ;
        END CASE ;
    END IF ;
END PROCESS ;

p <= '1' WHEN y = F ELSE '0' ;
END Behavior ;
```

8.14. The timing diagram is



8.15. The state table corresponding to Figure P8.1 is

Present state	Next state $w=0$	Next state $w=1$	Output z
A	C	D	0
B	B	A	0
C	D	A	0
D	C	B	1

Using one-hot encoding, the state-assigned table is

	Present state	Next state		Output
		$w=0$	$w=1$	
	$y_4y_3y_2y_1$	$Y_4Y_3Y_2Y_1$	$Y_4Y_3Y_2Y_1$	z
A	0001	0100	1000	0
B	0010	0010	0001	0
C	0100	1000	0001	0
D	1000	0100	0010	1

The next-state expressions are

$$\begin{aligned} D_4 &= Y_4 = \overline{w}y_3 + wy_1 \\ D_3 &= Y_3 = \overline{w}(y_1 + y_4) \\ D_2 &= Y_2 = \overline{w}y_2 + wy_4 \\ D_1 &= Y_1 = w(y_2 + y_1) \end{aligned}$$

The output is given by $z = y_4$.

8.16. The state-assignment given in problem 8.15 can be used, except that the state variable y_1 should be complemented. Thus, the state assignment will be $y_4y_3y_2y_1 = 0000, 0011, 0101$, and 1001, for the states A, B, C, and D, respectively. The circuit derived in problem 8.15 can be used, except that the signal for the state variable y_1 should be taken from the $\overline{\text{Q}}$ output of flip-flop 1, rather than from its Q output.

.17. The partitioning process gives

$$\begin{aligned} P_1 &= (ABCDEFG) \\ P_2 &= (ABD)(CEFG) \\ P_3 &= (ABD)(CEG)(F) \\ P_4 &= (ABD)(CEG)(F) \end{aligned}$$

The minimum state table is

Present state	Next state		Output z	
	$w=0$	$w=1$	$w=0$	$w=1$
A	A	C	0	0
C	F	C	0	1
F	C	A	0	1

8.18. The partitioning process gives

$$\begin{aligned} P_1 &= (ABCDEFG) \\ P_2 &= (ADG)(BCEF) \\ P_3 &= (AG)(D)(B)(CE)(F) \\ P_4 &= (A)(G)(D)(B)(CE)(F) \end{aligned}$$

The minimized state table is

Present state	Next state $w=0$	Next state $w=1$	Output z $w=0$	Output z $w=1$
A	B	C	0	0
B	D	–	0	1
C	F	C	0	1
D	B	G	0	0
F	C	D	0	1
G	F	–	0	0

8.19. An implementation for the Moore-type FSM in Figures 8.5.7 and 8.5.6 is given in the solution for problem 8.8. The Mealy-type FSM in Figure 8.58 is described in the form of a state table as

Present state	Next state				Output z			
	DN=00	01	10	11	00	01	10	11
S1	S1	S3	S2	–	0	0	0	1
S2	S2	S1	S3	–	0	1	1	–
S3	S3	S2	S1	–	0	0	1	–

The state-assigned table is

Present state	Next state				Output			
	DN=00	01	10	11	00	01	10	11
y_2y_1	Y_2Y_1	Y_2Y_1	Y_2Y_1	Y_2Y_1	z	z	z	z
00	00	10	01	–	0	0	0	–
01	01	00	10	–	0	1	1	–
10	10	01	00	–	0	0	1	–

The next-state and output expressions are

$$
\begin{aligned}
Y_2 &= Dy_1 + \overline{D}y_2\overline{N} + N\overline{y}_2\overline{y}_1 \\
Y_1 &= Ny_2 + \overline{D}y_1\overline{N} + D\overline{y}_2\overline{y}_1 \\
z &= Dy_1 + Dy_2 + Ny_1
\end{aligned}
$$

In this case, choosing the Mealy model results in a simpler circuit.

.20. Use w as the clock. Then the state table is

Present state	Next state	Output $z_1 z_0$
A	B	00
B	C	10
C	D	01
D	A	11

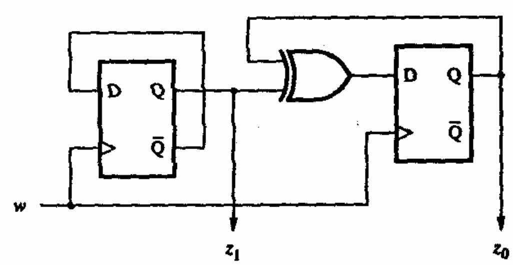
The state-assigned table is

Present state $y_1 y_0$	Next state $Y_1 Y_0$	Output $z_1 z_0$
00	10	00
10	01	10
01	11	01
11	00	11

The next-state expressions are

$$\begin{aligned} Y_1 &= \overline{y}_1 \\ Y_2 &= y_1 \oplus y_2 \end{aligned}$$

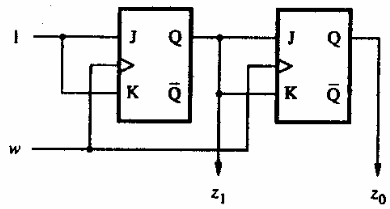
The resulting circuit is



8.21. From the state-assigned table given in the solution to Problem 8.20, the excitation table for JK flip-flops is

Present state $y_1 y_0$	Flip-flop inputs $J_1 K_1$	$J_0 K_0$	Output $z_1 z_0$
00	$1d$	$0d$	00
10	$d1$	$1d$	10
01	$1d$	$d0$	01
11	$d1$	$d1$	11

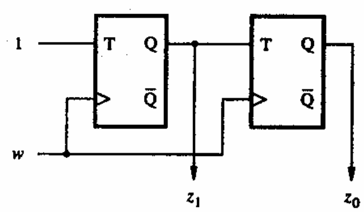
The flip-flop inputs are $J_1 = K_1 = 1$ and $J_2 = K_2 = y_1$. The resulting circuit is



8.22. From the state-assigned table given in the solution to Problem 8.20, the excitation table for T flip-flops is

Present state $y_1 y_0$	Flip-flop inputs T_1	T_0	Output $z_1 z_0$
00	1	0	00
10	1	1	10
01	1	0	01
11	1	1	11

The flip-flop inputs are $T_1 = 1$ and $T_2 = y_1$. The resulting circuit is



8.23. The state diagram is

Present state	Next state $w = 0$	$w = 1$	Output $z_2 z_1 z_0$
A	A	B	000
B	B	C	001
C	C	D	010
D	D	E	011
E	E	F	100
F	F	A	101

The state-assigned table is

Present state $y_2y_1y_0$	Next state $w=0$	$w=1$	Output $z_2z_1z_0$
	$Y_2Y_1Y_0$		
000	000	001	000
001	001	010	001
010	010	011	010
011	011	100	011
100	100	101	100
101	101	000	101

The next-state expressions are

$$\begin{aligned} Y_2 &= \bar{y}_0 y_2 + \bar{w} y_2 + w y_0 y_1 \\ Y_1 &= \bar{y}_0 y_1 + \bar{w} y_1 + w y_0 \bar{y}_1 \bar{y}_2 \\ Y_0 &= \bar{w} y_0 + w \bar{y}_0 \end{aligned}$$

The outputs are: $z_2 = y_2$, $z_1 = y_1$, and $z_0 = y_0$.

8.24. Using the state-assigned table given in the solution for problem 8.23, the excitation table for JK flip-flops is

Present state	Flip-flop inputs						Outputs
	$w=0$			$w=1$			
$y_2y_1y_0$	J_2K_2	J_1K_1	J_0K_0	J_2K_2	J_1K_1	J_0K_0	$z_2z_1z_0$
000	0 d	0 d	0 d	0 d	0 d	1 d	000
001	0 d	0 d	d 0	0 d	1 d	d 1	001
010	0 d	d 0	0 d	0 d	d 0	1 d	010
011	0 d	d 0	d 0	1 d	d 1	d 1	011
100	d 0	0 d	0 d	d 0	0 d	1 d	100
101	d 0	0 d	d 0	d 1	0 d	d 1	101

The expressions for the inputs of the flip-flops are

$$\begin{aligned} J_2 &= w y_1 y_0 \\ K_2 &= w y_2 y_0 \\ J_1 &= w \bar{y}_2 y_0 \\ K_1 &= w y_0 \\ J_0 &= w \\ K_0 &= w \end{aligned}$$

The outputs are: $z_2 = y_2$, $z_1 = y_1$, and $z_0 = y_0$.

8.25. Using the state-assigned table given in the solution for problem 8.23, the excitation table for T flip-flops is

Present state $y_2y_1y_0$	Flip-flop inputs $w=0$ $T_2T_1T_0$	$w=1$ $T_2T_1T_0$	Outputs $z_2z_1z_0$
000	000	001	000
001	000	011	001
010	000	001	010
011	000	111	011
100	000	001	100
101	000	101	101

The expressions for T inputs of the flip-flops are

$$\begin{aligned} T_2 &= wy_1y_0 + wy_2y_0 \\ T_1 &= w\overline{y}_2y_0 \\ T_0 &= w \end{aligned}$$

The outputs are: $z_2 = y_2$, $z_1 = y_1$, and $z_0 = y_0$.

8.26. The state diagram is

Present state	Next state $w=0$	$w=1$	Count
A	H	C	0
B	A	D	1
C	B	E	2
D	C	F	3
E	D	G	4
F	E	H	5
G	F	A	6
H	G	B	7

The state-assigned table is

	Present state $y_2y_1y_0$	Next state $w=0$ $Y_2Y_1Y_0$	Next state $w=1$ $Y_2Y_1Y_0$	Output $z_2z_1z_0$
A	000	111	010	000
B	001	000	011	001
C	010	001	100	010
D	011	010	101	011
E	100	011	110	100
F	101	100	111	101
G	110	101	000	110
H	111	110	001	111

The next-state expressions (inputs to D flip-flops) are

$$
\begin{aligned}
D_2 &= Y_2 = w\overline{y}_2y_1 + \overline{w}y_2y_1 + wy_2\overline{y}_1 + \overline{w}y_2y_0 + \overline{y}_2\overline{y}_1\overline{y}_0w \\
D_1 &= Y_1 = w\overline{y}_1 + \overline{y}_1\overline{y}_0 + \overline{w}y_1y_0 \\
D_0 &= Y_0 = \overline{y}_0\overline{w} + y_0w
\end{aligned}
$$

The outputs are: $z_2 = y_2$, $z_1 = y_1$, and $z_0 = y_0$.

8.27. From the state-assigned table given in the solution to problem 8.26, the excitation table for JK flip-flops is

Present state	Flip-flop inputs						Outputs
	$w=0$			$w=1$			
$y_2y_1y_0$	J_2K_2	J_1K_1	J_0K_0	J_2K_2	J_1K_1	J_0K_0	$z_2z_1z_0$
000	$1\,d$	$1\,d$	$1\,d$	$0\,d$	$1\,d$	$0\,d$	000
001	$0\,d$	$0\,d$	$d\,1$	$0\,d$	$1\,d$	$d\,0$	001
010	$0\,d$	$d\,1$	$1\,d$	$1\,d$	$d\,1$	$0\,d$	010
011	$0\,d$	$d\,0$	$d\,1$	$1\,d$	$d\,1$	$d\,0$	011
100	$d\,1$	$1\,d$	$1\,d$	$d\,0$	$1\,d$	$0\,d$	100
101	$d\,0$	$0\,d$	$d\,1$	$d\,0$	$1\,d$	$d\,0$	101
110	$d\,0$	$d\,1$	$1\,d$	$d\,1$	$d\,1$	$0\,d$	110
111	$d\,0$	$d\,0$	$d\,1$	$d\,1$	$d\,1$	$d\,0$	111

The expressions for J and K inputs to the three flip-flops are

$$
\begin{aligned}
J_2 &= y_1w + \overline{y}_1\overline{y}_0\overline{w} \\
K_2 &= J_2 \\
J_1 &= w + \overline{y}_0 \\
K_1 &= J_1 \\
J_0 &= \overline{w} \\
K_0 &= J_0
\end{aligned}
$$

The outputs are: $z_2 = y_2$, $z_1 = y_1$, and $z_0 = y_0$.

8.28. From the state-assigned table given in the solution to problem 8.26, the excitation table for T flip-flops is

Present state	Flip-flop inputs		Outputs
	$w = 0$	$w = 1$	
$y_2y_1y_0$	$T_2T_1T_0$	$T_2T_1T_0$	$z_2z_1z_0$
000	111	010	000
001	001	010	001
010	011	110	010
011	001	110	011
100	111	010	100
101	001	010	101
110	011	110	110
111	001	110	111

The expressions for T inputs of the flip-flops are

$$\begin{aligned} T_2 &= \overline{y}_1\overline{y}_0\overline{w} + y_1 w \\ T_1 &= w + \overline{y}_0 \\ T_0 &= \overline{w} \end{aligned}$$

The outputs are: $z_2 = y_2$, $z_1 = y_1$, and $z_0 = y_0$.

8.29. The next-state and output expressions are

$$\begin{aligned} D_1 = Y_1 &= w(y_1 + y_2) \\ D_2 = Y_2 &= w(\overline{y}_1 + \overline{y}_2) \\ z &= y_1\overline{y}_2 \end{aligned}$$

The corresponding state-assigned table is

Present state	Next state		Output
	$w = 0$	$w = 1$	
y_2y_1	Y_2Y_1	Y_2Y_1	z
00	00	10	0
01	00	11	1
10	00	11	0
11	00	01	0

This leads to the state table

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	C	0
B	A	D	1
C	A	D	0
D	A	B	0

The circuit produces $z = 1$ whenever the input sequence on w comprises a 0 followed by an even number of 1s.

8.30.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_30 IS
    PORT ( Clock   : IN    STD_LOGIC ;
           Resetn  : IN    STD_LOGIC ;
           N, D    : IN    STD_LOGIC ;
           z       : OUT   STD_LOGIC ) ;
END prob8_30 ;

ARCHITECTURE Behavior OF prob8_30 IS
    TYPE State_type IS ( S1, S2, S3, S4, S5 ) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            y <= S1 ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            CASE y IS
                WHEN S1 =>
                    IF N = '1' THEN y <= S3 ;
                    ELSIF D = '1' THEN y <= S2 ;
                    ELSE y <= S1 ;
                    END IF ;
                WHEN S2 =>
                    IF N = '1' THEN y <= S4 ;
                    ELSIF D = '1' THEN y <= S5 ;
                    ELSE y <= S2 ;
                    END IF ;
                WHEN S3 =>
                    IF N = '1' THEN y <= S2 ;
                    ELSIF D = '1' THEN y <= S4 ;
                    ELSE y <= S3 ;
                    END IF ;
                WHEN S4 =>
                    y <= S1 ;
                WHEN S5 =>
                    y <= S3 ;
            END CASE ;
        END IF ;
    END PROCESS ;

    z <= '1' WHEN y = S4 OR y = S5 ELSE '0' ;
END Behavior ;
```

8.31.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_32 IS
    PORT ( Resetn, Clock  : IN   STD_LOGIC ;
           N, D           : IN   STD_LOGIC ;
           z              : OUT  STD_LOGIC ) ;
END prob8_32 ;

ARCHITECTURE Behavior OF prob8_32 IS
    TYPE State_type IS ( S1, S2, S3 ) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
       IF Resetn = '0' THEN
          y <= S1 ;
       ELSIF Clock'EVENT AND Clock = '1' THEN
          CASE y IS
             WHEN S1 =>
                IF N = '1' THEN y <= S3 ;
                ELSIF D = '1' THEN y <= S2 ;
                ELSE y <= S1 ; END IF ;
             WHEN S2 =>
                IF N = '1' THEN y <= S1 ;
                ELSIF D = '1' THEN y <= S3 ;
                ELSE y <= S2 ; END IF ;
             WHEN S3 =>
                IF N = '1' THEN y <= S2 ;
                ELSIF D = '1' THEN y <= S1 ;
                ELSE y <= S3 ; END IF ;
          END CASE ;
       END IF ;
    END PROCESS ;

    z <= '1' WHEN (y = S2 AND (D = '1' OR N = '1')) OR (y = S3 AND D = '1') ELSE '0' ;
END Behavior ;
```

8.32.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_32 IS
    PORT ( Clock   : IN   STD_LOGIC ;
           Resetn  : IN   STD_LOGIC ;
           N, D    : IN   STD_LOGIC ;
           z       : OUT  STD_LOGIC ) ;
END prob8_32 ;
```

. . . con't

```
ARCHITECTURE Behavior OF prob8_32 IS
    TYPE State_type IS ( S1, S2, S3 ) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            y <= S1 ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            CASE y IS
                WHEN S1 =>
                    IF N = '1' THEN y <= S3 ;
                    ELSIF D = '1' THEN y <= S2 ;
                    ELSE y <= S1 ;
                    END IF ;
                WHEN S2 =>
                    IF N = '1' THEN y <= S1 ;
                    ELSIF D = '1' THEN y <= S3 ;
                    ELSE y <= S2 ;
                    END IF ;
                WHEN S3 =>
                    IF N = '1' THEN y <= S2 ;
                    ELSIF D = '1' THEN y <= S1 ;
                    ELSE y <= S3 ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;

    z <= '1' WHEN (y = S2 AND (D = '1' OR N = '1')) OR (y = S3 AND D = '1') ELSE '0' ;
END Behavior ;
```

8.33.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_33 IS
    PORT ( Clock    : IN    STD_LOGIC ;
           Resetn   : IN    STD_LOGIC ;
           N, D     : IN    STD_LOGIC ;
           z        : OUT   STD_LOGIC ) ;
END prob8_33 ;

ARCHITECTURE Behavior OF prob8_33 IS
    TYPE State_type IS ( S1, S2, S3 ) ;
    SIGNAL y_present, y_next : State_type ;

    . . . con't
```

```
BEGIN
    PROCESS ( N, D, y_present)
    BEGIN
        CASE y_present IS
            WHEN S1 =>
                IF N = '1' THEN y_next <= S3 ;
                ELSIF D = '1' THEN y_next <= S2 ;
                ELSE y_next <= S1 ;
                END IF ;
            WHEN S2 =>
                IF N = '1' THEN y_next <= S1 ;
                ELSIF D = '1' THEN y_next <= S3 ;
                ELSE y_next <= S2 ;
                END IF ;
            WHEN S3 =>
                IF N = '1' THEN y_next <= S2 ;
                ELSIF D = '1' THEN y_next <= S1 ;
                ELSE y_next <= S3 ;
                END IF ;
        END CASE ;
    END PROCESS ;

    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            y_present <= S1 ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            y_present <= y_next ;
        END IF ;
    END PROCESS ;

    z <= '1' WHEN (y_present = S2 AND (D = '1' OR N = '1')) OR
        (y_present = S3 AND D = '1') ELSE '0' ;
END Behavior ;
```

8.34.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_34 IS
    PORT ( Clock  : IN   STD_LOGIC ;
           Resetn : IN   STD_LOGIC ;
           w      : IN   STD_LOGIC ;
           z      : OUT  STD_LOGIC ) ;
END prob8_34 ;
```

. . . con't

```
ARCHITECTURE Behavior OF prob8_34 IS
    TYPE State_type IS ( A, B, C, D ) ;
    ATTRIBUTE ENUM_ENCODING : STRING ;
    ATTRIBUTE ENUM_ENCODING OF State_type : TYPE IS "00 01 10 11" ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
       IF Resetn = '0' THEN
          y <= A ;
       ELSIF Clock'EVENT AND Clock = '1' THEN
          CASE y IS
             WHEN A =>
                IF w = '0' THEN y <= C ;
                ELSE y <= D ;
                END IF ;
             WHEN B =>
                IF w = '0' THEN y <= B ;
                ELSE y <= A ;
                END IF ;
             WHEN C =>
                IF w = '0' THEN y <= D ;
                ELSE y <= A ;
                END IF ;
             WHEN D =>
                IF w = '0' THEN y <= C ;
                ELSE y <= B ;
                END IF ;
          END CASE ;
       END IF ;
    END PROCESS ;

    z <= '1' WHEN y = D ELSE '0' ;
END Behavior ;
```

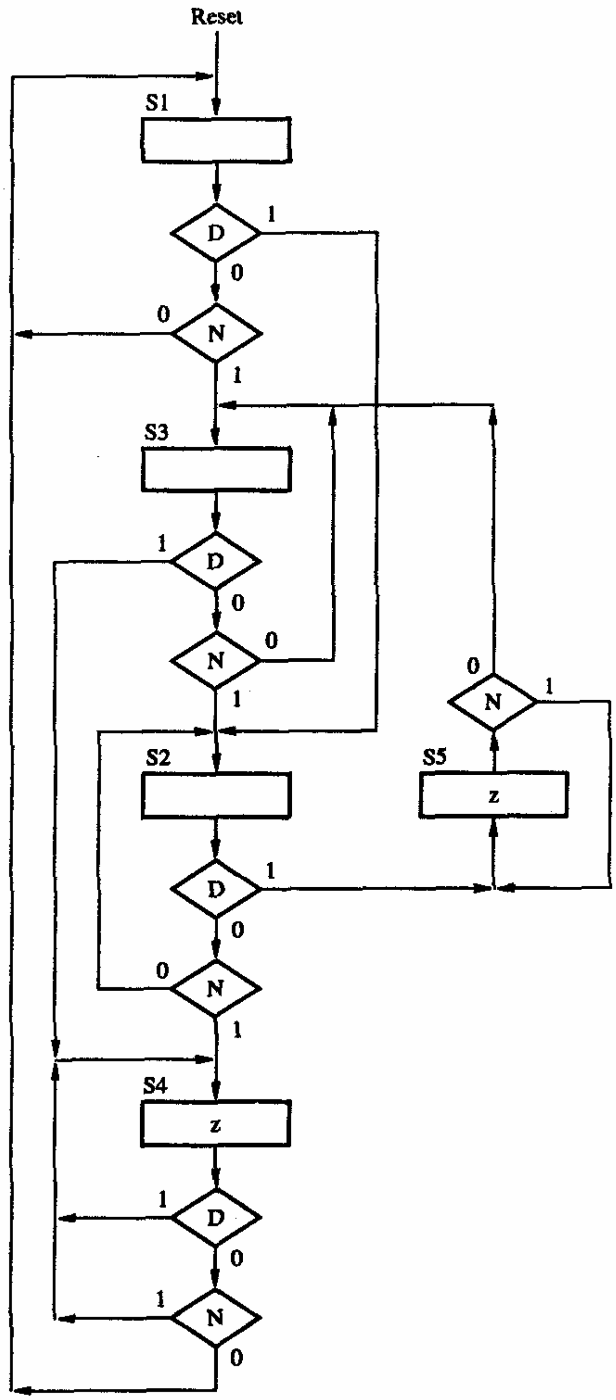
8.35.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_35 IS
    PORT ( Clock    : IN    STD_LOGIC ;
           Resetn   : IN    STD_LOGIC ;
           w        : IN    STD_LOGIC ;
           z        : OUT   STD_LOGIC ) ;
END prob8_35 ;
```

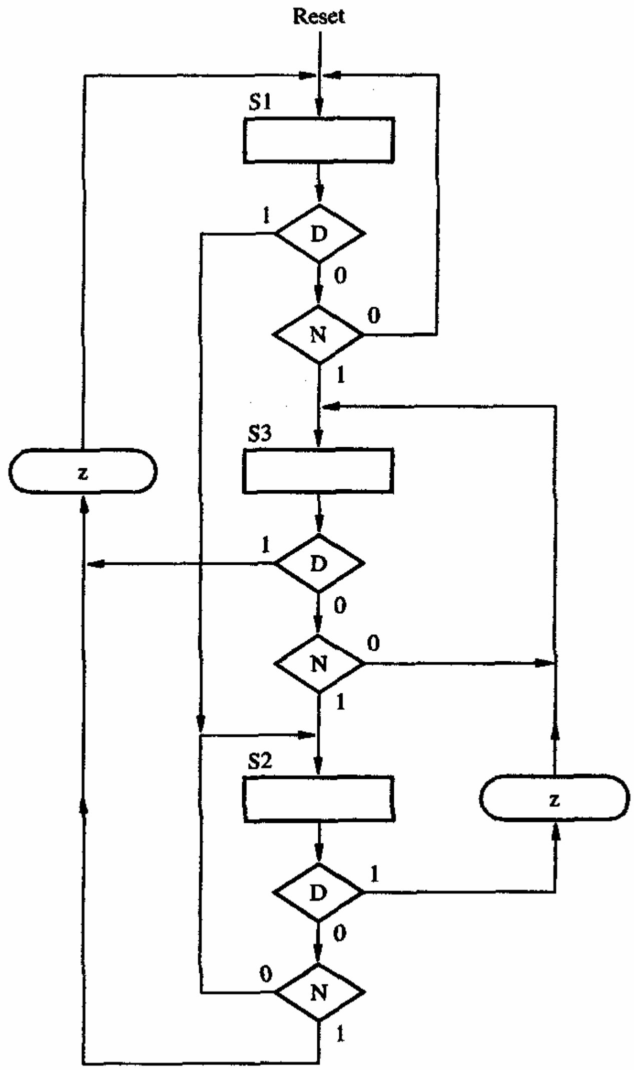
. . . con't

```
ARCHITECTURE Behavior OF prob8_35 IS
    SIGNAL y_present, y_next : STD_LOGIC_VECTOR(1 DOWNTO 0) ;
    CONSTANT A : STD_LOGIC_VECTOR(1 DOWNTO 0) := "00" ;
    CONSTANT B : STD_LOGIC_VECTOR(1 DOWNTO 0) := "01" ;
    CONSTANT C : STD_LOGIC_VECTOR(1 DOWNTO 0) := "10" ;
    CONSTANT D : STD_LOGIC_VECTOR(1 DOWNTO 0) := "11" ;
BEGIN
    PROCESS ( w, y_present )
    BEGIN
        CASE y_present IS
            WHEN A =>
                IF w = '0' THEN y_next <= C ;
                ELSE y_next <= D ;
                END IF ;
            WHEN B =>
                IF w = '0' THEN y_next <= B ;
                ELSE y_next <= A ;
                END IF ;
            WHEN C =>
                IF w = '0' THEN y_next <= D ;
                ELSE y_next <= A ;
                END IF ;
            WHEN OTHERS =>
                IF w = '0' THEN y_next <= C ;
                ELSE y_next <= B ;
                END IF ;
        END CASE ;
    END PROCESS ;

    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            y_present <= A ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            y_present <= y_next ;
        END IF ;
    END PROCESS ;

    z <= '1' WHEN y_present = D ELSE '0' ;
END Behavior ;
```

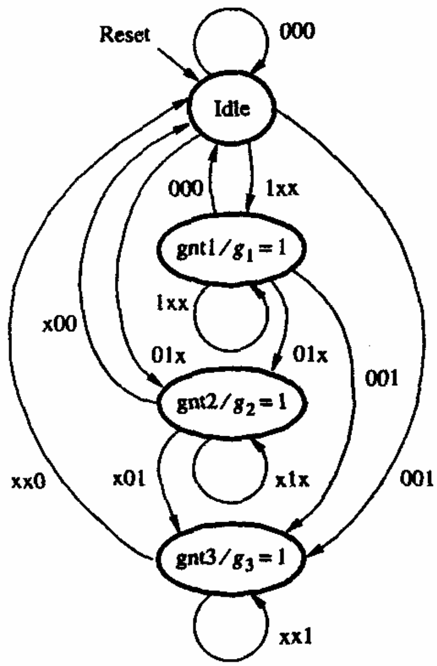
8.36. An ASM chart for the FSM in Figure 8.57 is



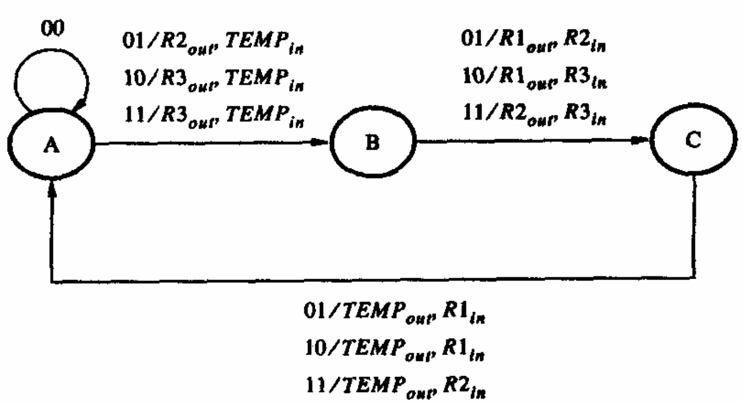
8.37. An ASM chart for the FSM in Figure 8.58 is



8.38. To ensure that the device 3 will get serviced the FSM in Figure 8.72 can be modified as follows:



8.40. The required control signals can be generated using the following FSM:



Let $k = w_2 + w_1$. Then the next-state transitions can be defined as

Present state	Next state	
	$k=0$	$k=1$
A	A	B
B	B	C
C	C	A

Using one-hot encoding, the state-assigned table becomes

Present state	Next state	
	$k=0$	$k=1$
$y_3y_2y_1$	$Y_3Y_2Y_1$	$Y_3Y_2Y_1$
001	001	010
010	010	100
100	100	001

The next-state expressions are

$$
\begin{aligned}
Y_3 &= \bar{k}y_3 + ky_2 \\
Y_2 &= \bar{k}y_2 + ky_1 \\
Y_1 &= \bar{k}y_1 + ky_3
\end{aligned}
$$

The output expressions are

$$
\begin{aligned}
TEMP_{in} &= ky_1 \\
TEMP_{out} &= ky_3 \\
R1_{out} &= y_2(w_2 \oplus w_1) \\
R1_{in} &= y_3(w_2 \oplus w_1) \\
R2_{out} &= y_1\overline{w}_2w_1 + y_2w_2w_1 \\
R2_{in} &= y_2\overline{w}_2w_1 + y_3w_2w_1 \\
R3_{out} &= y_1w_2 \\
R3_{in} &= y_2w_2
\end{aligned}
$$

Please, report any mistake to your instructor, He really appreciates any effort.