Chapter 8

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

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

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

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

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. VHDL code for the solution given in problem 8.3 is

```
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		Output z
	$w = 0$	$w = 1$	
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		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
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		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
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	0 0 0	0 0 1	0 1 0	1
B	0 0 1	0 1 1	1 0 1	1
C	0 1 0	1 0 1	1 0 0	0
D	0 1 1	0 0 1	1 1 0	1
E	1 0 0	1 0 1	0 1 0	0
F	1 0 1	1 0 0	0 1 1	0
G	1 1 0	1 0 1	1 1 0	0

This leads to

$$\begin{aligned} Y_3 &= \overline{w}y_3 + \overline{y}_1y_2 + wy_1\overline{y}_3 \\ Y_2 &= wy_3 + w\overline{y}_1\overline{y}_2 + wy_1y_2 + \overline{w}y_1\overline{y}_2\overline{y}_3 \\ Y_1 &= \overline{y}_3\overline{w} + \overline{y}_1\overline{w} + wy_1\overline{y}_2 \\ z &= y_1\overline{y}_3 + \overline{y}_2\overline{y}_3 \end{aligned}$$

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	0 0	0 1	1 0	1
B	0 1	0 0	1 1	1
C	1 0	1 1	1 0	0
F	1 1	1 0	0 0	0

This leads to

$$\begin{aligned} Y_2 &= \overline{w}y_2 + \overline{y}_1y_2 + w\overline{y}_2 \\ Y_1 &= \overline{y}_1\overline{w} + wy_1\overline{y}_2 \\ z &= \overline{y}_2 \end{aligned}$$

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 DN=00	01	10	11	Output z
		$Y_4Y_3Y_2Y_1$				
S1	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 1	–	0
S2	0 0 0 1	0 0 0 1	0 0 1 1	0 1 0 0	–	0
S3	0 0 1 0	0 0 1 0	0 1 0 1	0 1 1 0	–	0
S4	0 0 1 1	0 0 0 0	–	–	–	1
S5	0 1 0 0	0 0 1 0	–	–	–	1
S6	0 1 0 1	0 1 0 1	0 1 1 1	1 0 0 0	–	0
S7	0 1 1 0	0 0 0 0	–	–	–	1
S8	0 1 1 1	0 0 0 0	–	–	–	1
S9	1 0 0 0	0 0 1 0	–	–	–	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 DN=00	01	10	11	Output z
		$Y_3Y_2Y_1$				
S1	0 0 0	0 0 0	0 1 0	0 0 1	–	0
S2	0 0 1	0 0 1	0 1 1	1 0 0	–	0
S3	0 1 0	0 1 0	0 0 1	0 1 1	–	0
S4	0 1 1	0 0 0	–	–	–	1
S5	1 0 0	0 1 0	–	–	–	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 y_2y_1	Next State		Output	
	$k = 0$	$k = 1$	$k = 0$	$k = 1$
	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. VHDL code for the solution given in problem 8.9 is

```
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.

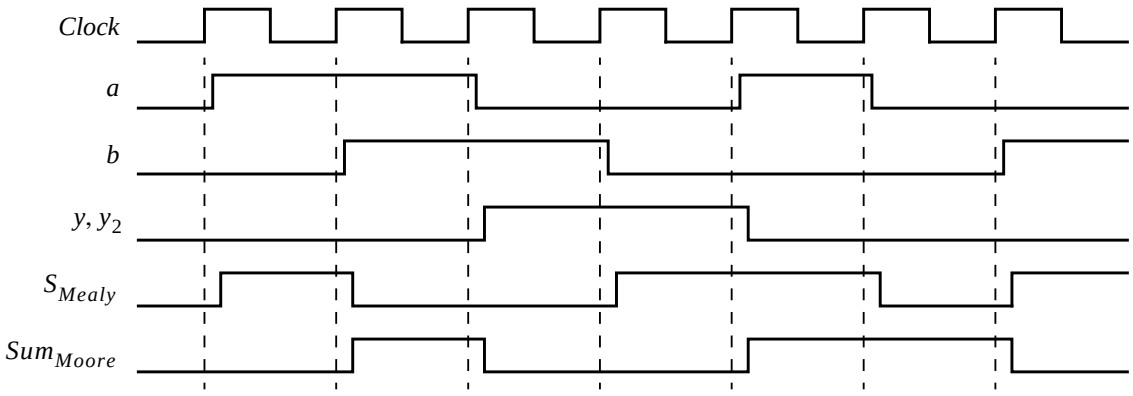
Present state	Next state		Output p
	$w = 0$	$w = 1$	
A	B	C	0
B	D	E	0
C	E	D	0
D	A	F	0
E	F	A	0
F	B	C	1

8.13. VHDL code for the solution given in problem 8.12 is

```
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		Output
	$w = 0$	$w = 1$	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 $y_4y_3y_2y_1$	Next state $w=0$ $Y_4Y_3Y_2Y_1$	Next state $w=1$ $Y_4Y_3Y_2Y_1$	Output z
A	0 0 0 1	0 1 0 0	1 0 0 0	0
B	0 0 1 0	0 0 1 0	0 0 0 1	0
C	0 1 0 0	1 0 0 0	0 0 0 1	0
D	1 0 0 0	0 1 0 0	0 0 1 0	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.

8.17. 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 $w=0$	Next state $w=1$	Output z $w=0$	Output z $w=1$
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		Output z	
	$w=0$	$w=1$	$w=0$	$w=1$
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 y_2y_1	Next state				Output			
	DN=00	01	10	11	00	01	10	11
	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.

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

Present state	Next state	Output $z_1 z_0$
A	B	0 0
B	C	1 0
C	D	0 1
D	A	1 1

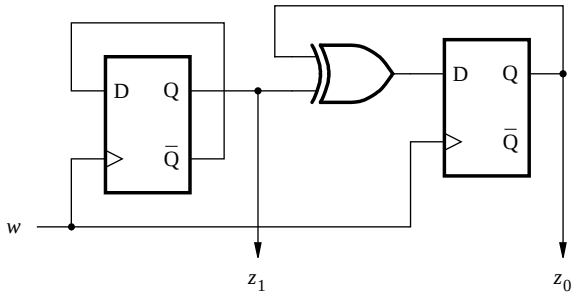
The state-assigned table is

Present state $y_1 y_0$	Next state $Y_1 Y_0$	Output $z_1 z_0$
0 0	1 0	0 0
1 0	0 1	1 0
0 1	1 1	0 1
1 1	0 0	1 1

The next-state expressions are

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

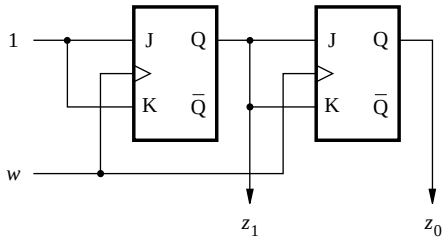
The resulting circuit is



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 $z_1 z_0$
0 0	1 d	0 d	0 0
1 0	d 1	1 d	1 0
0 1	1 d	d 0	0 1
1 1	d 1	d 1	1 1

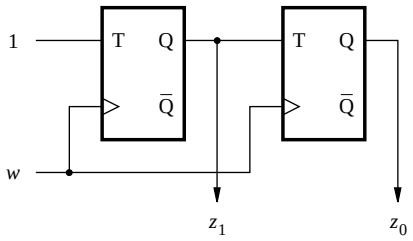
The flip-flop inputs are $J_1 = K_1 = 1$ and $J_2 = K_2 = y_1$. The resulting circuit is



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 $z_1 z_0$
0 0	1	0	0 0
1 0	1	1	1 0
0 1	1	0	0 1
1 1	1	1	1 1

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	0 0 0
B	B	C	0 0 1
C	C	D	0 1 0
D	D	E	0 1 1
E	E	F	1 0 0
F	F	A	1 0 1

The state-assigned table is

Present state $y_2y_1y_0$	Next state $Y_2Y_1Y_0$ $w=0$	Next state $Y_2Y_1Y_0$ $w=1$	Output $z_2z_1z_0$
0 0 0	0 0 0	0 0 1	0 0 0
0 0 1	0 0 1	0 1 0	0 0 1
0 1 0	0 1 0	0 1 1	0 1 0
0 1 1	0 1 1	1 0 0	0 1 1
1 0 0	1 0 0	1 0 1	1 0 0
1 0 1	1 0 1	0 0 0	1 0 1

The next-state expressions are

$$\begin{aligned} Y_2 &= \overline{y}_0y_2 + \overline{w}y_2 + wy_0y_1 \\ Y_1 &= \overline{y}_0y_1 + \overline{w}y_1 + wy_0\overline{y}_1\overline{y}_2 \\ Y_0 &= \overline{w}y_0 + w\overline{y}_0 \end{aligned}$$

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 $y_2y_1y_0$	Flip-flop inputs $w=0$ J_2K_2	J_1K_1	J_0K_0	$w=1$ J_2K_2	J_1K_1	J_0K_0	Outputs $z_2z_1z_0$
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

$$\begin{aligned} J_2 &= wy_1y_0 \\ K_2 &= wy_2y_0 \\ J_1 &= w\overline{y}_2y_0 \\ K_1 &= wy_0 \\ J_0 &= w \\ K_0 &= w \end{aligned}$$

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 $w=0$ $T_2T_1T_0$	Flip-flop inputs $w=1$ $T_2T_1T_0$	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 $w=0$	Next state $w=1$	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	0 0 0	1 1 1	0 1 0	0 0 0
B	0 0 1	0 0 0	0 1 1	0 0 1
C	0 1 0	0 0 1	1 0 0	0 1 0
D	0 1 1	0 1 0	1 0 1	0 1 1
E	1 0 0	0 1 1	1 1 0	1 0 0
F	1 0 1	1 0 0	1 1 1	1 0 1
G	1 1 0	1 0 1	0 0 0	1 1 0
H	1 1 1	1 1 0	0 0 1	1 1 1

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 $y_2y_1y_0$	Flip-flop inputs $w=0$ J_2K_2	J_1K_1	J_0K_0	$w=1$ J_2K_2	J_1K_1	J_0K_0	Outputs $z_2z_1z_0$
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 $y_2y_1y_0$	Flip-flop inputs $w=0$ $T_2T_1T_0$	Flip-flop inputs $w=1$ $T_2T_1T_0$	Outputs $z_2z_1z_0$
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

$$\begin{aligned} T_2 &= \overline{y}_1\overline{y}_0\overline{w} + y_1w \\ T_1 &= w + \overline{y}_0 \\ T_0 &= \overline{w} \end{aligned}$$

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 y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
0 0	0 0	1 0	0
0 1	0 0	1 1	1
1 0	0 0	1 1	0
1 1	0 0	0 1	0

This leads to the state table

Present state	Next state $w=0$	Next state $w=1$	Output 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. The VHDL code based on the style of code in Figure 8.29 is

```
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. The VHDL code based on the style of code in Figure 8.33 is

```
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. The VHDL code based on the style of code in Figure 8.29 is

```
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. The VHDL code based on the style of code in Figure 8.33 is

```
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. VHDL code for the FSM in Figure P8.1 is

```
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
```

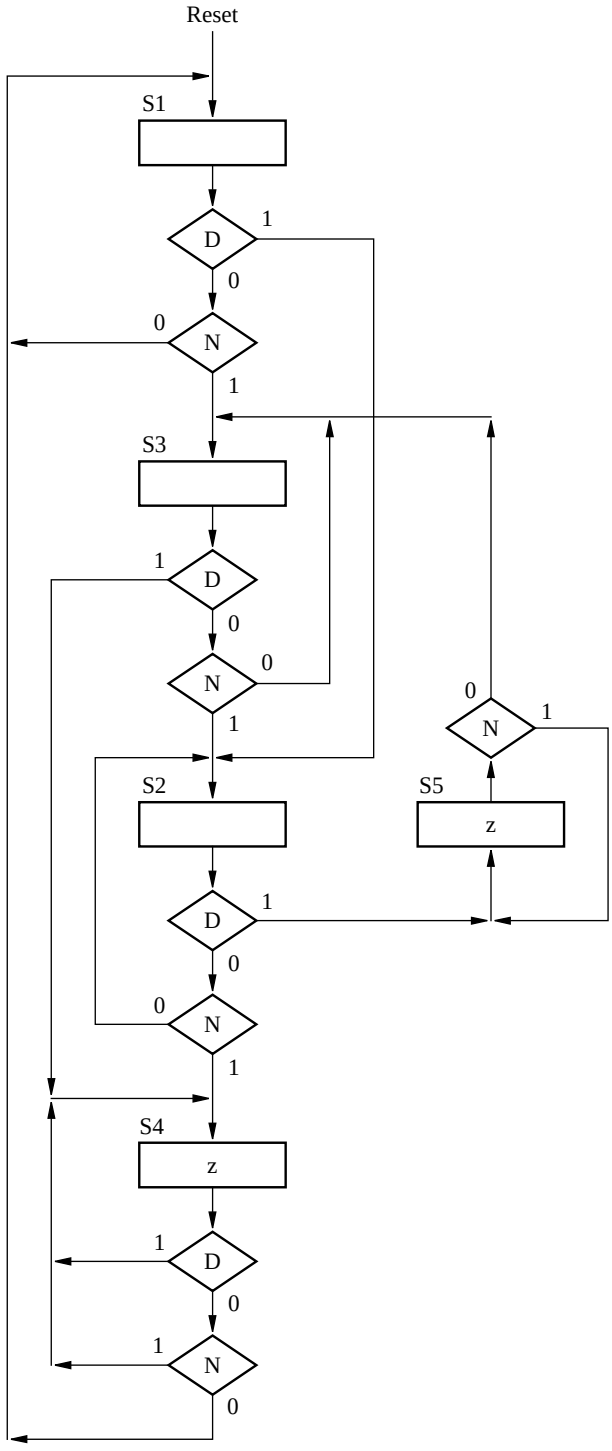
```
ARCHITECTURE Behavior OF prob8_34 IS
    TYPE State_type IS ( A, B, C, D ) ;
    ATTRIBUTE ENUM_ENCODING : STRING ;
    ATTRIBUTE ENUM_ENCODING OF State_type : TYPE IS "00 01 10 11" ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            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. The VHDL code is

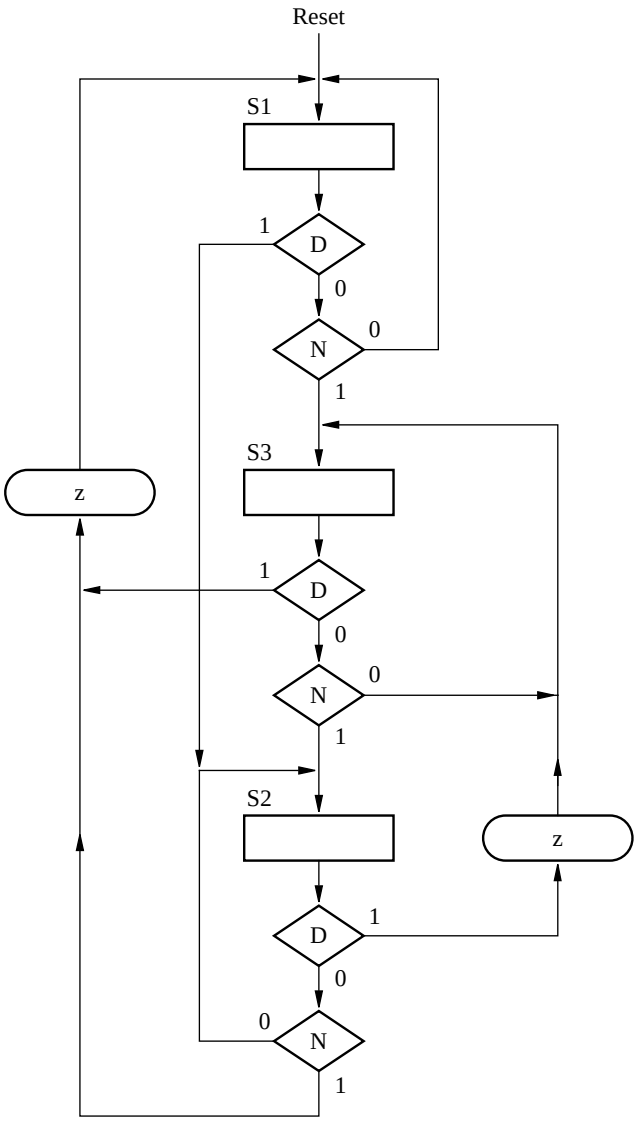
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 ;
```

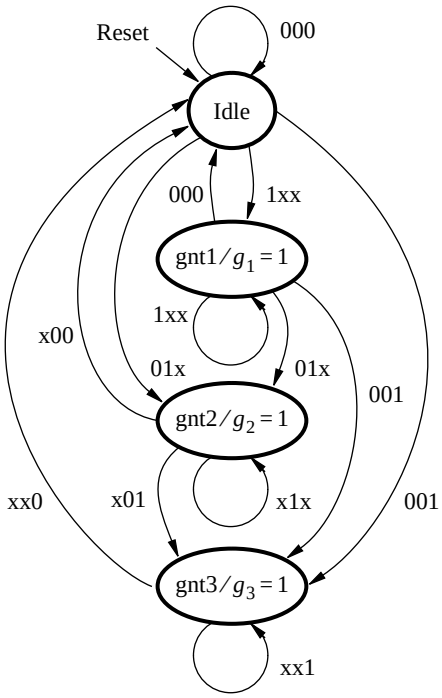
8.36. An ASM chart for the FSM in Figure 8.57 is



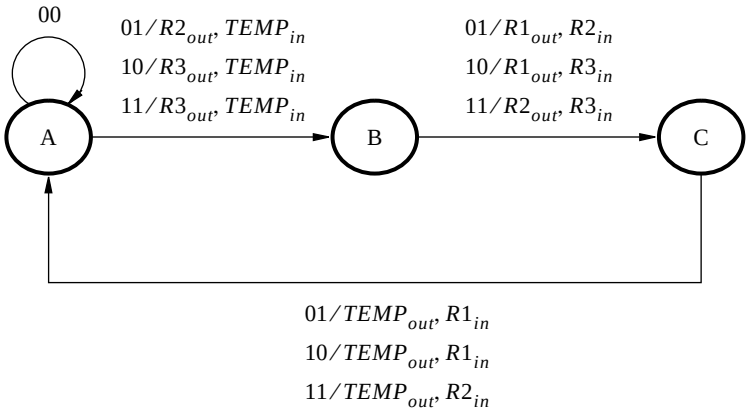
8.37. An ASM chart for the FSM in Figure 8.58 is



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

The next-state expressions are

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

The output expressions are

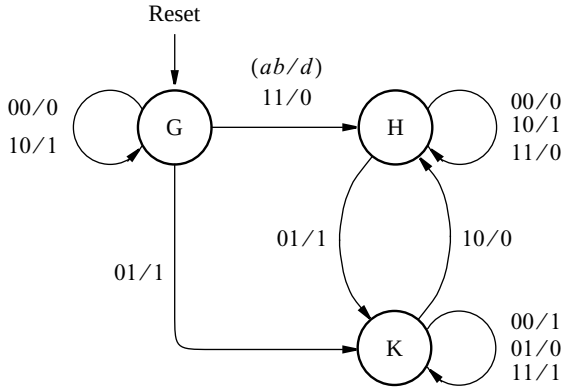
$$\begin{aligned} TEMP_{in} &= ky_1 \\ TEMP_{out} &= ky_3 \\ R1_{out} &= y_2(w_2 \oplus w_1) \\ R1_{in} &= y_3(w_2 \oplus w_1) \\ R2_{out} &= y_1\overline{w}_2w_1 + y_2w_2w_1 \\ R2_{in} &= y_2\overline{w}_2w_1 + y_3w_2w_1 \\ R3_{out} &= y_1w_2 \\ R3_{in} &= y_2w_2 \end{aligned}$$

8.41. VHDL code for the circuit in Figure 8.102 is

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

ENTITY converter IS
    PORT ( B            : IN    STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           Clock, Load  : IN    STD_LOGIC ) ;
           z            : OUT   STD_LOGIC ) ;
END converter ;

ARCHITECTURE Behavior OF converter IS
    TYPE State_type IS (Even, Odd) ;
    SIGNAL y : State_type ;
    SIGNAL Count : STD_LOGIC_VECTOR (2 DOWNTO 0) ;
    SIGNAL Q : STD_LOGIC_VECTOR (7 DOWNTO 0) ;
    SIGNAL Resetn, w, p, Sel : STD_LOGIC ;
BEGIN
    Resetn <= NOT Load ;
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF Load = '1' THEN Q <= B ;
        ELSE
            Genbits: FOR i IN 0 TO 6 LOOP
                Q(i) <= Q(i+1) ;
            END LOOP ;
            Q(7) <= '0' ;
        END IF ;
    END PROCESS ;
    w <= Q(0) ;

    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN Count <= "000" ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            Count <= Count + 1 ;
        END IF ;
    END PROCESS ;
    Sel <= Count(2) AND Count(1) AND Count(0) ;

. . . con't
```

```
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN y <= Even ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            CASE y IS
                WHEN Even =>
                    IF w = '0' THEN y <= Even ;
                    ELSE y <= Odd ;
                    END IF ;
                WHEN Odd =>
                    IF w = '0' THEN y <= Odd ;
                    ELSE y <= Even ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;
    p <= '1' WHEN y = Odd ELSE '0' ;

    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF Sel = '0' THEN z <= w ;
        ELSE z <= p ;
        END IF ;
    END PROCESS ;

END Behavior ;
```

8.42. We can use the scheme given in Figure 8.39. However, instead of adding the vector B in its existing form, we need its 2's complement. This can be done by using the rule for finding 2's complements, in section 5.3.1. Rather than generating the 2's complement of B explicitly, we can change the specification of the Adder FSM to deal with the bits of B using the rule. As a straightforward attempt, we can introduce an extra state to complement the incoming bits of B after the first 1 has been detected. This leads to the following state diagram



The corresponding state table is

Present state	Next state				Output s			
	ab = 00	01	10	11	00	01	10	11
G	G	K	G	H	0	1	1	0
H	H	K	H	H	0	1	1	0
J	K	K	H	K	1	0	0	1

It is apparent that states G and H are equivalent, hence the table can be reduced to

Present state	Next state				Output s			
	ab = 00	01	10	11	00	01	10	11
G	G	K	G	G	0	1	1	0
K	K	K	G	K	1	0	0	1

The state assigned table is

Present state	Next state				Output			
	ab = 00	01	10	11	00	01	10	11
y	Y				s			
0	0	1	0	0	0	1	1	0
1	1	1	0	1	1	0	0	1

The resulting next state and output expressions are

$$\begin{aligned} Y &= \overline{a}b + \overline{a}y + by \\ s &= a \oplus b \oplus y \end{aligned}$$

These expressions define a *full subtractor*, which replaces the *full adder* in Figure 8.43.

8.43. The VHDL code in Figure 8.49 can be used. It is only necessary to modify statements numbered 36 and 40, to correspond to the state diagram derived in problem 8.42. The required code for the Adder FSM becomes

```
29      AdderFSM: PROCESS ( Reset, Clock )
30      BEGIN
31          IF Reset = '1' THEN
32              y <= G ;
33          ELSIF Clock'EVENT AND Clock = '1' THEN
34              CASE y IS
35                  WHEN G =>
36                      IF QA(0) = '0' AND QB(0) = '1' THEN y <= H ;
37                      ELSE y <= G ;
38                      END IF ;
39                  WHEN H =>
40                      IF QA(0) = '1' AND QB(0) = '0' THEN y <= G ;
41                      ELSE y <= H ;
42                      END IF ;
43              END CASE ;
44          END IF ;
45      END PROCESS AdderFSM ;
```

8.44. The state diagram is

Present state	Next state		Output
	$w = 0$	$w = 1$	$z_2 z_1 z_0$
A	A	B	0 0 0
B	B	C	0 0 1
C	C	D	0 1 1
D	D	E	0 1 0
E	E	F	1 1 0
F	F	G	1 1 1
G	G	H	1 0 1
H	H	A	1 0 0

The state-assigned table is

Present state $y_2y_1y_0$	Next state $Y_2Y_1Y_0$		Output $z_2z_1z_0$
	$w=0$	$w=1$	
0 0 0	0 0 0	0 0 1	0 0 0
0 0 1	0 0 1	0 1 1	0 0 1
0 1 1	0 1 1	0 1 0	0 1 1
0 1 0	0 1 0	1 1 0	0 1 0
1 1 0	1 1 0	1 1 1	1 1 0
1 1 1	1 1 1	1 0 1	1 1 1
1 0 1	1 0 1	1 0 0	1 0 1
1 0 0	1 0 0	0 0 0	1 0 0

The next-state expressions are

$$\begin{aligned} Y_2 &= \overline{w}y_2 + y_0y_2 + w\overline{y}_0y_1 \\ Y_1 &= \overline{w}y_1 + w\overline{y}_0y_1 + wy_0\overline{y}_2 \\ Y_0 &= \overline{w}y_0 + wy_1y_2 + w\overline{y}_1\overline{y}_2 \end{aligned}$$

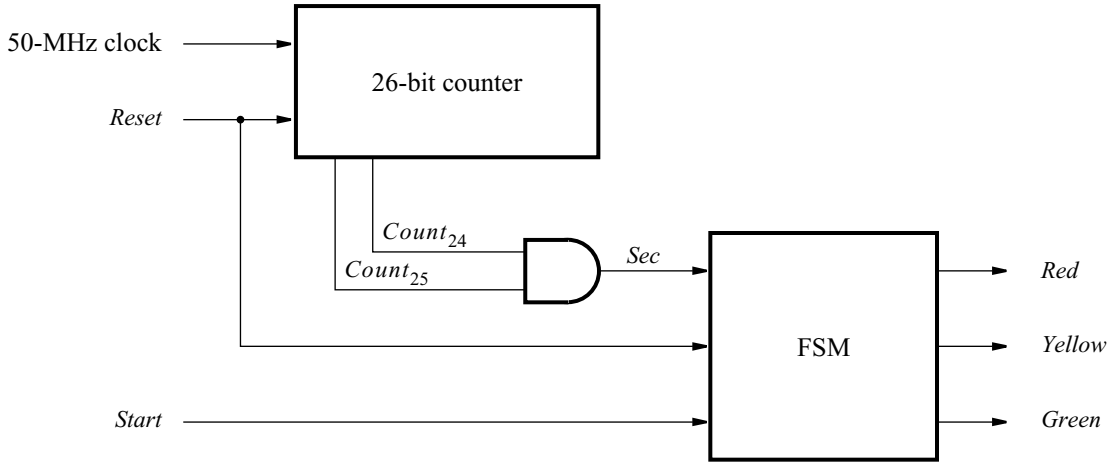
The outputs are: $z_2 = y_2$, $z_1 = y_1$, and $z_0 = y_0$.

8.45. The following VHDL code specifies the desired counter:

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

ENTITY prob8_45 IS
    PORT ( Clock, Resetn, w : IN STD_LOGIC ;
           z : OUT STD_LOGIC_VECTOR(2 DOWNTO 0) ) ;
END prob8_45 ;

ARCHITECTURE Behavior OF prob8_45 IS
    SIGNAL y_present, y_next : STD_LOGIC_VECTOR(2 DOWNTO 0);
    CONSTANT A : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000" ;
    CONSTANT B : STD_LOGIC_VECTOR(2 DOWNTO 0) := "001" ;
    CONSTANT C : STD_LOGIC_VECTOR(2 DOWNTO 0) := "011" ;
    CONSTANT D : STD_LOGIC_VECTOR(2 DOWNTO 0) := "010" ;
    CONSTANT E : STD_LOGIC_VECTOR(2 DOWNTO 0) := "110" ;
    CONSTANT F : STD_LOGIC_VECTOR(2 DOWNTO 0) := "111" ;
    CONSTANT G : STD_LOGIC_VECTOR(2 DOWNTO 0) := "101" ;
    CONSTANT H : STD_LOGIC_VECTOR(2 DOWNTO 0) := "100" ;

. . . con't
```

```
BEGIN
    PROCESS ( w, y_present )
    BEGIN
        CASE y_present IS
            WHEN A =>
                IF w = '0' THEN y_next <= A ;
                ELSE y_next <= B ;
                END IF ;
            WHEN B =>
                IF w = '0' THEN y_next <= B ;
                ELSE y_next <= C ;
                END IF ;
            WHEN C =>
                IF w = '0' THEN y_next <= C ;
                ELSE y_next <= D ;
                END IF ;
            WHEN D =>
                IF w = '0' THEN y_next <= D ;
                ELSE y_next <= E ;
                END IF ;
            WHEN E =>
                IF w = '0' THEN y_next <= E ;
                ELSE y_next <= F ;
                END IF ;
            WHEN F =>
                IF w = '0' THEN y_next <= F ;
                ELSE y_next <= G ;
                END IF ;
            WHEN G =>
                IF w = '0' THEN y_next <= G ;
                ELSE y_next <= H ;
                END IF ;
            WHEN H =>
                IF w = '0' THEN y_next <= H ;
                ELSE y_next <= A ;
                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 <= y_present ;
END Behavior ;
```

8.46. A suitable circuit is



The two most-significant bits of the 26-bit counter become equal to 1 after the elapsed time of just over a second. They are used to generate a ”clock” signal, called *Sec*, for the finite state machine. At each positive edge of the *Sec* signal the FSM changes state as follows

Present state	Next state		Output light
	$Start = 0$	$Start = 1$	
A	A	B	Red
B	C	C	Red
C	D	D	Yellow
D	E	E	Green
E	F	F	Green
F	A	A	Green

8.47. The control circuit can be specified by the following VHDL code:

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

ENTITY prob8_47 IS
    PORT ( Clock, Resetn, Start : IN STD_LOGIC ;
           Red, Yellow, Green : OUT STD_LOGIC ) ;
END prob8_47 ;

ARCHITECTURE Behavior OF prob8_47 IS
    TYPE State_type IS (A, B, C, D, E, F) ;
    SIGNAL y : State_type ;
    SIGNAL Count : STD_LOGIC_VECTOR(25 DOWNTO 0) ;
    SIGNAL Sec : STD_LOGIC ;
BEGIN
    PROCESS ( Resetn, Sec )
    BEGIN
        IF Resetn = '0' THEN
            y <= A ;
        ELSIF (Sec'EVENT AND Sec = '1') THEN
            CASE y IS
                WHEN A =>
                    IF Start = '0' THEN
                        y <= A ;
                    ELSE
                        y <= B ;
                    END IF ;
                WHEN B =>  y <= C ;
                WHEN C =>  y <= D ;
                WHEN D =>  y <= E ;
                WHEN E =>  y <= F ;
                WHEN F =>  y <= A ;
            END CASE ;
        END IF ;
    END PROCESS ;
    PROCESS (Resetn, Clock)
    BEGIN
        IF Resetn = '0' THEN
            Count <= (OTHERS => '0') ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            Count <= Count + 1 ;
        END IF ;
    END PROCESS ;
    Sec <= Count(25) AND Count(24) ;
    Red <= '1' WHEN ((y = A) OR (y = B)) ELSE '0' ;
    Yellow <= '1' WHEN y = C ELSE '0' ;
    Green <= '1' WHEN ((y = D) OR (y = E) OR (y = F)) ELSE '0' ;
END Behavior ;
```