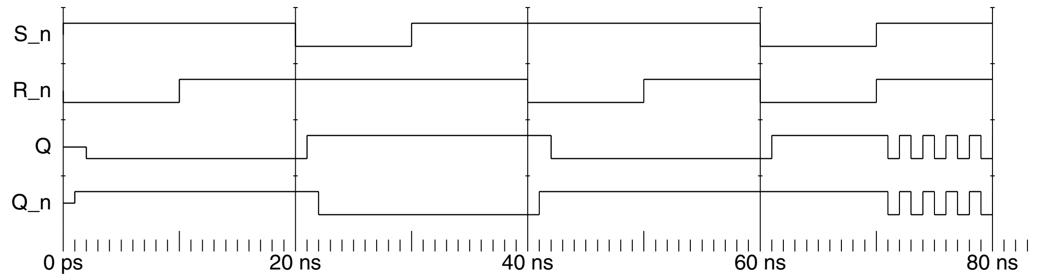
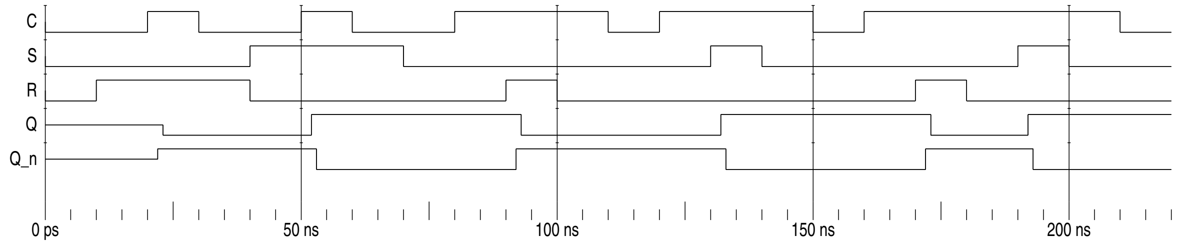
CHAPTER 5

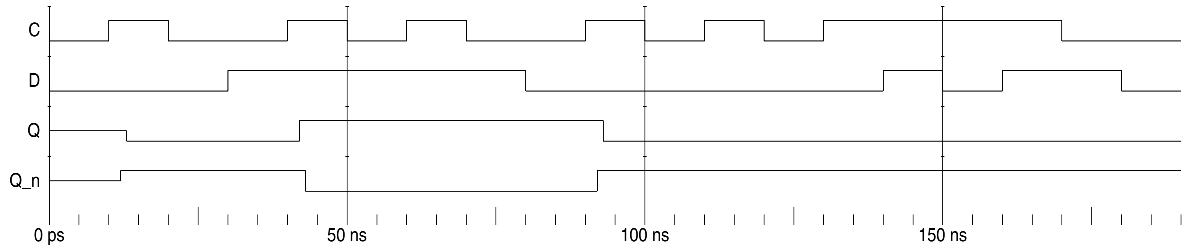
5-1.



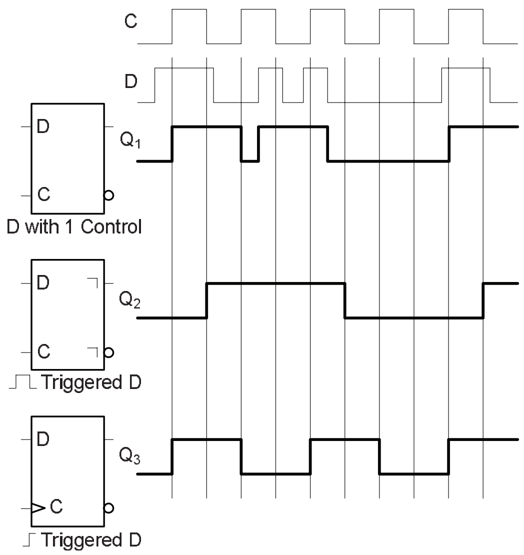
5-2.



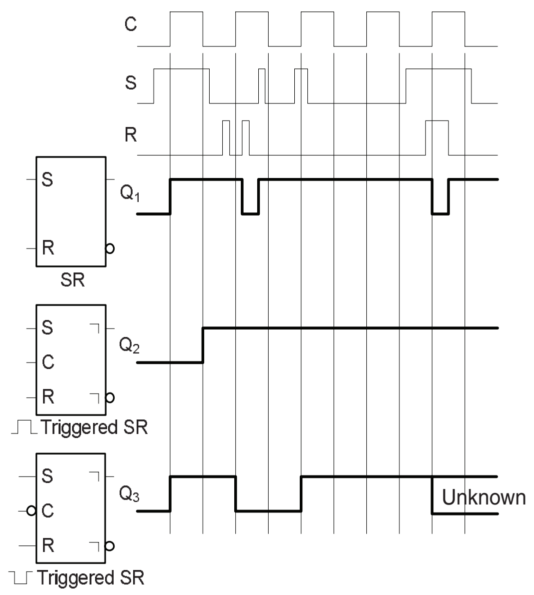
5-3.



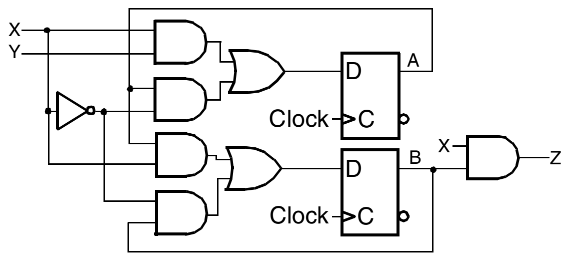
5-4.



5-5.



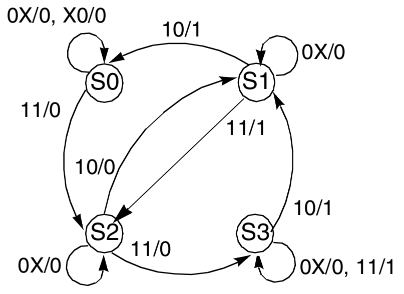
5-6.



Present state		Inputs		Next state		Output
A	*B*	*X*	*Y*	*A*	*B*	*Z*
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	1	0	0
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	1	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	1	1	0
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	0	1	1
1	1	1	1	1	1	1

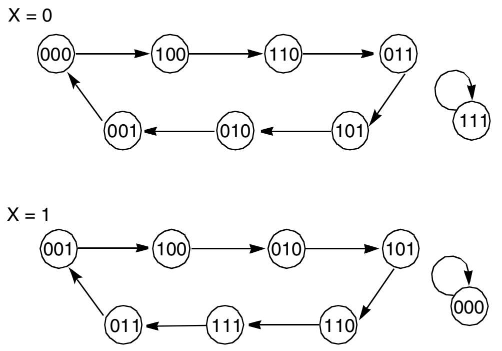
S0 - 00
S1 - 01
S2 - 10
S3 - 11

Format: XY/Z (X = unspecified)



5-7.*

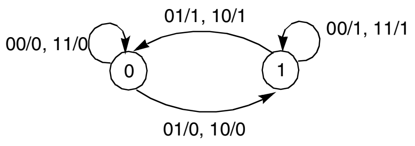
Present state			Input	Next state		
A	*B*	*C*	*X*	*A*	*B*	*C*
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	1	0	1
0	1	1	0	1	0	1
0	1	1	1	0	0	1
1	0	0	0	1	1	0
1	0	0	1	0	1	0
1	0	1	0	0	1	0
1	0	1	1	1	1	0
1	1	0	0	0	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	0	1	1



State diagram is the combination of the above two diagrams.

5-8.

Present state	Inputs		Next state	Output
Q	*X*	*Y*	*Q*	*S*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

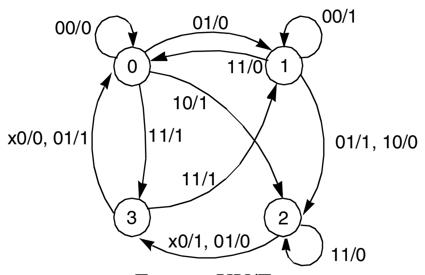


Format: XY/S

5-9.

Present State	00	01	00	00	01	11	00	01	11	10	10
Input	1	0	0	1	1	0	1	1	1	1	0
Output	0	1	0	0	0	1	0	0	0	0	1
Next State	01	00	00	01	11	00	01	11	10	10	00

5-10.



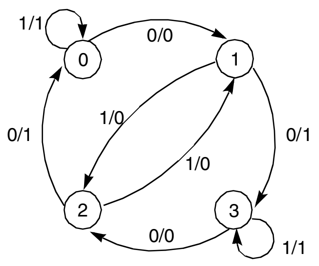
Format: XY/Z

5-11.*

$S_A = B \qquad S_B = \overline{X \oplus A}$

$R_A = \overline{B} \qquad R_B = X \oplus A$

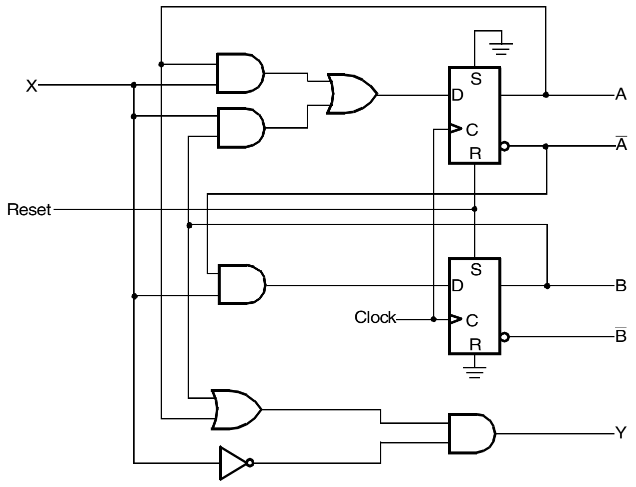
Present state		Input	Next state		Output
A	***B***	***X***	***A***	***B***	***Y***
0	0	0	0	1	0
0	0	1	0	0	1
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1



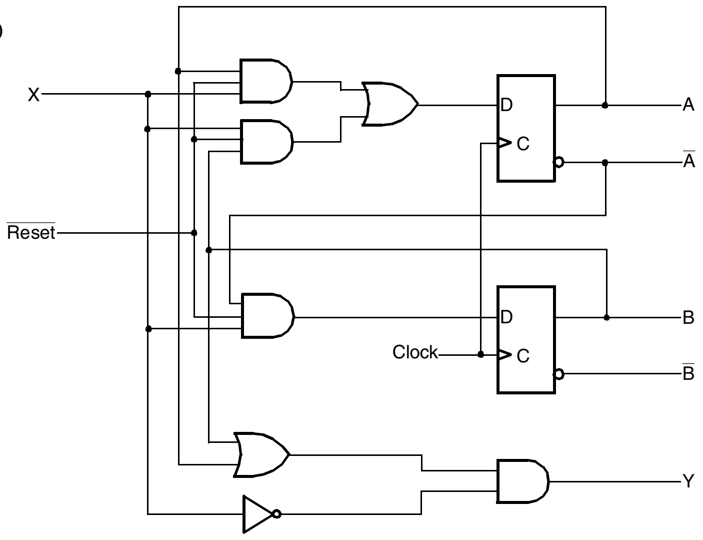
Format: X/Y

5-12.

a)

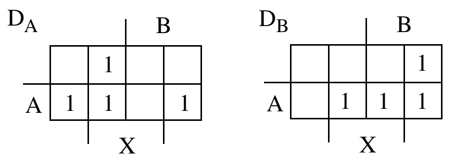


b)



5-13.*

Present state		Input	Next state	
A	*B*	*X*	*A*	*B*
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1



$D_A = A\overline{X} + \overline{B}X$ $D_B = AX + B\overline{X}$

Logic diagram not given.

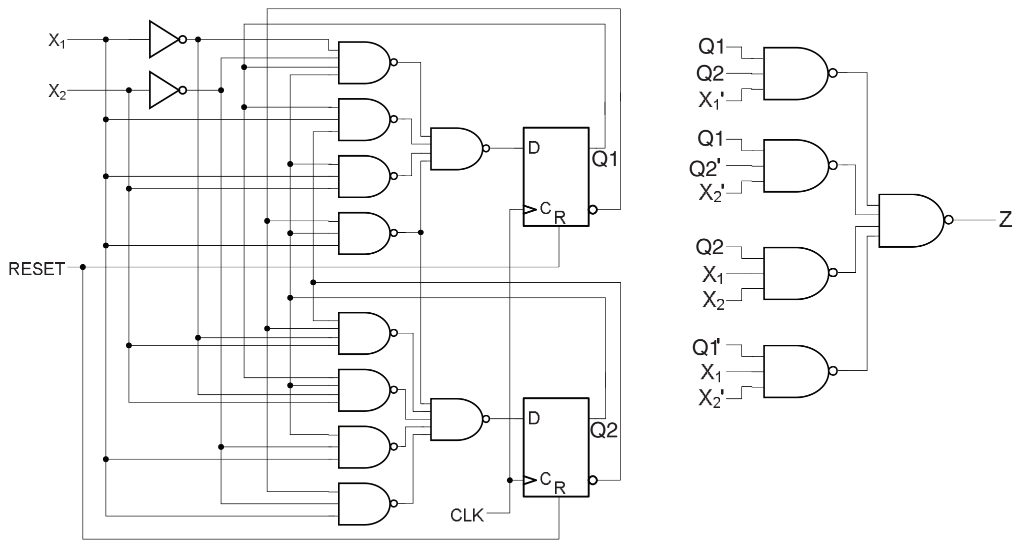
5-14.

For part a) results, replace codes in table below with state name, e.g., 00 with A.

Present state		Inputs		Next state		Output
Q_1	*Q_2*	*X_1*	*X_2*	*$Q_1(t+1)$*	*$Q_2(t+1)$*	*Z*
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	1
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	0	0	1
1	0	0	1	0	0	0
1	0	1	0	1	0	1
1	0	1	1	1	0	0
1	1	0	0	1	0	1
1	1	0	1	0	1	1
1	1	1	0	0	1	0
1	1	1	1	1	0	1

Encoding:

State	Code (Q_1Q_2)
A	0 0
B	0 1
C	1 0
D	1 1



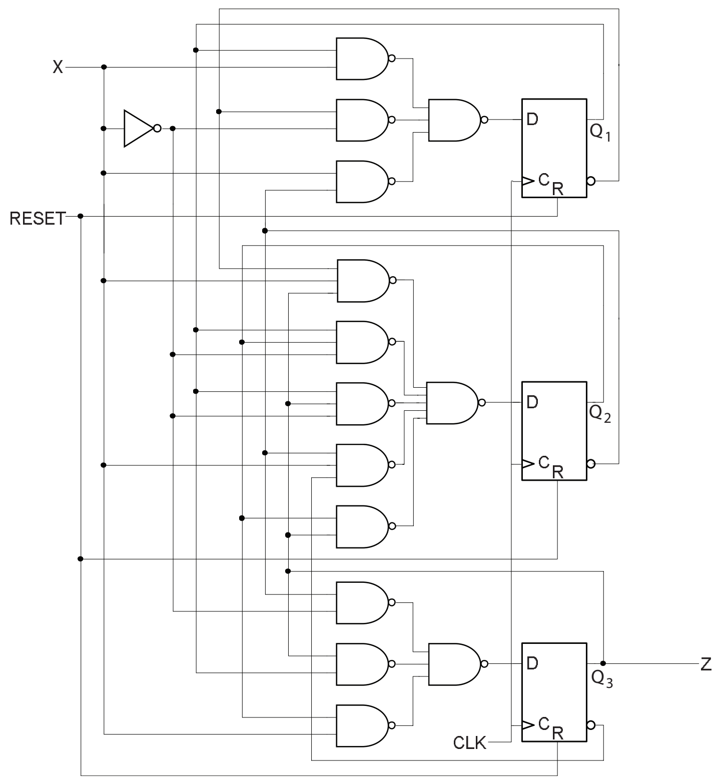
5-15.

Present state	Next State *Q(t+1)* For Input		Output
Q(t)	***X = 0***	***X = 1***	***Z***
A	B	D	0
B	D	C	0
C	A	F	0
D	F	C	1
E	C	E	1
F	E	F	1

Encoding:

State	Code ($Q_1Q_2Q_3$)
A	010
B	100
C	110
D	001
E	011
F	101

Unused states: 000, 111. The state assignment could be different. E.g. states A, B or C could be 000 and states D, E or F could be 111.



5-16.

Encoding:

State	Code ($Q_1Q_2Q_3$)
A	000
B	001
C	010
D	011
E	101
F	110

Present state	Next State *Q(t+1)* For Input		Output
Q(t)	***X = 0***	***X = 1***	***Z***
A	C	E	0
B	E	D	1
C	C	E	0
D	F	A	1
E	B	D	1
F	C	E	0

Present state	Next State $Q(t+1)$ For Input		Output
Q(t)	*X = 0*	*X = 1*	*X*
M	M	N	0
N	N	O	1
O	M	M	0

States A, C, F are equivalent and merged to get state M.
States B and E are equivalent and merged to get state N.
State D becomes state O.

Encoding: Z is a state variable and an output, and
Y is a state variable.

State	Code (Y Z)
M	00
N	01
O	11

$D_Y = \overline{Y}ZX$ $\quad$ $D_Z = \overline{Z}X + \overline{Y}Z$

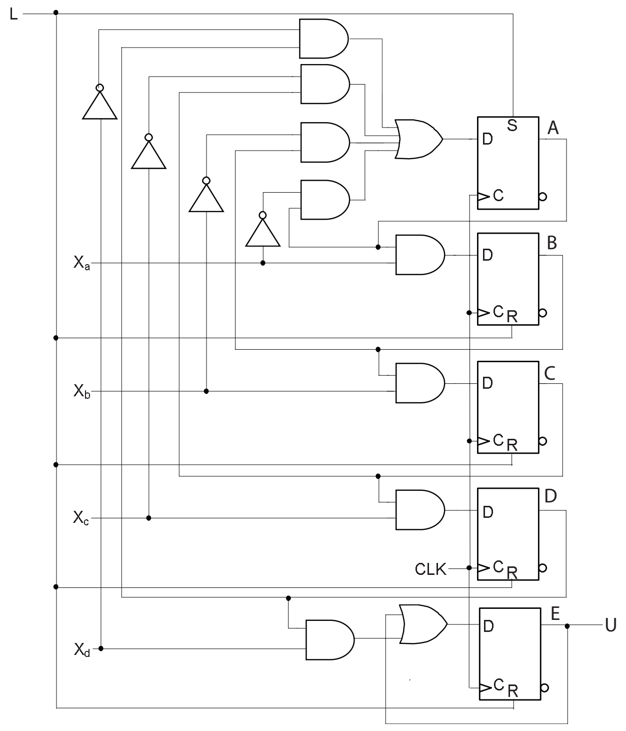
Gate input cost of the original circuit: $21 + 42 = 63$
Gate input cost of the new circuit: $9 + 28 = 37$

5-17.

Present state	Inputs				Next state	Output
ABCDE	*Xa*	*Xb*	*Xc*	*Xd*	*ABCDE*	*U*
10000	0	x	x	x	10000	0
10000	1	x	x	x	01000	0
01000	x	0	x	x	10000	0
01000	x	1	x	x	00100	0
00100	x	x	0	x	10000	0
00100	x	x	1	x	00010	0
00010	x	x	x	0	10000	0
00010	x	x	x	1	00001	0
00001	x	x	x	x	00001	1

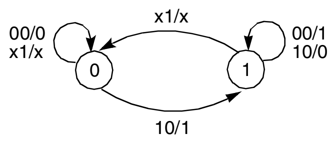
$D_A = A\overline{X}_A + B\overline{X}_B + C\overline{X}_C + D\overline{X}_D$
$D_B = AX_A$
$D_C = BX_B$
$D_D = CX_C$
$D_E = E + DX_D$
$U = E$

Signal L is applied to the asynchronous set/reset inputs on the flip-flops to implement locking.



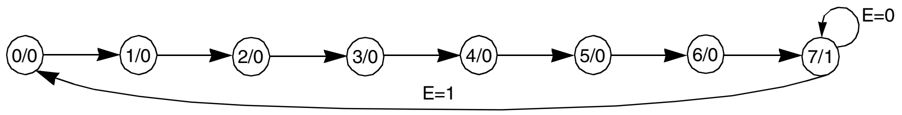
5-18.*

Format: XY/Z (x = unspecified)



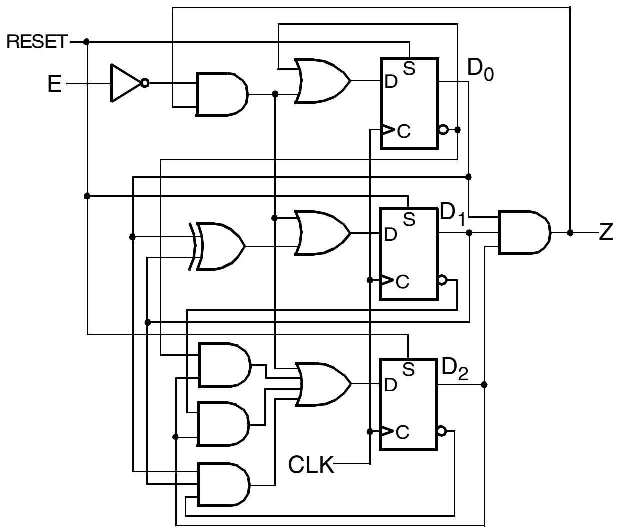
Present state	Inputs		Next state	Output
Q(t)	*X*	*Y*	*Q(t+1)*	*Z*
0	0	0	0	0
0	0	1	0	X
0	1	0	1	1
0	1	1	0	X
1	0	0	1	1
1	0	1	0	X
1	1	0	1	0
1	1	1	0	X

5-19.

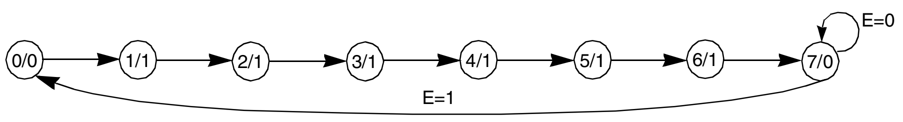


Present state	Next State For Input		Output
$D_2D_1D_0$	*E=0*	*E=1*	*Z*
000	001	001	0
001	010	010	0
010	011	011	0
011	100	100	0
100	101	101	0
101	110	110	0
110	111	111	0
111	111	000	1

The state assignment could be different. E. g., state 7 could be 000 with state 0 001. This would permit use of R inputs on the D flip-flops for RESET.

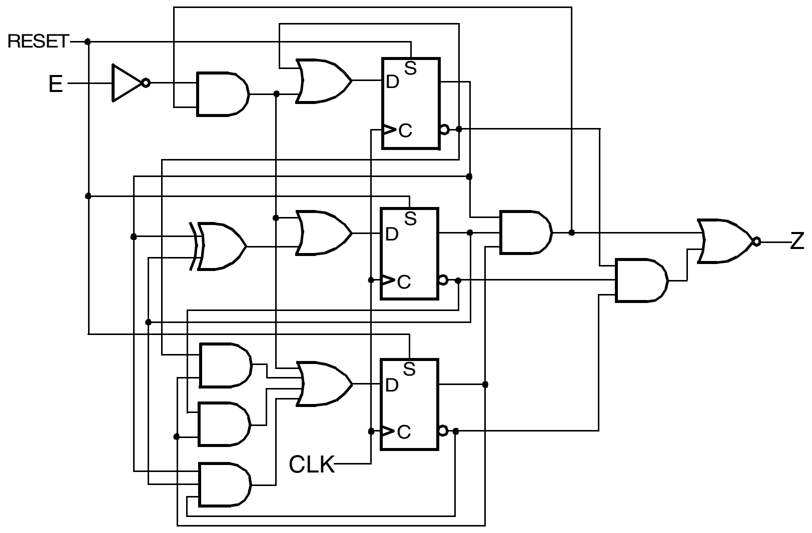


5-20.

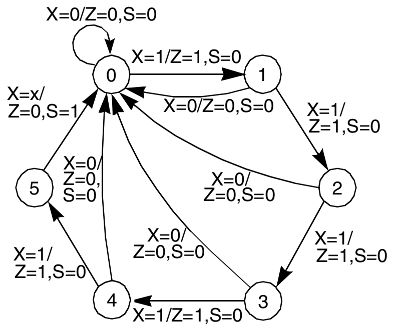


Assumes for E = 0, the output remains at 0.

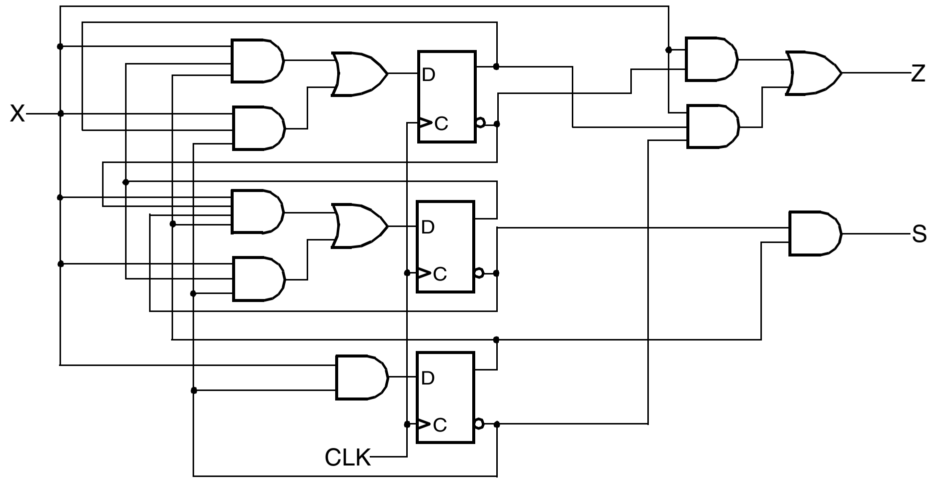
Present state	Next State For Input		Output
$D_2D_1D_0$	*E=0*	*E=1*	*Z*
000	001	001	0
001	010	010	1
010	011	011	1
011	100	100	1
100	101	101	1
101	110	110	1
110	111	111	1
111	111	000	0



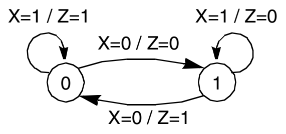
5-21.$^+$



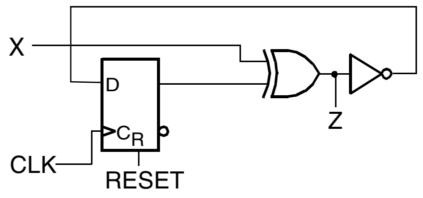
Present state			Input	Next state			Output	
A	*B*	*C*	*X*	*A*	*B*	*C*	*Z*	*S*
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	1	0
0	0	1	0	0	0	0	0	0
0	0	1	1	0	1	0	1	0
0	1	0	0	0	0	0	0	0
0	1	0	1	0	1	1	1	0
0	1	1	0	0	0	0	0	0
0	1	1	1	1	0	0	1	0
1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	1	0
1	0	1	0	0	0	0	0	1
1	0	1	1	0	0	0	0	1



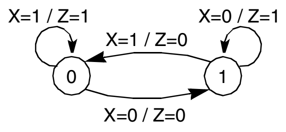
5-22.



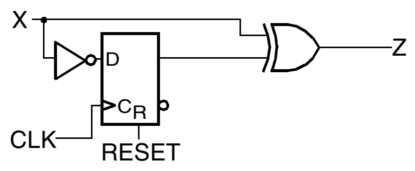
Present state	Input	Next state	Output
A	*X*	*A*	*Z*
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0



5-23.+

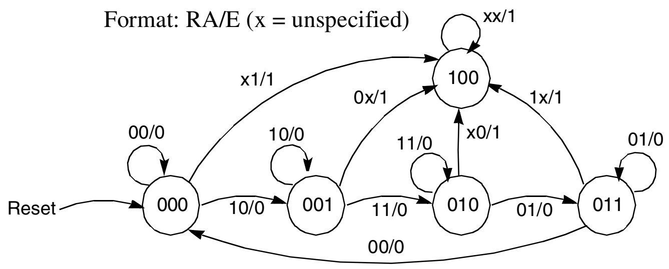


Present state	Input	Next state	Output
A	*X*	*A*	*Z*
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	0



5-24.

Format: RA/E (x = unspecified)

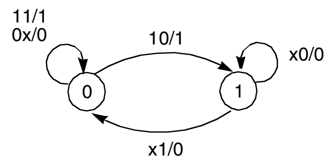


Present state			Inputs		Next state			Output
B	*C*	*D*	*R*	*A*	*B*	*C*	*D*	*E*
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	1
0	0	0	1	0	0	0	1	0
0	0	0	1	1	1	0	0	1
0	0	1	0	0	1	0	0	1
0	0	1	0	1	1	0	0	1
0	0	1	1	0	0	0	1	0
0	0	1	1	1	0	1	0	0
0	1	0	0	0	1	0	0	1
0	1	0	0	1	0	1	1	0
0	1	0	1	0	1	0	0	1
0	1	0	1	1	0	1	0	0
0	1	1	0	0	0	0	0	0
0	1	1	0	1	0	1	1	0
0	1	1	1	0	1	0	0	1
0	1	1	1	1	1	0	0	1
1	0	0	0	0	1	0	0	1
1	0	0	0	1	1	0	0	1
1	0	0	1	0	1	0	0	1
1	0	0	1	1	1	0	0	1

5-25.

Present state	Input		Next state	Output
A	*X*	*Y*	*A*	*Z*
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	0
1	1	1	0	0

Format: XY/Z (x = unspecified)



5-26.*

To use a one-hot assignment, the two flip-flops A and B need to be replaced with four flip-flops Y4, Y3, Y2. Y1.

No Reset State Specified.

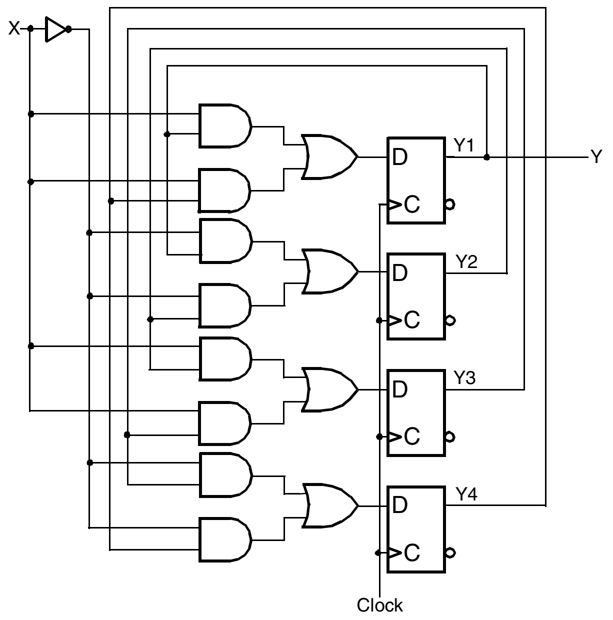
Present State		Input	Next State		Output
A B	*Y4 Y3 Y2 Y1*	*X*	*A' B"*	*Y4'Y3'Y2'Y1*	*Z*
0 0	0 0 0 1	0	0 1	0 0 1 0	1
0 0	0 0 0 1	1	0 0	0 0 0 1	1
0 1	0 0 1 0	0	0 1	0 0 1 0	0
0 1	0 0 1 0	1	1 0	0 1 0 0	0
1 0	0 1 0 0	0	1 1	1 0 0 0	0
1 0	0 1 0 0	1	1 0	0 1 0 0	0
1 1	1 0 0 0	0	1 1	1 0 0 0	0
1 1	1 0 0 0	1	0 0	0 0 0 1	0

D1 = Y1'= X·Y1 + X·Y4

D2 = Y2' = $\overline{X}$·Y1 + $\overline{X}$·Y2

D3 = Y3' = X·Y2 + X·Y3

D4 = Y4' = $\overline{X}$·Y3 + $\overline{X}$·Y4

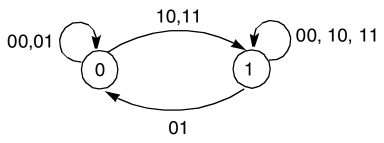


5-27.*

a)

S	R	Q	
0	0	Q	No Change
0	1	0	Reset
1	0	1	Set
1	1	1	Set

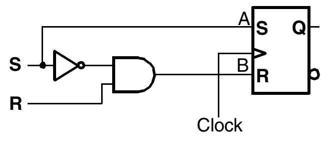
b) Format: SR



c)

Present state	Input		Next state		
Q	***S***	***R***	***Q(t+1)***	***A***	***B***
0	0	0	0	0	x
0	0	1	0	0	x
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	x	0
1	0	1	0	0	1
1	1	0	1	x	0
1	1	1	1	x	0

A = S
B = $\overline{S}$R



5-28.+

Present State ABC	Next State ABC
000	100
001	000
010	XXX
011	001
100	110
101	XXX
110	111
111	011

a) $D_A = \overline{C}$
$D_B = A$
$D_C = B$

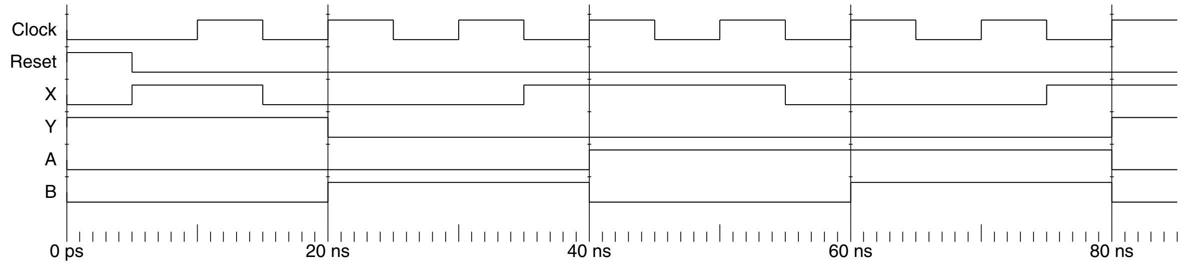
b) Clear A = $\overline{\text{Reset}}$
Clear B = $\overline{\text{Reset}}$
Clear C = $\overline{\text{Reset}}$

c, d, e, f) The circuit is suitable for child's toy, but not for life critical applications. In the case of the child's toy, it is the cheapest implementation. If an error occurs the child just needs to reset it. In life critical applications, the immediate detection of errors is critical. The circuit above enters invalid states for some errors. For a life critical application, additional circuitry is needed for immediate detection of the error (Error = $\overline{A}B\overline{C} + A\overline{B}C$). This circuit using the design in a), does return from the invalid states to a valid state automatically after one or two clock periods.

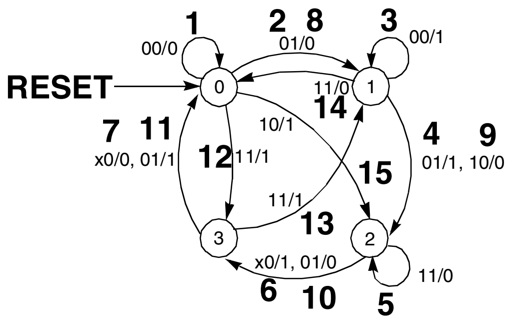
5-29.

	Specification					Verification		
Present state	Input		Next state					Next state
Q	***S***	***R***	***Q(t+1)***	***A***	***B***	***A***	***B***	***Q(t+1)***
0	0	0	0	0	x	0	0	0
0	0	1	0	0	x	0	1	0
0	1	0	1	1	0	1	0	1
0	1	1	1	1	0	1	0	1
1	0	0	1	x	0	0	0	1
1	0	1	0	0	1	0	1	0
1	1	0	1	x	0	1	0	1
1	1	1	1	x	0	1	0	1

5-30.



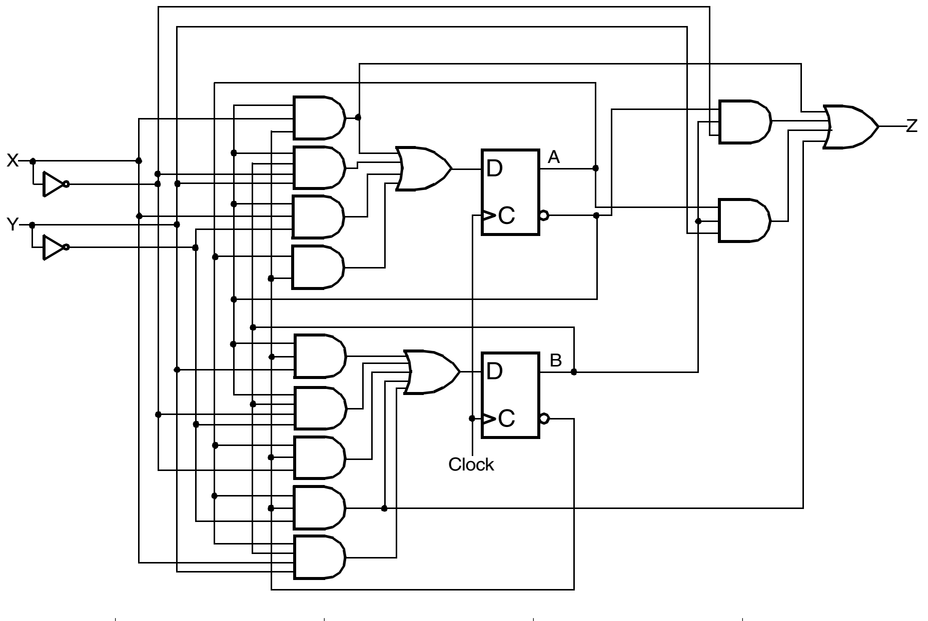
5-31.*

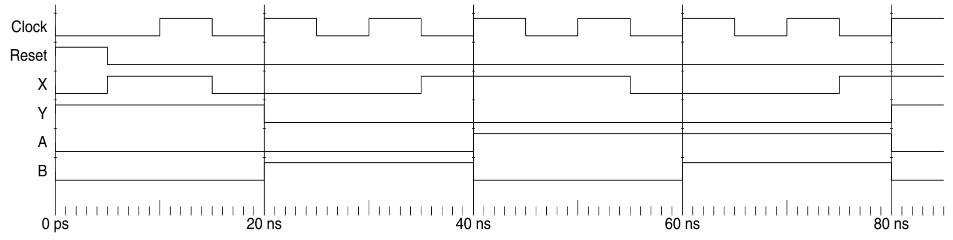


Reset, 00, 01, 00, 01, 11, x0, x0, 01, 10, 01, 01, 11, 11, 11, 10.

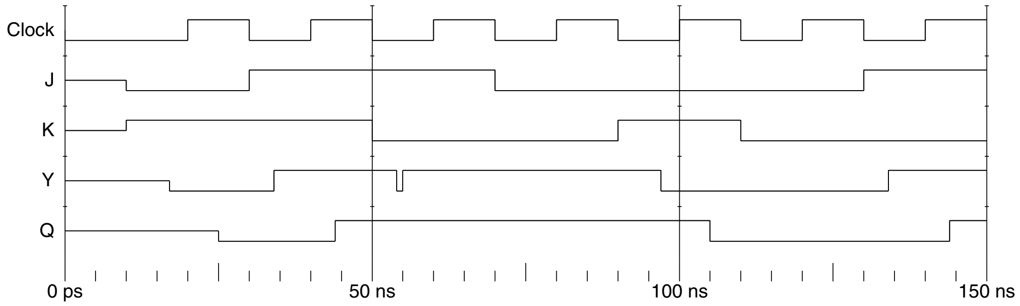
Format: XY/Z

5-32.





5-33.*



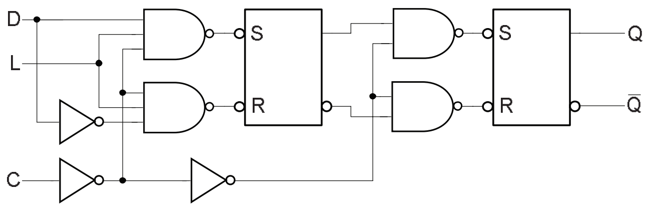
This simulation was performed without initializing the state of the latches of the flip-flop beforehand. Each gate in the flip-flop implementation has a delay of 1 ns. The interaction of these delays with the input change times produced a narrow pulse in Y at about 55 ns. In this case, the pulse is not harmful since it dies out well before the positive clock edge occurs. Nevertheless, a thorough examination of such a pulse to be sure that it does not represent a design error or important timing problem is critical.

5-34.

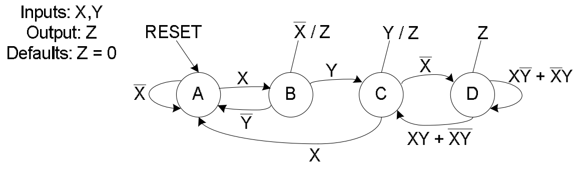
Function table for the LH flip-flop.

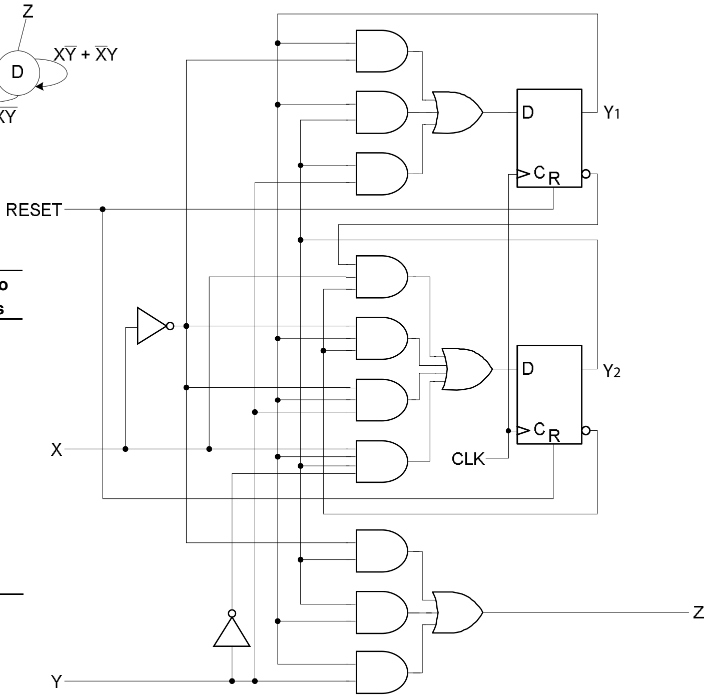
C	L	D	Q(t+1)	S	R
0	X	X	No change	X	X
1	0	X	No change	0	0
1	1	0	0	0	1
1	1	1	1	1	0

LH flip-flop and SR latch in a master-slave circuit:



5-35.





State table:

State	State Code	Transition Condition	Next State	State Code	Non-zero Outputs
A	00	$\overline{X}$	A	00	
		X	B	01	
B	01				$\overline{X}$ / Z
		$\overline{Y}$	A	00	
		Y	C	10	
C	10				Y / Z
		$\overline{X}$	D	11	
		X	A	00	
D	11				Z
		$X\overline{Y} + \overline{X}Y$	D	11	
		$XY + \overline{X}\overline{Y}$	C	10	

5-36.

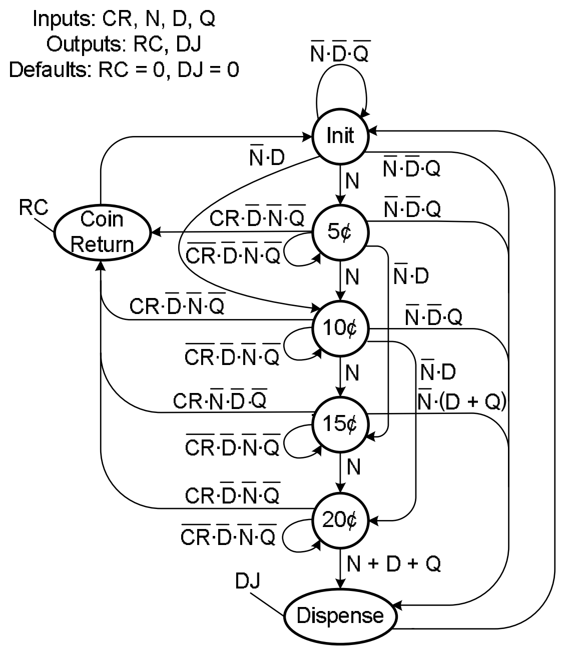
Constraint 1 checks on the transition conditions (TC):

Init: There is one pair of TCs to check:	$(\overline{START} + STOP)\, START\cdot \overline{STOP} = 0$
Fill_1: There are three pairs of TCs to check:	$\overline{L1} \cdot \overline{STOP} \cdot STOP = 0,$
	$\overline{L1} \cdot \overline{STOP} \cdot L1\cdot \overline{STOP} = 0,$
	$STOP \cdot L1\cdot \overline{STOP} = 0$
Fill_2: There are six pairs of TCs to check:	$L2 \cdot \overline{NI} \cdot \overline{STOP} \cdot STOP = 0,$
	$L2 \cdot \overline{NI} \cdot \overline{STOP} \cdot \overline{L2} \cdot \overline{STOP} = 0,$
	$L2 \cdot \overline{NI} \cdot \overline{STOP} \cdot L2\cdot NI \cdot \overline{STOP} = 0,$
	$STOP \cdot \overline{L2} \cdot \overline{STOP} = 0,$
	$STOP \cdot L2\cdot NI \cdot \overline{STOP} = 0,$
	$\overline{L2} \cdot \overline{STOP} \cdot L2\cdot NI \cdot \overline{STOP} = 0$
Fill_3: There are three pairs of TCs to check:	$\overline{L3} \cdot \overline{STOP} \cdot STOP = 0,$
	$\overline{L3} \cdot \overline{STOP} \cdot L3\cdot \overline{STOP} = 0,$
	$STOP \cdot L3\cdot \overline{STOP} = 0$
Mix: There are three pairs of TCs to check:	$\overline{TZ} \cdot \overline{STOP} \cdot STOP = 0,$
	$\overline{TZ} \cdot \overline{STOP} \cdot TZ\cdot \overline{STOP} = 0,$
	$STOP \cdot TZ\cdot \overline{STOP} = 0$
Empty: There is one pair of TCs to check:	$\overline{L0} \cdot \overline{STOP} \cdot (L0 + STOP) = 0$

Constraint 2 checks on the transition conditions (TC):

Init:	$\overline{START} + STOP + START\cdot \overline{STOP} = 1$
Fill_1:	$\overline{L1} \cdot \overline{STOP} + STOP + L1\cdot \overline{STOP} = 1$
Fill_2:	$L2 \cdot \overline{NI} \cdot \overline{STOP} + STOP + \overline{L2} \cdot \overline{STOP} + L2\cdot NI \cdot \overline{STOP} = 1$
Fill_3:	$\overline{L3} \cdot \overline{STOP} + STOP + L3\cdot \overline{STOP} = 1$
Mix:	$\overline{TZ} \cdot \overline{STOP} + STOP + TZ\cdot \overline{STOP} = 1$
Empty:	$\overline{L0} \cdot \overline{STOP} + L0 + STOP = 1$

5-37.*



5-38.

State	State Code	Transition Condition	Next State	State Code	Non-zero Outputs
Init	1000000	N	5¢	0010000	
		D	10¢	0001000	
		Q	Dispense	0000001	
		$\overline{N}\cdot\overline{D}\cdot\overline{Q}$	Init	1000000	
Coin_Return					RC
	0100000	1	Init	1000000	
5¢	0010000	N	10¢	0001000	
		D	15¢	0000100	
		Q	Dispense	0000001	
		CR	Coin_Return	0100000	
		$\overline{CR}\cdot\overline{N}\cdot\overline{D}\cdot\overline{Q}$	5¢	0010000	
10¢	0001000	N	15¢	0000100	
		D	20¢	0000010	
		Q	Dispense	0000001	
		CR	Coin_Return	0100000	
		$\overline{CR}\cdot\overline{N}\cdot\overline{D}\cdot\overline{Q}$	10¢	0001000	
15¢	0000100	N	20¢	0000010	
		D + Q	Dispense	0000001	
		CR	Coin_Return	0100000	
		$\overline{CR}\cdot\overline{N}\cdot\overline{D}\cdot\overline{Q}$	15¢	0000100	
20¢	0000010	N + D + Q	Dispense	0000001	
		CR	Coin_Return	0100000	
		$\overline{CR}\cdot\overline{N}\cdot\overline{D}\cdot\overline{Q}$	20¢	0000010	
Dispense					DJ
	0000001	1	Init	1000000	

D flip-flop inputs:

Init (t + 1) = *Init* $\cdot\overline{N}\cdot\overline{D}\cdot\overline{Q}$ + *Coin_Return* + *Dispense*

Coin_Return (t + 1) = *5¢* · CR + *10¢* · CR + *15¢* · CR + *20¢* · CR

5¢ (t + 1) = *Init* · N + *5¢* $\cdot\overline{CR}\cdot\overline{N}\cdot\overline{D}\cdot\overline{Q}$

10¢ (t + 1) = *Init* · D + *5¢* · N + *10¢* $\cdot\overline{CR}\cdot\overline{N}\cdot\overline{D}\cdot\overline{Q}$

15¢ (t + 1) = *5¢* · D + *10¢* · N + *15¢* $\cdot\overline{CR}\cdot\overline{N}\cdot\overline{D}\cdot\overline{Q}$

20¢ (t + 1) = *10¢* · D + *15¢* · N + *20¢* $\cdot\overline{CR}\cdot\overline{N}\cdot\overline{D}\cdot\overline{Q}$

Dispense (t + 1) = Q + *15¢* · D + *20¢* · D + *20¢* · N

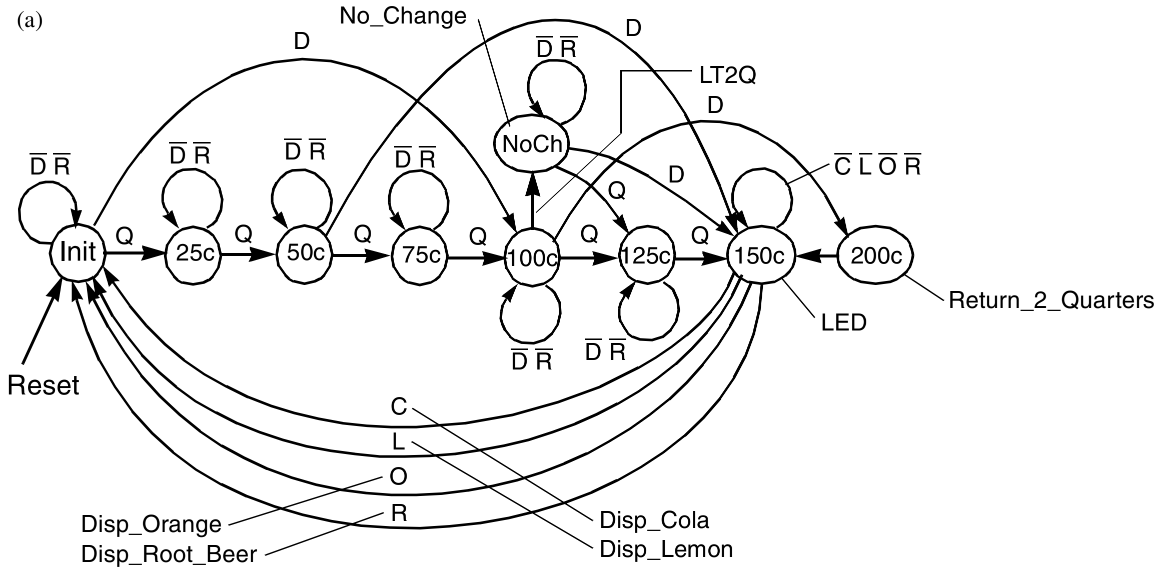
Outputs:

RC = *Coin_Return*

DJ = *Dispense*

5-39.

(a)



(b) Some possible changes to the specification follow. 1. Provide appropriate change and dispense soda for the cases in which 75 cents and 125 cents followed by a dollar have been deposited. Provide a coin return button for the event that the user is out of quarters and dollars before the 150c state is reached. 3. Change the No Change warning to cover all added cases in 1.

5-40.*

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux_4to1 is
    port (
        S: in STD_LOGIC_VECTOR (1 downto 0);
        D: in STD_LOGIC_VECTOR (3 downto 0);
        Y: out STD_LOGIC
    );
end mux_4to1;

-- (continued in the next column)

architecture mux_4to1_arch of mux_4to1 is
begin

process (S, D)
    begin
    case S is
        when "00" => Y <= D(0);
        when "01" => Y <= D(1);
        when "10" => Y <= D(2);
        when "11" => Y <= D(3);
        when others => null;
    end case;

end process;
end mux_4to1_arch;
```

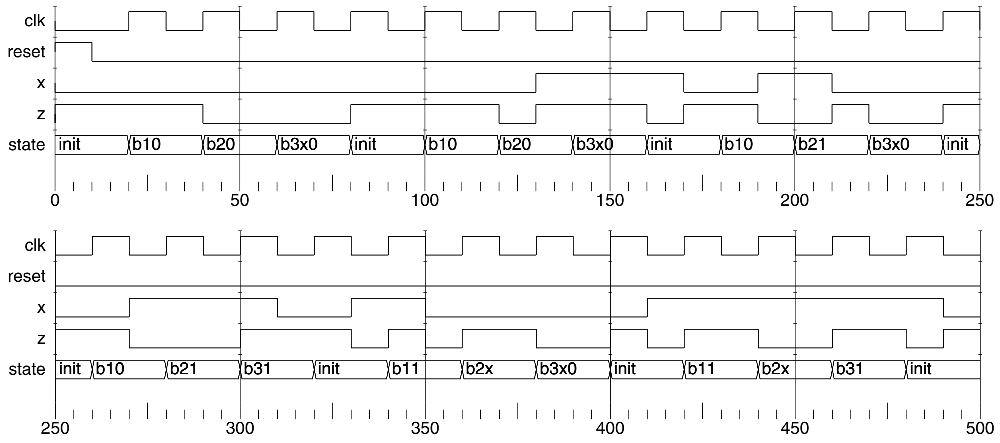
5-41.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux_4to1 is
    port (
        S: in STD_LOGIC_VECTOR (1 downto 0);
        D: in STD_LOGIC_VECTOR (3 downto 0);
        Y: out STD_LOGIC
    );
end mux_4to1;

-- (continued in the next column)

architecture mux_4to1_arch of mux_4to1 is
begin

process (S, D)
    begin

    if S = "00" then Y <= D(0);
    elsif S = "01" then Y <= D(1);
    elsif S = "10" then Y <= D(2);
    elsif S = "11" then Y <= D(3);
    else null;
    end if;

end process;
end mux_4to1_arch;
```

5-42.+

```
library IEEE;
use IEEE.std_logic_1164.all;
entity serial_BCD_Ex3  is
   port (clk, reset, X : in STD_LOGIC;
         Z  : out STD_LOGIC);
end serial_BCD_Ex3;

architecture process_3 of serial_BCD_Ex3  is
type state_type is (Init, B10, B11, B20, B21, B2X,
B3X0, B31);
signal state, next_state: state_type;
begin
-- Process 1 - state register
state_register: process (clk, reset)
begin
    if (reset = '1') then
        state <= Init;
    else if (CLK'event and CLK='1') then
        state <= next_state;
        end if;
    end if;
end process;
-- (continued in the next column)

-- Process 2 - next state function
next_state_func: process (X, state)
begin
    case state is
        when Init =>
    if (X = '0') then
        next_state <= B10;
    else
        next_state <= B11;
    end if;
        when B10 =>
    if (X = '0') then
        next_state <= B20;
    else
        next_state <= B21;
    end if;
        when B11 =>
        next_state <= B2X;
        when B20 =>
        next_state <= B3X0;
        when B21 =>
    if (X = '0') then
        next_state <= B3X0;
```

```
      else
           next_state <= B31;
      end if;
           when B2X =>
      if (X = '0') then
           next_state <= B3X0;
      else
           next_state <= B31;
      end if;
           when B3X0 =>
           next_state <= Init;
           when B31 =>
           next_state <= Init;
end case;
end process;

-- Process 3 -output function
output_func: process (X, state)
begin
      case state is
           when Init =>
      if (X = '0') then
           Z <= '1';
      else
           Z <= '0';
      end if;
           when B10 =>
-- (continued in the next column)
      if (X = '0') then
           Z <= '1';else
           Z<= '0';
      end if;
           when B11 =>
           Z <= X;
           when B20 =>
           Z <= X;
           when B21 =>
      if (X = '0') then
           Z <= '1';
      else
           Z <= '0';
      end if;
           when B2X =>
      if (X = '0') then
           Z <= '1';
      else
           Z <= '0';
      end if;
           when B3X0 =>
           Z <= X;
           when B31 =>
           Z <= '1';
end case;
end process;

end process_3;
```



5-43.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prob_5_43  is
    port (clk, reset : in STD_LOGIC;
         X : in STD_LOGIC_VECTOR(2 downto 1) ;
         Z  : out STD_LOGIC);
end prob_5_43;

architecture process_3 of prob_5_43  is
type state_type is (A, B, C, D);
signal next_state, state : state_type;
begin

-- Continued in next column
-- Process 1 - state register
state_register: process (clk, reset)
begin
     if (reset = '1') then
          state <= A;
     elsif (clk'event and clk = '1') then
          state <= next_state;
     end if;
end process;

-- Process 2 - next state function
next_state_func: process (X, state)
begin
-- Continued on next page
```

```
    case state is
        when A =>
            case X is
                when "00" =>
                    next_state <= A;
                when "01" =>
                    next_state <= B;
                when "10" =>
                    next_state <= B;
                when "11" =>
                    next_state <= A;
                when others => next_state <= A;
                end case;
        when B =>
            case X is
                when "00" =>
                    next_state <= A;
                when "01" =>
                    next_state <= A;
                when "10" =>
                    next_state <= D;
                when "11" =>
                    next_state <= D;
                when others => next_state <= A;
            end case;
        when C =>
            case X is
                when "00" =>
                    next_state <= A;
                when "01" =>
                    next_state <= A;
                when "10" =>
                    next_state <= C;
                when "11" =>
                    next_state <= C;
                when others => next_state <= A;
            end case;
        when D =>
            case X is
                when "00" =>
                    next_state <= C;
                when "01" =>
                    next_state <= B;
                when "10" =>
                    next_state <= B;
                when "11" =>
                    next_state <= C;
                when others => next_state <= A;
            end case;
    end case;
end process;

-- Continued in next column
```

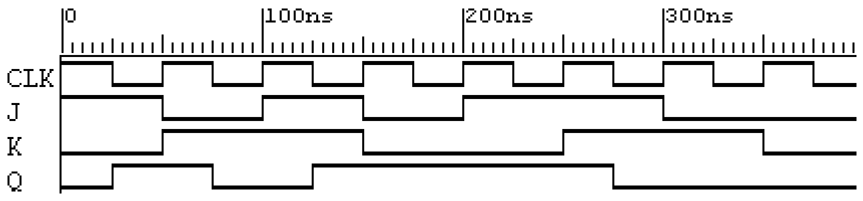
```
-- Process 3 -output function
output_func: process (X, state)
begin
    case state is
        when A =>
            case X is
                when "00" =>
                    Z <= '0';
                when "01" =>
                    Z <='0';
                when "10" =>
                    Z <= '1';
                when "11" =>
                    Z <= '0';
                when others => Z <= 'X';
            end case;
        when B =>
            case X is
                when "00" =>
                    Z <= '0';
                when "01" =>
                    Z <='0';
                when "10" =>
                    Z <= '1';
                when "11" =>
                    Z <= '1';
                when others => Z <= 'X';
          end case;
        when C =>
            case X is
                when "00" =>
                    Z <= '1';
                when "01" =>
                    Z <='0';
                when "10" =>
                    Z <= '1';
                when "11" =>
                    Z <= '0';
                when others => Z <= 'X';
            end case;
        when D =>
            case X is
                when "00" =>
                    Z <= '1';
                when "01" =>
                    Z <='1';
                when "10" =>
                    Z <= '0';
                when "11" =>
                    Z <= '1';
                when others => Z <= 'X';
            end case;
    end case;
end process;
end process_3;
```

5-44.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prob_5_44  is
   port (clk, reset,
         X : in STD_LOGIC;
         Z  : out STD_LOGIC);
end prob_5_44;

architecture process_3 of prob_5_44  is
type state_type is (A, B, C, D, E, F);
signal state, next_state: state_type;
begin

-- Process 1 - state register
state_register: process (clk, reset)
begin
     if (reset = '1') then
          state <= A;
     else if (CLK'event and CLK='1') then
          state <= next_state;
          end if;
     end if;
end process;

-- Process 2 - next state function
next_state_func: process (X, state)
begin
     case state is
          when A =>
               if X = '0' then
                    next_state <= B;
               else
                    next_state <= D;
               end if;
          when B =>
               if X = '0' then
                    next_state <= D;
               else
                    next_state <= C;
               end if;
          when C =>
               if X = '0' then
                    next_state <= A;
-- Continued in next column
               else
                    next_state <= F;
               end if;
          when D =>
               if X = '0' then
                    next_state <= F;
               else
                    next_state <= C;
               end if;
          when E =>
               if X = '0' then
                    next_state <= C;
               else
                    next_state <= E;
               end if;
          when F =>
               if X = '0' then
                    next_state <= E;
               else
                    next_state <= F;
               end if;
     end case;
end process;

-- Process 3 -output function
output_func: process (X, state)
begin
     case state is
          when A =>
               Z <= '0';
          when B =>
               Z <= '0';
          when C =>
               Z <= '0';
          when D =>
               Z <= '1';
          when E =>
               Z <= '1';
          when F =>
               Z <= '1';
     end case;
end process;
end process_3;
```

5-45.*

```
library IEEE;
use IEEE.std_logic_1164.all;
entity jkff is
   port (
      J,K,CLK: in STD_LOGIC;
      Q: out STD_LOGIC
   );
end jkff;

architecture jkff_arch of jkff is
signal q_out: std_logic;
begin

state_register: process (CLK)
begin
   if CLK'event and CLK='0' then  --CLK falling edge

-- (continued in the next column)
      case J is
           when '0' =>
                if K = '1' then
                     q_out <= '0';
                end if;
           when '1' =>
                if K = '0' then
                     q_out <= '1';
                else
                     q_out <= not q_out;
                end if;
           when others => null;
      end case;
   end if;
end process;

Q <= q_out;
end jkff_arch;
```



5-46. (Errata: Replace "5-10" with "5-11")

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prob_5_46  is
   port (clk, reset, NI, Start, Stop, L0, L1, L2, L3, TZ :
in STD_LOGIC;
          MX, PST, TM, V1, V2, V3, VE : out
STD_LOGIC);
end prob_5_46;

architecture process_3 of prob_5_46 is
type state_type is (Init, Fill_1, Fill_2, Fill_3, Mix,
Empty);
signal state, next_state: state_type;
begin

-- Process 1 - state register
state_register: process (clk, reset)
begin
     if (reset = '1') then
          state <= Init;
     else if (CLK'event and CLK='1') then
          state <= next_state;
          end if;
     end if;
end process;

-- Process 2 - next state function
next_state_func: process (NI, Start, Stop, L0, L1, L2,
L3, TZ, state)
begin
     if Stop = '1' then
          next_state <= Init;
     else
          case state is
               when Init =>
                    if Start = '1' then
                         next_state <= Fill_1;
                    else
                         next_state <= Init;
                    end if;
               when Fill_1 =>
                    if L1 = '1' then
                         next_state <= Fill_2;
                    else
                         next_state <= Fill_1;
                    end if;
          when Fill_2 =>
                    if L2 = '1' then
                         if NI = '1'then
                              next_state <= Fill_3;
                         else
                              next_state <= Mix;
                         end if;
                    else
                         next_state <= Fill_2;
                    end if;
-- Continued in next column
               when Fill_3 =>
                    if L3 = '1' then
                         next_state <= Mix;
                    else
                         next_state <= Fill_3;
                    end if;
               when Mix =>
                    if TZ = '1' then
                         next_state <= Empty;
                    else
                         next_state <= Mix;
                    end if;
               when Empty =>
                    if L0 = '1' then
                         next_state <= Init;
                    else
                         next_state <= Empty;
                    end if;
          end case;
     end if;
end process;

-- Process 3 - output function
output_func: process (L2, L3, NI, TZ, Stop, state)
begin
     MX <= '0';
     PST <='0';
     TM <='0';
     V1 <='0';
     V2 <='0';
     V3 <='0';
     VE <='0';
     case state is
        when Init =>
          when Fill_1 =>
               V1 <= '1';
          when Fill_2 =>
             V2 <= '1';
               if ((L2 and not NI and not Stop) = '1') then
                    PST <= '1';
               end if;
          when Fill_3 =>
               V3 <= '1';
               if ((L3 and not Stop) = '1') then
                    PST <= '1';
               end if;
          when Mix =>
               MX <= '1';
               if ((not TZ and not Stop) = '1') then
                    TM <= '1';
               end if;
          when Empty =>
               VE <= '1';
     end case;
end process;
end process_3;
```

5-47.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prob_5_47  is
   port (clk, reset, CR, N, D, Q  : in STD_LOGIC;
          DJ, RC : out STD_LOGIC);
end prob_5_47;

architecture process_3 of prob_5_47  is
type state_type is (S0, S5, S10, S15, S20, S25, RT);
signal state, next_state: state_type;
begin

-- Process 1 - state register
state_register: process (clk, reset)
begin
     if (reset = '1') then
          state <= S0;
     else if (CLK'event and CLK='1') then
          state <= next_state;
          end if;
     end if;
end process;

-- Process 2 - next state function
next_state_func: process (CR, N, D, Q, state)
begin
     case state is
          when S0 =>
               if N = '1' then
                    next_state <= S5;
               elsif D = '1' then
                    next_state <= S10;
               elsif Q = '1' then
                    next_state <= S25;
               else
                    next_state <= S0;
               end if;
          when S5 =>
               if CR = '1' then
                    next_state <= RT;
               elsif N = '1' then
                    next_state <= S10;
               elsif D = '1' then
                    next_state <= S15;
               elsif Q = '1' then
                    next_state <= S25;
               else
                    next_state <= S5;
               end if;
          when S10 =>
               if CR = '1' then
                    next_state <= RT;
               elsif N = '1' then
                    next_state <= S15;
               elsif D = '1' then
                    next_state <= S20;
               elsif Q = '1' then
                    next_state <= S25;
               else
                    next_state <= S10;
               end if;
-- Continued in next column
          when S15 =>
               if CR = '1' then
                    next_state <= RT;
               elsif N = '1' then
                    next_state <= S20;
               elsif D = '1' then
                    next_state <= S25;
               elsif Q = '1' then
                    next_state <= S25;
               else
                    next_state <= S15;
               end if;
          when S20 =>
               if CR = '1' then
                    next_state <= RT;
               elsif N = '1' then
                    next_state <= S25;
               elsif D = '1' then
                    next_state <= S25;
               elsif Q = '1' then
                    next_state <= S25;
               else
                    next_state <= S20;
               end if;
          when S25 =>
               next_state <= S0;
          when RT =>
               next_state <= S0;
     end case;
end process;

-- Process 3 - output function
output_func: process (CR, N, D, Q, state)
begin
    DJ <= '0';
     RC <= '0';
     case state is
          when S25 =>
               DJ <= '1';
          when RT =>
               RC  <= '1';
          when others => null;
     end case;
end process;
end process_3;
```

5-48.

```
module problem_6_38 (S, D, Y) ;

input [1:0] S ;
input [3:0] D ;
output Y;
reg Y ;

// (continued in the next column)

always @(S or D)
     begin
     case (S)
          2'b00 : Y <= D[0] ;
          2'b01 : Y <= D[1] ;
          2'b10 : Y <= D[2] ;
          2'b11 : Y <= D[3] ;
     endcase;
     end
endmodule
```

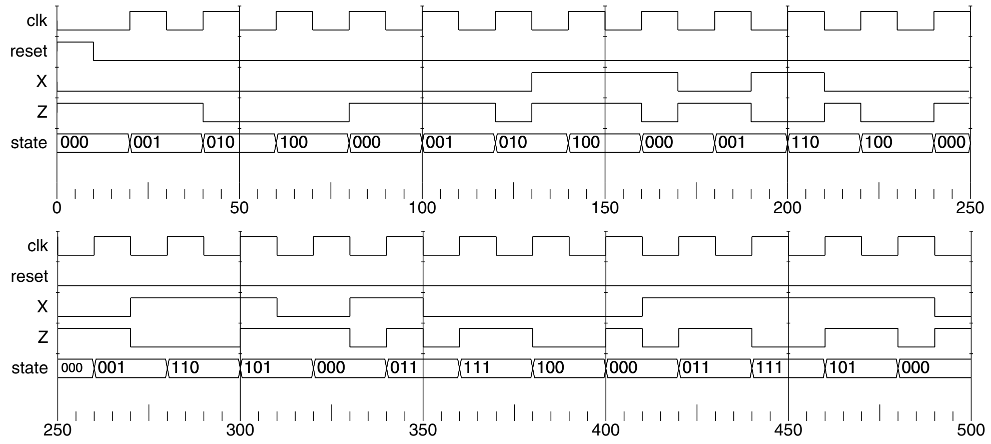
5-49.*

```
module problem_6_39 (S, D, Y) ;

input [1:0] S ;
input [3:0] D ;
output Y;
reg Y ;

// (continued in the next column)

always @(S or D)
     begin
          if (S == 2'b00) Y <= D[0];
          else if (S == 2'b01) Y <= D[1];
          else if (S == 2'b10) Y <= D[2];
          else Y <= D[3];

     end
endmodule
```

5-50.$^+$

```
//Serial BCD to Excess 3 Converter
module serial_BCD_Ex3(clk, reset, X, Z);
input clk, reset, X;
output Z;

reg[2:0] state, next_state;
parameter Init = 3'b000, B10 = 3'b001,
B11=3'b011, B20= 3'b010, B21 = 3'b110,
B2X = 3'b111, B3X0 = 3'b100, B31 = 3'b101;
reg Z;

// State Register
always@(posedge clk or posedge reset)
begin
if (reset == 1)
     state <= Init;
else
     state <= next_state;
end

// Next StateFunction
always@(X or state)
begin
     case (state)
     Init: if (X == 0)
               next_state <= B10;
          else
               next_state <= B11;
     B10: if (X == 0)
               next_state <= B20;
          else
               next_state <= B21;
     B11: next_state <= B2X;
     B20: next_state <= B3X0;
     B21: if (X == 0)
               next_state <= B3X0;
          else
               next_state <= B31;
// (continued in the next column)
     B2X: if (X ==0)
               next_state <= B3X0;
          else
               next_state <= B31;

     B3X0: next_state <= Init;
     B31: next_state <= Init;
endcase
end

// Output Function
always@(X or state)
begin
     case (state)
     Init: if (X == 0)
               Z <= 1;
          else
               Z <= 0;
     B10: if (X == 0)
               Z <= 1 ;
          else
               Z <= 0;
     B11: Z <= X;
     B20: Z <= X;
     B21: if (X == 0)
               Z <= 1;
          else
               Z <= 0;
     B2X: if (X == 0)
          Z <= 1;
     else
          Z <= 0;
     B3X0: Z <= X;
     B31: Z <= 1;
endcase
end
endmodule
```

State Assignment	
State Code	**State Name**
000	Init
001	B10
010	B20
011	B11
100	B3X0
101	B31
110	B21
111	B2X



5-51.

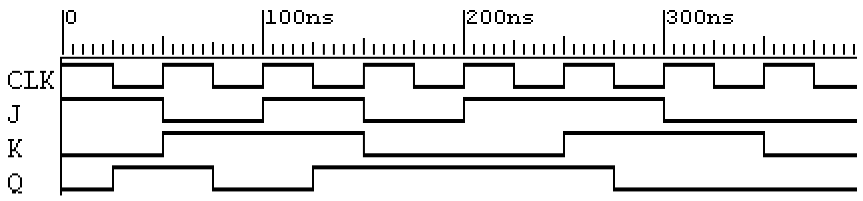
```
// State Diagram in Figure 5-40 using Verilog
module prob_5_51 (clk, reset, X, Z);
input clk, reset;
input[1:2] X;
output Z;

reg[1:0] state, next_state;
parameter A = 2'b00, B = 2'b01, C = 2'b11, D = 2'b10;
reg Z;

// State Register
always@(posedge clk or posedge reset)
begin
if (reset == 1)
    state <= A;
else
    state <= next_state;
end

// Next StateFunction
always@(X or state)
begin
    case (state)
    A: if (X == 2'b01 | X == 2'b10)
            next_state <= B;
        else
            next_state <= A;
    B: if (X == 2'b10 | X == 2'b11)
            next_state <= D;
        else
            next_state <= A;
    C: if (X == 2'b00 | X == 2'b01)
// (continued in the next column)
            next_state <= A;
        else
            next_state <= C;
    D: if (X == 2'b00 | X == 2'b11)
        next_state <= C;
        else
            next_state <= B;
    endcase
end

// Output Function
always@(X or state)
begin
    case (state)
    A: if (X == 2'b01 | X == 2'b00 | X == 2'b11)
            Z <= 0;
        else
            Z <= 1;
    B: if (X == 2'b10 | X == 2'b11)
            Z <= 1;
        else
            Z <= 0;
    C: if (X == 2'b00 | X == 2'b10)
            Z <= 1;
        else
            Z <= 0;
    D: if (X == 2'b00 | X == 2'b11 | X == 01)
            Z <= 1;
        else
            Z = 0;
    endcase
end
endmodule
```

5-52.

```
// State Diagram in Figure 5-41 using Verilog
module prob_5_52 (clk, reset, X, Z);
input clk, reset;
input X;
output Z;
reg[2:0] state, next_state;
parameter A = 3'b000, B = 3'b001,
C = 3'b010, D = 3'b011, E = 3'b100,
F = 3'b101;
reg Z;
// State Register
always@(posedge clk or posedge reset)
begin
if (reset == 1)
    state <= A;
else
    state <= next_state;
end
// Next StateFunction
always@(X or state)
begin
    case (state)
    A: if (X == 1)
            next_state <= D;
        else
            next_state <= B;
    B: if (X == 1)
            next_state <= C;
        else
            next_state <= D;
C: if (X == 1)
(continued in the next column)
            next_state <= F;
        else
            next_state <= A;
    D: if (X == 1)
            next_state <= C;
        else
            next_state <= F;
    E: if (X == 1)
            next_state <= E;
        else
            next_state <= C;
    F: if (X == 1)
            next_state <= F;
        else
            next_state <= E;
    endcase
end

// Output Function
always@(X or state)
begin
    case (state)
        A: Z <= 0;
        B: Z <= 0;
        C: Z <= 0;
        D: Z <= 1;
        E: Z <= 1;
        F: Z <= 1;
    endcase
end
endmodule
```

5-53.*

```
module JK_FF (J, K, CLK, Q) ;

input J, K, CLK ;
output Q;
reg Q;

always @(negedge CLK)
    case (J)
        0'b0: Q <= K ? 0: Q;
        1'b1: Q <= K ? ~Q: 1;
    endcase
endmodule
```



5-54.

```
// State Machine for Batch Mixing System (Figure 5-29)
module batch_mixing_system (clk, reset, START, STOP, L0,
L1, L2, L3, NI, TZ, V1, V2, V3, PST, MX, TM, VE);
input clk, reset, START, STOP, L0, L1, L2, L3, NI, TZ;
output V1, V2, V3, PST, MX, TM, VE;
reg V1, V2, V3, PST, MX, TM, VE;
reg[2:0] state, next_state;
parameter Init = 3'b000, Fill_1 = 3'b001,
Fill_2 = 3'b010, Fill_3 = 3'b011, Mix = 3'b100,
Empty = 3'b101;
// State Register
always@(posedge clk or posedge reset)
begin
// (continued in the next column)
if (reset == 1)
    state <= Init;
else
    state <= next_state;
end
// Next StateFunction
always@( START or STOP or L0 or L1 or L2 or L3 or TZ or
state)
begin
    case (state)
    Init: if (START == 1 & STOP == 0)
            next_state <= Fill_1;
        else
            next_state <= Init; // (continued on the next page)
```

```
 Fill_1: if (STOP == 1)
                    next_state <= Init;
            else if (L1 == 1)
                    next_state <= Fill_2;
                else
                    next_state <= Fill_1;
    Fill_2: if (STOP == 1)
                    next_state <= Init;
            else if (L2 == 1)
                      if (NI == 1)
                            next_state <= Fill_3;
                      else
                            next_state <= Mix;
                  else
                      next_state <= Fill_2;
    Fill_3: if (STOP == 1)
                    next_state <= Init;
            else if (L3 == 1)
                    next_state <= Mix;
                else
                    next_state <= Fill_2;
    Mix: if (STOP == 1)
                    next_state <= Init;
            else if (TZ == 1)
                    next_state <= Empty;
                else
                    next_state <= Mix;
    Empty: if (STOP == 1 | L0 == 1)
                next_state <= Init;
            else
                next_state <= Empty;
 // (continued in the next column)
    endcase
end
// Output Function
always@(L2 or NI or STOP or L3 or TZ or state)
begin
    case (state)
        Fill_1: V1 <= 0;
        Fill_2: begin
            V2 <= 0;
            if (L2 & ~ NI & ~STOP)
                PST <= 1;
            else
                PST <= 0;
            end
        Fill_3: begin
            V3 <= 0;
            if (L3 & ~STOP)
                PST <= 1;
            else
                PST <= 0;
            end
        Mix:  begin
            MX <= 1;
            if (~TZ & ~STOP)
                TM <= 1;
            else
                TM <= 0;
            end
        Empty: VE <= 1;
    endcase
end
endmodule
```

5-55.

```
// State Machine for Jawbreaker Vending Machine
module jawbreaker_vending_machine (clk, reset, CR, N, D, Q,
RC, DJ);
input clk, reset, CR, N, D, Q;
output RC, DJ;
reg RC, DJ;
reg[6:0] state, next_state;
parameter Init = 7'b0000001, S5c = 7'b0000010,
S10c = 7'b0000100, S15c = 7'b0001000, S20c = 7'b0010000,
Dispense = 7'b0100000, Coin_Return = 7'b1000000;

// State Register
always@(posedge clk or posedge reset)
begin
if (reset == 1)
    state <= Init;
else
    state <= next_state;
end
// Next StateFunction
always@(CR or N or D or Q or state)
begin
    case (state)
    Init: if (N == 1)
                next_state <= S5c;
        else if (D == 1)
                next_state <= S10c;
            else if (Q == 1)
                    next_state <= Dispense;
                else
                    next_state <= Init;
// (continued in the next column)
S5c: if (N == 1)
            next_state <= S10c;
        else if (D == 1)
                next_state <= S15c;
            else if (Q == 1)
                    next_state <= Dispense;
                else if (CR == 1)
                        next_state <= Coin_Return;
                    else
                        next_state <= S5c;
S10c: if (N == 1)
            next_state <= S15c;
        else if (D == 1)
                next_state <= S20c;
            else if (Q == 1)
                    next_state <= Dispense;
                else if (CR == 1)
                        next_state <= Coin_Return;
                    else
                        next_state <= S10c;
S15c: if (N == 1)
            next_state <= S20c;
        else if (D == 1)
                next_state <= Dispense;
            else if (Q == 1)
                    next_state <= Dispense;
                else if (CR == 1)
                        next_state <= Coin_Return;
                    else
                        next_state <= S15c;
S20c: if (N == 1)
            next_state <= Dispense;
```

```
      else if (D == 1)
            next_state <= Dispense;
                 else if (Q == 1)
                        next_state <= Dispense;
                        else if (CR == 1)
                               next_state <= Coin_Return;
                               else
                                   next_state <= S20c;
      Dispense: next_state <= Init;
      Coin_Return: next_state <= Init;
  endcase
end
```

(continued in the next column)

```
// Output Function
always@(state)
begin
  RC <= 0;
  DJ <= 0;
  case (state)
    Dispense: DJ <= 1;
    Coin_Return: RC <= 1;
  endcase
end
endmodule
```