CHAPTER 7

7-1.

(a) R1 + 2's complement of R2 = 2^n + R1 - R2. If R1 ≥ R2, the result is ≥ 2^n. The 2^n gives C = 1.

R1 + 2's complement of R2 = 2^n + R1 - R2, if R1 < R2, the result is < 2^n giving C = 0.

(b) If C = 1 then R1 ≥ R2 and there is no borrow.

If C = 0 then R1 < R2 and there is a borrow. Thus, the borrow is the complement of the C status bit.

7-2. *

1001 1001	
<u>1100 0011</u>	
1000 0001	AND
1101 1011	OR
0101 1010	XOR

7-3.

(a) AND, 1010 1010 1010 1010 (b) OR, 0000 0000 0000 1111

(c) XOR, 1111 1111 0000 0000

7-4.*

sl 1001 0100 sr 0110 0101

7-5.*

Q_i remains connected to MUX data input 0. Connect D_i to MUX data input 1 instead of Mux data input 3. Connect Q_{i-1} to MUX data input 2 instead of MUX data input 1. Finally, 0 is connected to MUX data input 3.

7-6.*

a) 1000, 0100, 0010, 0001, 1000. ...

b) # States = n

7-7.

a) 000, 100, 110, 111, 011, 001, 000, ...

b) # States = $2n$

7-8.

a) 8 b) 3

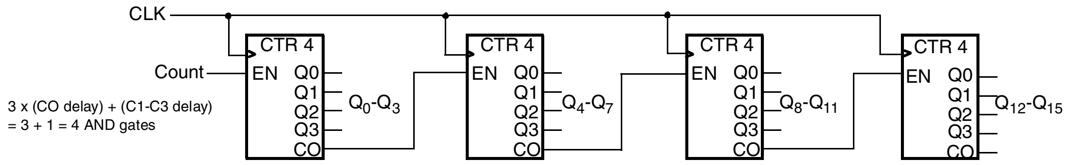
7-9.+

Examine an n-bit ripple counter and an n-bit synchronous counter. If either of these counters cycles through all of its states, there are $2(2^n) = 2^{n+1}$ transitions for the clock, and there are $2^{n+1} - 2$ total transitions for all flip-flop outputs. For the ripple counter, the clock transitions occur on the input of only one stage, the 0th stage. For the synchronous counter, the clock transistions occur on the inputs to all of the n stages. Combining the transition counts above, the ratio of the input + output transitions for the synchronous counter compared to the ripple counter is:

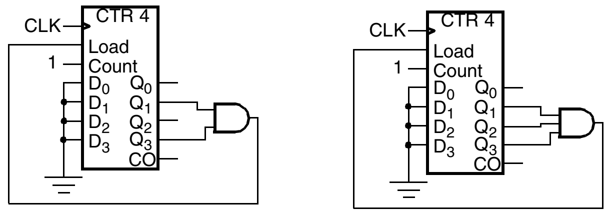
$[n\,2^{n+1} + 2^{n+1} - 2]/[2^{n+1} + 2^{n+1} - 2] \approx (n + 1)\,2^{n+1}/2\,(2^{n+1}) = (n+1)/2$

Thus, the power dissipated by the synchronous counter is at least as large as that dissipated by the ripple counter in all cases and grows more rapidly with the number of stages.

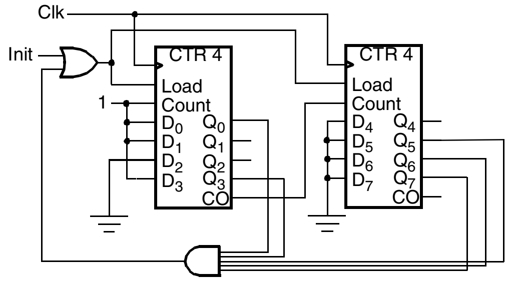
7-10.



7-11.

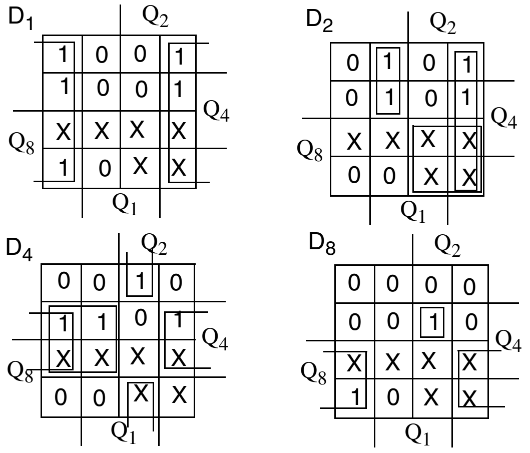


7-12.



7-13.*

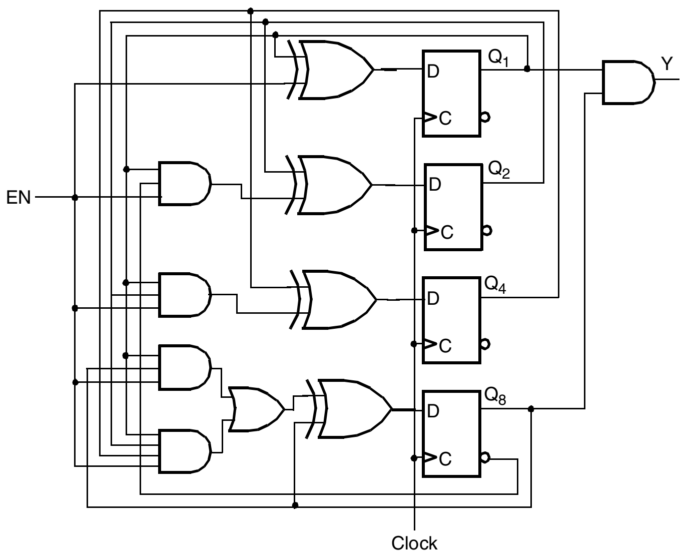
The equations given on page 364-5 can be manipulated into SOP form as follows: $D_1 = \overline{Q_1}$, $D_2 = Q_2 \oplus Q_1\overline{Q_8} = Q_1\overline{Q_2}\,\overline{Q_8} + \overline{Q_1}Q_2 + Q_2Q_8$, $D_4 = Q_4 \oplus Q_1Q_2 = Q_1Q_2\overline{Q_4} + \overline{Q_1}Q_4 + \overline{Q_2}Q_4$, $D_8 = Q_8 \oplus (Q_1Q_8 + Q_1Q_2Q_4) = \overline{Q_8}(Q_1Q_8+Q_1Q_2Q_4) + Q_8(\overline{Q_1} + \overline{Q_8})(\overline{Q_1} + \overline{Q_2} + \overline{Q_4}) = Q_1Q_2Q_4\overline{Q_8} + \overline{Q_1}\,Q_8$. These equations are mapped onto the K-maps for Table 7-9 below and meet the specifications given by the maps and the table.



To add the enable, change D1 to:

$D_1 = Q_1 \oplus EN$.

For the other three functions, AND EN with the expression XORed with the state variable. The circuit below results.



7-14.*

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

a) $D_B = C$

$D_C = \overline{B}\,\overline{C}$

b) $D_A = BC + A\overline{C}$

$D_B = \overline{A}\,\overline{B}C + B\overline{C}$

$D_C = \overline{C}$

7-15.

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

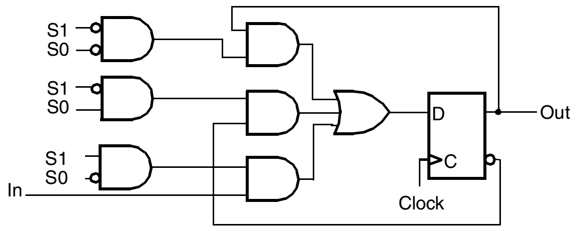
$D_A = A\overline{B} + A\overline{C} + \overline{A}BC$

$D_B = \overline{B}$

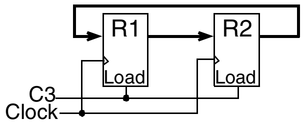
$D_C = \overline{B}C + B\overline{C}$

7-16.

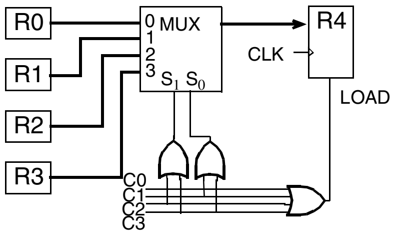
The basic cell of the register is as follows:



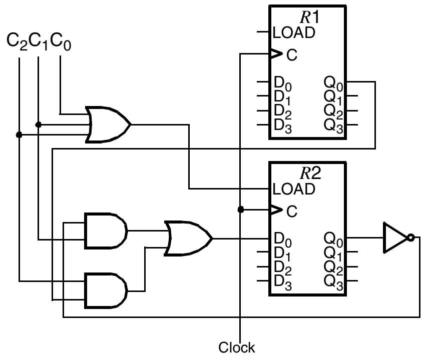
7-17.*



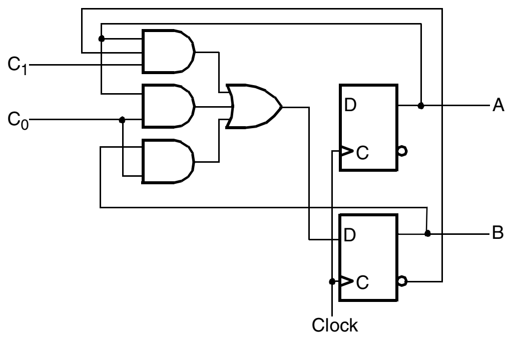
7-18.



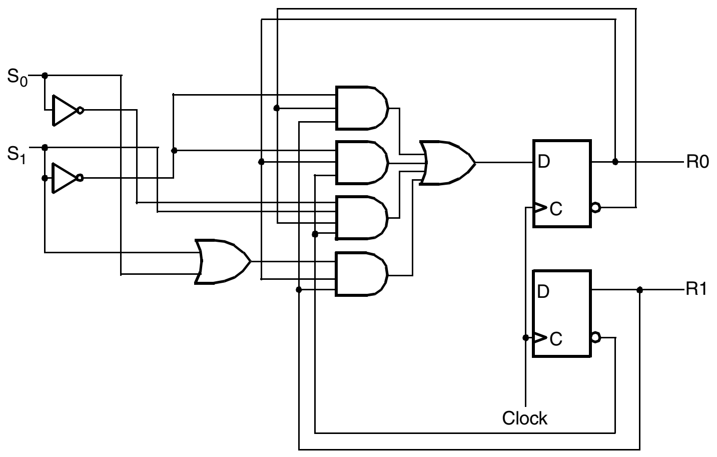
7-19.*



7-20. (Errata: Change “register A” to “register B”)

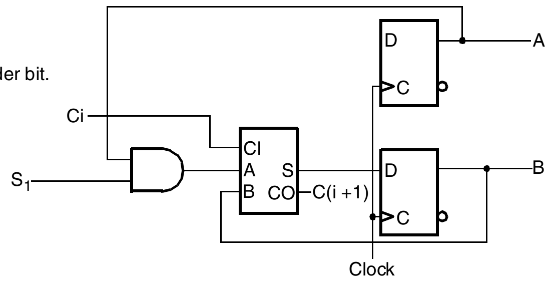


7-21.

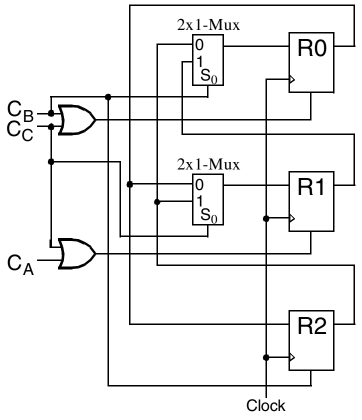


7-22.

Ci = S_0 for lowest order bit.

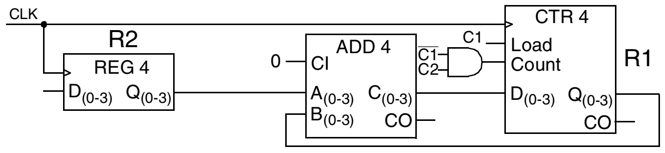


7-23.

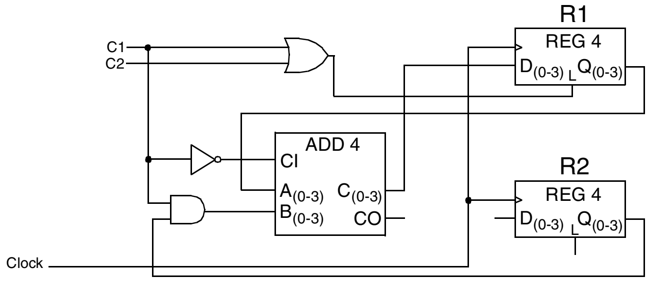


7-24.*

a)



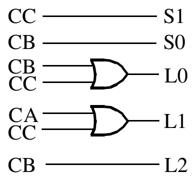
b)

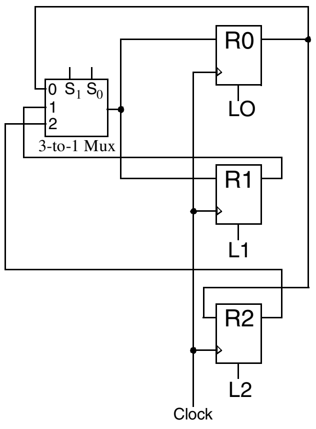


7-25.

The register transfer logic is as follows:

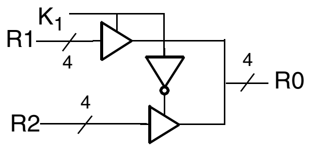
Operation	Select		Load		
	S1	S0	L0	L1	L2
CA: R1 <- R0	0	0	0	1	0
CB: R0 <- R1, R2 <- R0	0	1	1	0	1
CC: R1 <- R2, R0 <- R2	1	0	1	1	0





7-26.

Replace multiplexer with:

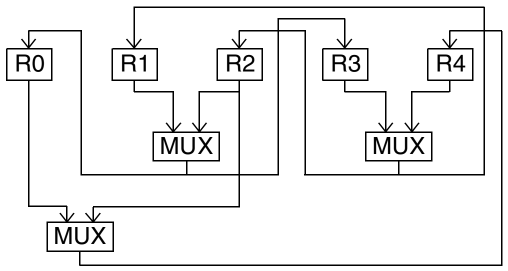


7-27.*

a) Destination <- Source Registers	b) Source Registers -> Destination
R0 <- R1, R2	R0 -> R4
R1 <- R4	R1 -> R0, R3
R2 <- R3, R4	R2 -> R0, R4
R3 <- R1	R3 -> R2
R4 <- R0, R2	R4 -> R1, R2

c) The minimum number of buses needed for operation of the transfers is three since transfer Cb requires three different sources.

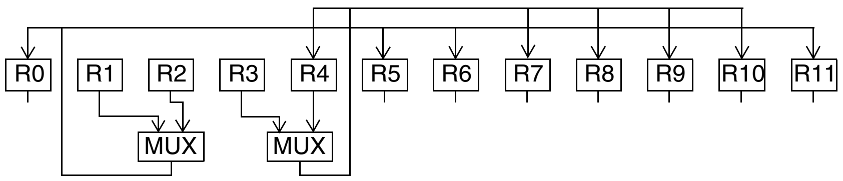
d)



7-28.

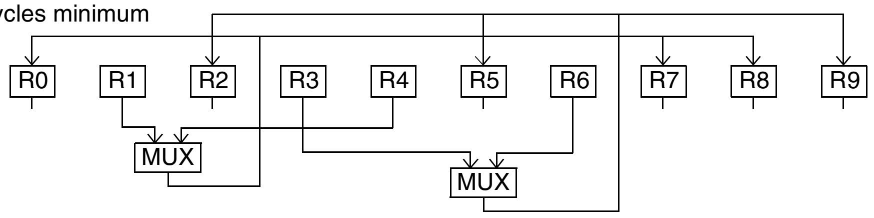
a) Using two clock cycles, the minimum # of buses is 2 .

b)



7-29.

Two clock cycles minimum



7-30.*

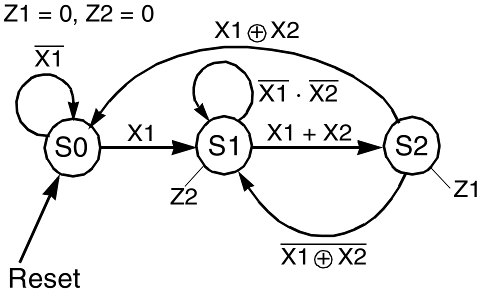
0101, 1010, 0101, 1010, 1101, 0110, 0011, 0001, 1000

7-31.*

Shifts:	0	1	2	3	4
A	0111	0011	0001	1000	1100
B	0101	0010	0001	0000	0000
C	0	1	1	1	0

7-32.*

Default: Z1 = 0, Z2 = 0


$\overline{X1}$
$X1 \oplus X2$
$\overline{X1} \cdot \overline{X2}$
S0
X1
S1
X1 + X2
S2
Z2
Z1
$\overline{X1 \oplus X2}$
Reset

7-33.*

State: STA, STA, STB, STC, STA, STB, STC, STA, STB
Z: 0, 0, 1, 1, 0, 0, 1, 0, -

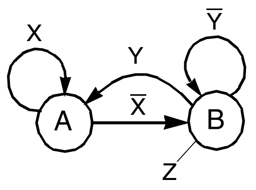
7-34.

State	Input	Next State	Output
STA	$\overline{W}$	STA	*
STA	W	STB	*
STB	$\overline{X}$ Y	STA	*
STB	X	STC	*
STB	$\overline{X}\ \overline{Y}$	STC	Z
STC		STA	Z

*Default: Z = 0

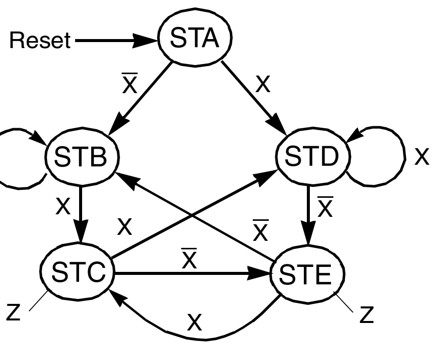
7-35.

Default: Z = 0



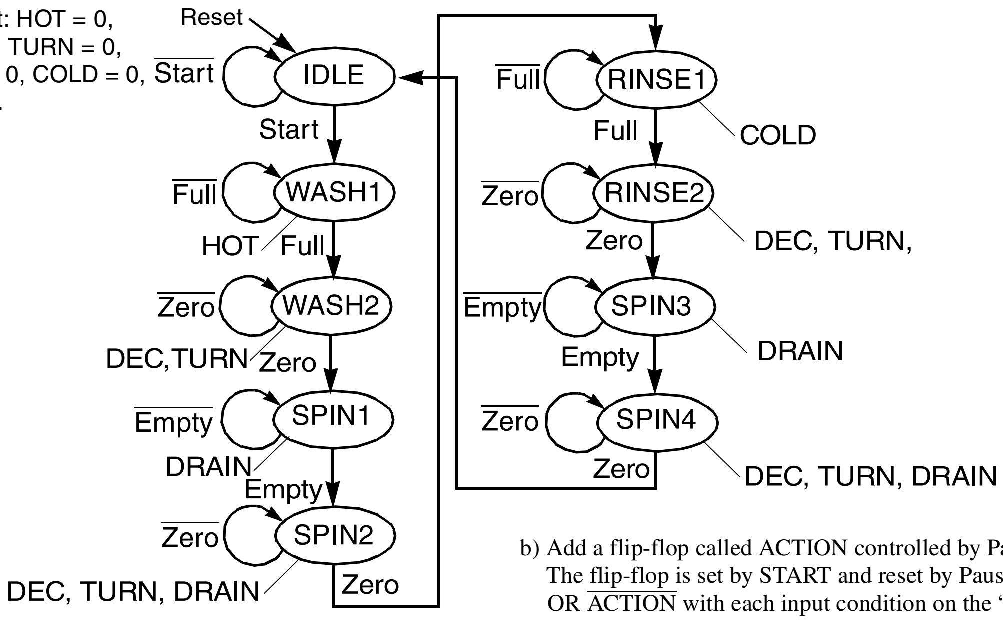
7-36.*

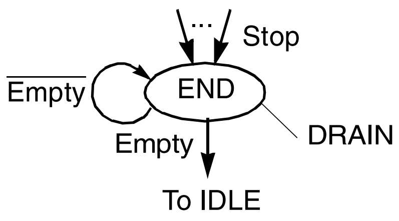
Default: Z = 0



7-37.+

a) Default: HOT = 0, DEC = 0, TURN = 0, DRAIN = 0, COLD = 0, Load = 0.

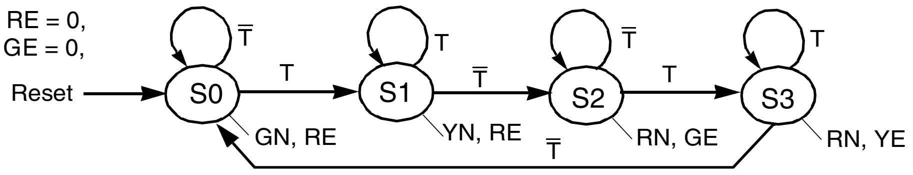




b) Add a flip-flop called ACTION controlled by Pause and Start. The flip-flop is set by START and reset by Pause. OR $\overline{\text{ACTION}}$ with each input condition on the "loop" for each state. AND ACTION with1) each input condition on t the transition to the following state from each state and 2) all the output signals.
Stop causes a transition from each state to a new state END and $\overline{\text{Stop}}$ is ANDed with all of the input conditions on each of the states. The partial diagram for state END appears at the left.

7-38.

Default: GN = 0, RE = 0, YN = 0, RN = 0, GE = 0, YE = 0



7-39.*

Present state				Input	Next state			Output
	A	***B***	***C***		***A***	***B***	***C***	
STA	1	0	0	$\overline{W}$	1	0	0	
	1	0	0	W	0	1	0	
	0	1	0	$\overline{X}Y$	1	0	0	
STB	0	1	0	X	0	0	1	
	0	1	0	$\overline{X}\,\overline{Y}$	0	0	1	Z
STC	0	0	1		1	0	0	Z

$D_A = A\overline{W} + B\overline{X}Y + C$

$D_B = AW$

$D_C = B(X + \overline{Y})$

$Z = B\,\overline{X}\,\overline{Y} + C$

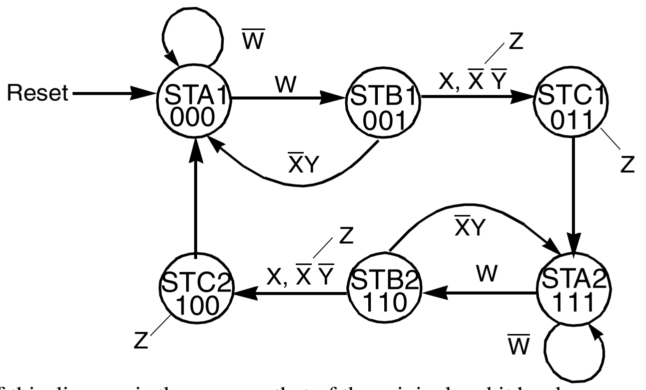
The implementation consists of the logic represented by the above equations and three D flip-flops with Reset connected to S on the first flip-flop and to R on the other two flip-flops.

7-40.

This state diagram has a closed loop of three transitions (STA to STB to STC to STA). In a Gray code, only one bit may change in going from one state to another. Any state machine diagram with a loop of an odd number of transitions is impossible to encode with a Gray code. For example, to go from STA to STB suppose bit B1 of the code changes. Then to go from STB to STC, some other bit, say B2 must change. Since two bits have changed, It is impossible to return to state STA. Thus, the answer to this problem is that this state diagram cannot be implemented with a Gray code.

But suppose that we use two equivaent states to represent each of the original states, STA1, STB1, STC1, STA2, STB2, and STC2. Is it possible to implement the new diagram generated such the it has exactly the same properties as the old diagram. Suppose that the codes are 000, 001, 011, 111, 110, 100, respectively, for the six states. The coded diagram is:

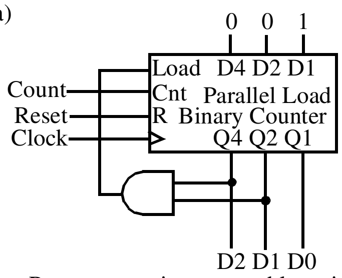
Default: Z = 0



The behavior of this diagram is the same as that of the original and it has been successfully Gray coded by assigning two codes to each state. The implementation is a straightforward design problem with two unused states.

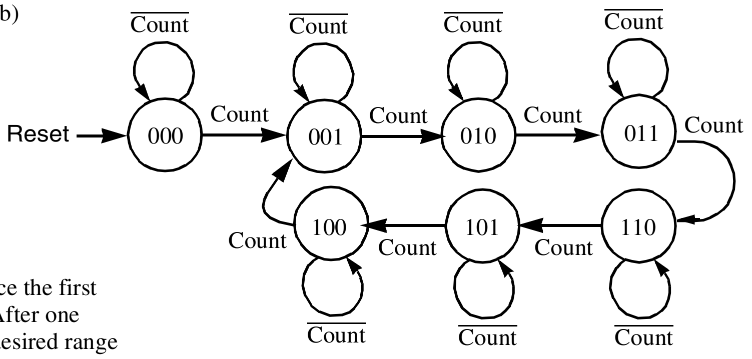
7-41.

a)



Note: Reset to zero is not a problem since the first value will never be used in the design. After one count, the values will have entered the desired range of 1 through 6.

b)



State	Cnt = 0	Cnt = 1
000	000	001
001	001	010
010	010	011
011	011	100
100	100	101
101	101	110
110	110	111
111	ddd	ddd

Applying K-maps to the table entries:

$D_{Q4} = Q4\,\overline{C} + Q4\,\overline{Q2} + Q1\,Q2\,C$

$D_{Q2} = Q2\,\overline{C} + Q1\,Q2\,C + \overline{Q4}\,Q2\,\overline{Q1}$

$D_{Q1} = Q1\,\overline{C} + \overline{Q1}\,C$

Cost comparison:

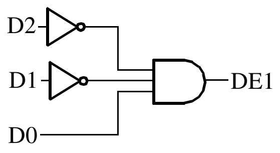
a) 3(14 + 2 + 8 + 6) + 1 + 2 = 93

b) 3(14) + 4 + 6 + 11 + 10 = 73

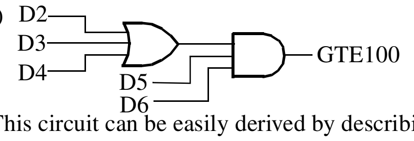
The gate input cost of b) is 78.5 % that of a).

7-42.

a) $DE1 = \overline{D4}\,\overline{D2}\,D1$



b)



This circuit can be easily derived by describing the range of values greater than or equal to 100 in terms of powers of 2.
The smallest value is $2^6 + 2^5 + 2^2$ and the largest value is $2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1$. This range of D can be described by saying that 2^6 **and** 2^5 must be present **and** any of 2^2, 2^3, **or** 2^4 must be present in D.
The resulting equation is D6 D5 (D4 + D3 + D2). An alternative way of finding this is to contract the carry circuit for D + 2's comp of 1100100.

7-43.

Binary	BCD ($C_0 = 0$)	BCD ($C_0 = 1$)
$B_3B_2B_1B_0$	$C_4D_3D_2D_1D_0$	$C_4D_3D_2D_1D_0$
0 0 0 0	0 0 0 0 0	0 0 0 0 1
0 0 0 1	0 0 0 0 1	0 0 0 1 0
0 0 1 0	0 0 0 1 0	0 0 0 1 1
0 0 1 1	0 0 0 1 1	0 0 1 0 0
0 1 0 0	0 0 1 0 0	0 0 1 0 1
0 1 0 1	0 0 1 0 1	0 0 1 1 0
0 1 1 0	0 0 1 1 0	0 0 1 1 1
0 1 1 1	0 0 1 1 1	0 1 0 0 0
1 0 0 0	0 1 0 0 0	0 1 0 0 1
1 0 0 1	0 1 0 0 1	1 0 0 0 0
1 0 1 0	1 0 0 0 0	1 0 0 0 1
1 0 1 1	1 0 0 0 1	1 0 0 1 0
1 1 0 0	1 0 0 1 0	1 0 0 11
1 1 0 1	1 0 0 1 1	1 0 1 0 0
1 1 1 0	1 0 1 0 0	1 0 1 0 1
1 1 1 1	1 0 1 0 1	1 0 1 1 0

For $C_0 = 0$,
C4 = B3 B2 + B3 B1
D3 = B3 $\overline{\text{B2}}$ $\overline{\text{B1}}$
D2 = $\overline{\text{B3}}$ B2 + B2 B1
D1= $\overline{\text{B3}}$ B1 + B3 B2 $\overline{\text{B1}}$
D0 = B0

For $C_0 = 1$,
C4 = B3 B2 + B3 B1 + B3 B0
D3 = B3 $\overline{\text{B2}}$ $\overline{\text{B1}}$ $\overline{\text{B0}}$ + $\overline{\text{B3}}$ B2 B1 B0
D2 = $\overline{\text{B3}}$ B2 $\overline{\text{B1}}$ + B3 B2 B0 + B2 B1 $\overline{\text{B0}}$ + $\overline{\text{B3}}$ $\overline{\text{B2}}$ B1 B0
D1= $\overline{\text{B3}}$ $\overline{\text{B1}}$ B0 + $\overline{\text{B3}}$ B1 $\overline{\text{B0}}$ + B3 B1 B0 + B3 B2 $\overline{\text{B1}}$ $\overline{\text{B0}}$
D0 = $\overline{\text{B0}}$

Combining,
C4 = $\overline{\text{C0}}$ (B3 B2 + B3 B1) + C0 (B3 B2 + B3 B1 + B3 B0)
D3 = $\overline{\text{C0}}$ (B3 $\overline{\text{B2}}$ $\overline{\text{B1}}$) + C0 (B3 $\overline{\text{B2}}$ $\overline{\text{B1}}$ $\overline{\text{B0}}$ + $\overline{\text{B3}}$ B2 B1 B0)
D2 = $\overline{\text{C0}}$ ($\overline{\text{B3}}$ B2 + B2 B1) + C0 ($\overline{\text{B3}}$ B2 $\overline{\text{B1}}$ + B3 B2 B0 + B2 B1 $\overline{\text{B0}}$ + $\overline{\text{B3}}$ $\overline{\text{B2}}$ B1 B0)
D1 = $\overline{\text{C0}}$ ($\overline{\text{B3}}$ B1 + B3 B2 $\overline{\text{B1}}$) + C0 ($\overline{\text{B3}}$ $\overline{\text{B1}}$ B0 + $\overline{\text{B3}}$ B1 $\overline{\text{B0}}$ + B3 B1 B0 + B3 B2 $\overline{\text{B1}}$ $\overline{\text{B0}}$)
D0 = $\overline{\text{C0}}$ B0 + C0 $\overline{\text{B0}}$

Optimizing,
C4 = B3 (B2 + B1 + C0 B0)
D3 =($\overline{\text{C0}}$ + $\overline{\text{B0}}$) B3 $\overline{\text{B2}}$ $\overline{\text{B1}}$ + C0 $\overline{\text{B3}}$ B2 B1 B0
D2 = B2 ($\overline{\text{C0}}$ B1 + $\overline{\text{B3}}$ $\overline{\text{B1}}$ + B1 $\overline{\text{B0}}$ + C0 B3 B0) + C0 $\overline{\text{B3}}$ $\overline{\text{B2}}$ B1 B0
D1 = ($\overline{\text{C0}}$ + $\overline{\text{B0}}$) B3 B2 $\overline{\text{B1}}$ + $\overline{\text{C0}}$ $\overline{\text{B3}}$ B1 + C0 B0 (B3 B1 + $\overline{\text{B3}}$ $\overline{\text{B1}}$) + $\overline{\text{B3}}$ B1 $\overline{\text{B0}}$
D0 = $\overline{\text{C0}}$ B0 + C0 $\overline{\text{B0}}$

7-44.

a) Transition constraint checking for Figure 7-28.

Constraint 1:

INIT: No possible conflicts since a single transition.	
BEGIN: $\overline{\text{ROLL}} \cdot \text{ROLL} = 0$	OK
ROL: $\text{ROLL} \cdot \overline{\text{ROLL}} = 0$	OK
ONE: $\text{DIE1} \cdot \overline{\text{DIE1}} = 0$	OK
ROH: $\text{ROLL} \cdot \overline{\text{ROLL}} \cdot \text{HOLD} = 0$	OK
$\text{ROLL} \cdot \overline{\text{ROLL}} \cdot \overline{\text{HOLD}} = 0$	OK
$\overline{\text{ROLL}} \cdot \text{HOLD} \cdot \overline{\text{ROLL}} \cdot \overline{\text{HOLD}} = 0$	OK
TEST: $\text{WN} \cdot \overline{\text{WN}} = 0$	OK
WIN: $\text{NEW\_GAME} \cdot \overline{\text{NEW\_GAME}} = 0$	OK

Constraint 2:

Condition implicitly = 1	OK
BEGIN: $\overline{\text{ROLL}} + \text{ROLL} = 1$	OK
ROL: $\text{ROLL} + \overline{\text{ROLL}} = 1$	OK
$\text{DIE1} + \overline{\text{DIE1}} = 1$	OK
ROH: $\text{ROLL} + \overline{\text{ROLL}} \cdot \text{HOLD} + \overline{\text{ROLL}} \cdot \overline{\text{HOLD}} = 1$	OK
TEST: $\text{WN} + \overline{\text{WN}} = 1$	OK
WIN: $\text{NEW\_GAME} + \overline{\text{NEW\_GAME}} = 1$	OK

b) Implementation of state machine diagram Figure 7-28 using 1-hot code.

The order from LSB to MSB for the state variables is the same as the order of the states in the diagram from top to bottom. The state variables have the same respective names as the states, e.g., INIT, BEGIN, ...

The flip-flop input equations:

D_{INIT} = INIT(t + 1) = WIN · NEW_GAME

D_{BEGIN} = BEGIN(t + 1) = INIT + ONE · DIE1 + TEST · $\overline{\text{WN}}$ + BEGIN · $\overline{\text{ROLL}}$

D_{ROL} = BEGIN · ROLL + ROH · ROLL + ROL · ROLL

D_{ONE} = ROL · $\overline{\text{ROLL}}$

D_{ROH} = ONE · $\overline{\text{DIE1}}$ + ROH · $\overline{\text{ROLL}}$ · $\overline{\text{HOLD}}$

D_{TEST} = ROH · $\overline{\text{ROLL}}$ · HOLD

D_{WIN} = TEST · WN + WIN · $\overline{\text{NEW\_GAME}}$

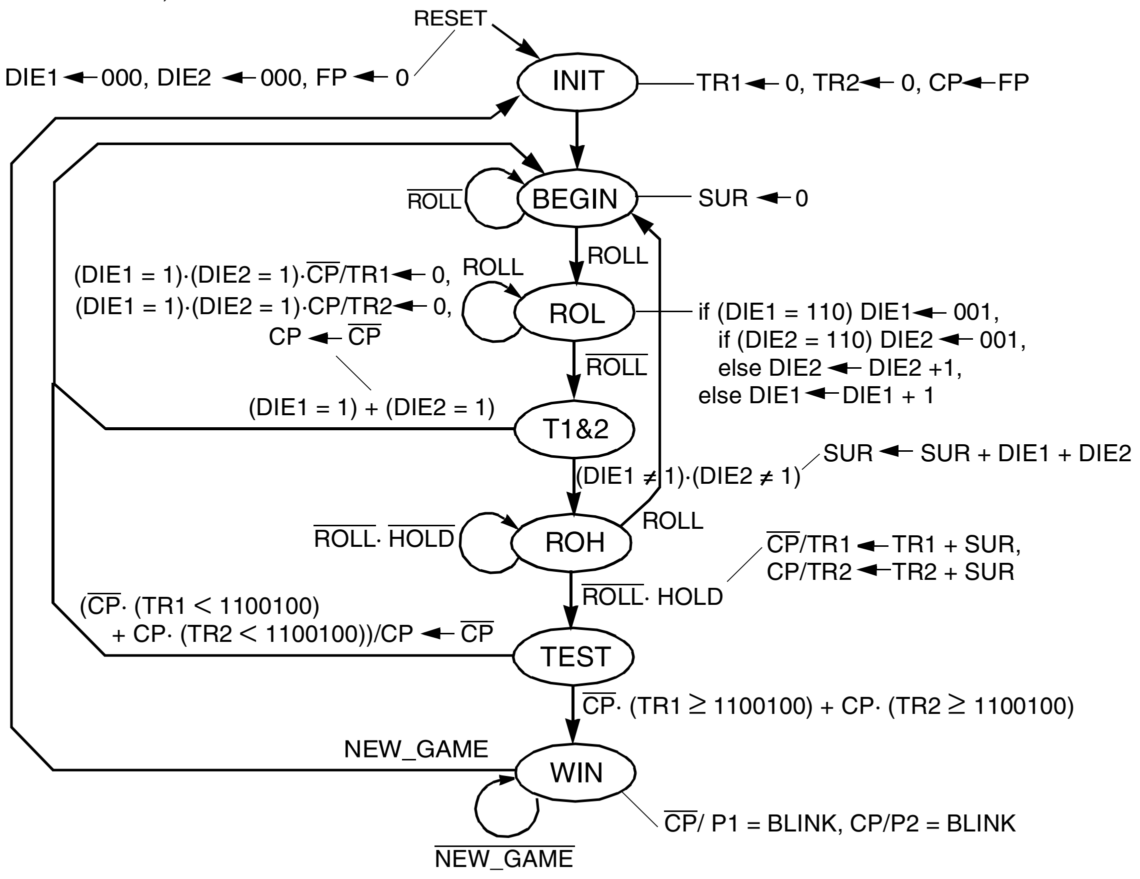
The output equations:

RST1 = INIT, RST2 = INIT, CPFI = INIT, LDCP = INIT + TEST · $\overline{\text{WN}}$ + ONE · DIE1, RSSU = BEGIN, ENDI = ROL, LDSU = ONE, LDT1 = ROH · $\overline{\text{CP}}$ · $\overline{\text{ROLL}}$ · HOLD, LDT2 = ROH · CP · $\overline{\text{ROLL}}$ · HOLD, BP1 = WIN · $\overline{\text{CP}}$, BP2 = WIN · CP

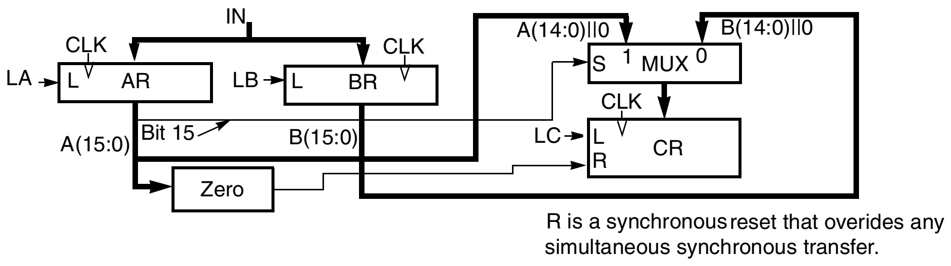
The circuit consists of gates implementing the above equations with logic shared where possible, and seven D flip-flops. The flip-flop for INIT has Reset attached to S and the remaining flip-flops have Reset attached to R.

7-45.+

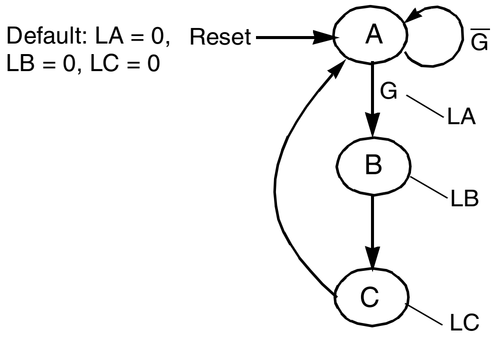
Default: P1 = $\overline{CP}$, P2 = CP

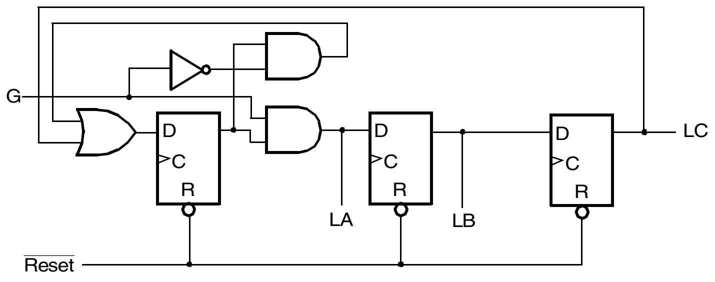

RESET
INIT
DIE1 ← 000, DIE2 ← 000, FP ← 0
TR1 ← 0, TR2 ← 0, CP ← FP
BEGIN
$\overline{ROLL}$
SUR ← 0
ROLL
ROL
(DIE1 = 1)·(DIE2 = 1)·$\overline{CP}$/TR1 ← 0,
(DIE1 = 1)·(DIE2 = 1)·CP/TR2 ← 0,
CP ← $\overline{CP}$
ROLL
if (DIE1 = 110) DIE1 ← 001,
if (DIE2 = 110) DIE2 ← 001,
else DIE2 ← DIE2 +1,
else DIE1 ← DIE1 + 1
$\overline{ROLL}$
T1&2
(DIE1 = 1) + (DIE2 = 1)
(DIE1 ≠ 1)·(DIE2 ≠ 1)
SUR ← SUR + DIE1 + DIE2
ROH
ROLL
$\overline{ROLL}$· $\overline{HOLD}$
$\overline{CP}$/TR1 ← TR1 + SUR,
CP/TR2 ← TR2 + SUR
$\overline{ROLL}$· HOLD
TEST
($\overline{CP}$· (TR1 < 1100100)
+ CP· (TR2 < 1100100))/CP ← $\overline{CP}$
$\overline{CP}$· (TR1 ≥ 1100100) + CP· (TR2 ≥ 1100100)
WIN
NEW_GAME
$\overline{NEW\_GAME}$
$\overline{CP}$/ P1 = BLINK, CP/P2 = BLINK

7-46.*



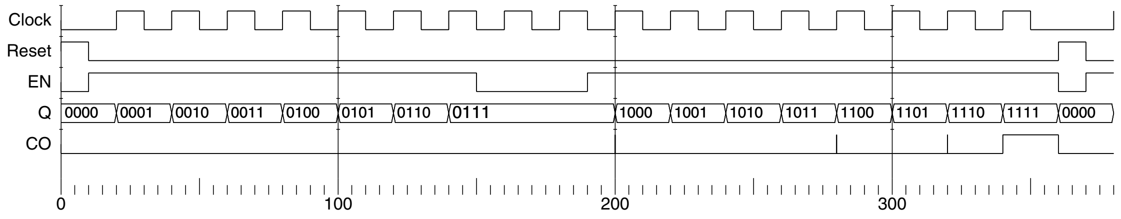
R is a synchronous reset that overides any simultaneous synchronous transfer.





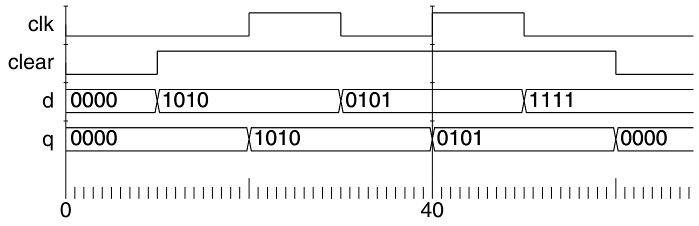
7-47.

```
// 4-bit Binary Counter

// Positive Edge-Triggered D Flip-Flop with Reset:

module dff_v(CLK, RESET, D, Q);
  input CLK, RESET, D;
  output Q;
  reg state;

  assign Q = state;

always @(posedge CLK or posedge RESET)
begin
  if (RESET)
    state <= 0;
  else
    state <= D;
end
endmodule

module Counter_4bit (Clock, Reset, EN, Q, CO) ;

input Clock, Reset, EN ;
output [3:0] Q ;
output CO ;
wire[3:0] Q ;

wire [3:0] C, D_in;

// (continued in next column)
        C[0] = EN,
        C[1] = C[0] & Q[0],
        C[2] = C[1] & Q[1],
        C[3] = C[2] & Q[2],
        CO = C[3] & Q[3];

assign
        D_in[0] = C[0] ^ Q[0],
        D_in[1] = C[1] ^ Q[1],
        D_in[2] = C[2] ^ Q[2],
        D_in[3] = C[3] ^ Q[3];


dff_v
        g1(Clock, Reset, D_in[0], Q[0]),
        g2(Clock, Reset, D_in[1], Q[1]),
        g3(Clock, Reset, D_in[2], Q[2]),
        g4(Clock, Reset, D_in[3], Q[3]);

endmodule
```



7-48.*

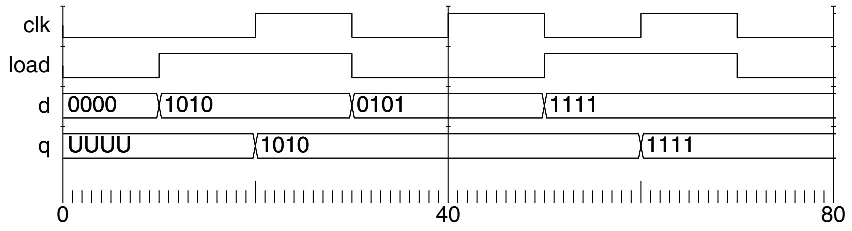
```
library IEEE;
use IEEE.std_logic_1164.all;

entity reg_4_bit is
   port (
      CLEAR, CLK: in STD_LOGIC;
      D: in STD_LOGIC_VECTOR (3 downto 0);
      Q: out STD_LOGIC_VECTOR (3 downto 0)
   );
end reg_4_bit;

architecture reg_4_bit_arch of reg_4_bit is
begin

process (CLK, CLEAR)
begin
  if CLEAR ='0' then                          --asynchronous RESET active Low
     Q <= "0000";
  elsif (CLK'event and CLK='1') then          --CLK rising edge
     Q <= D;
  end if;
end process;

end reg_4_bit_arch;
```



7-49.

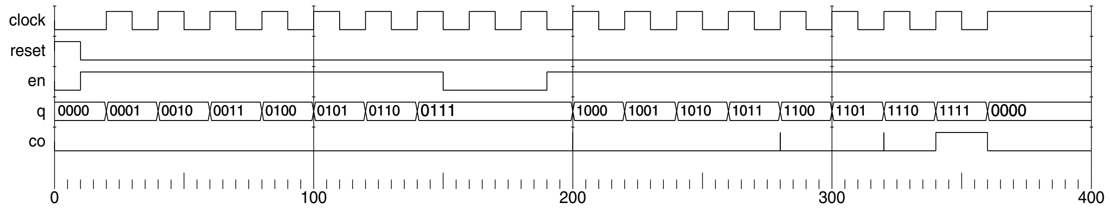
```
library IEEE;
use IEEE.std_logic_1164.all;

entity reg_4_bit is
   port (
      LOAD, CLK: in STD_LOGIC;
      D: in STD_LOGIC_VECTOR (3 downto 0);
      Q: out STD_LOGIC_VECTOR (3 downto 0)
   );
end reg_4_bit;

-- (continued in next column)
```

```
architecture reg_4_bit_load_arch of reg_4_bit is
begin

process (CLK)
begin
     if (CLK'event and CLK='1') then  --CLK rising edge
          if LOAD = '1' then
               Q <= D;
          end if;
     end if;
end process;

endreg_4_bit_load_arch;
```



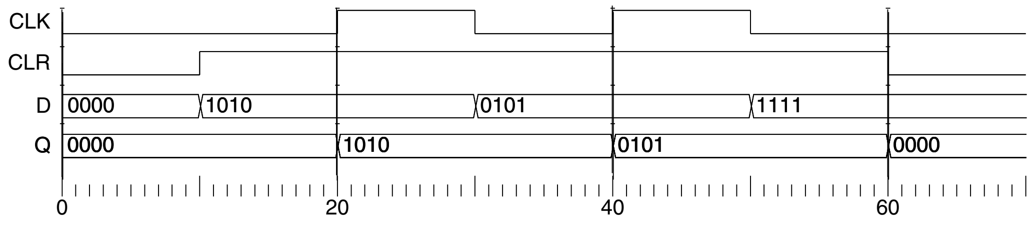
7-50.

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
   port(CLK, RESET, D: in std_logic;
        Q : out std_logic);
end dff;

architecture pet_pr of dff is
-- Implements positive edge-triggered bit state storage
-- with asynchronous reset.
   signal state: std_logic;
begin
   Q <= state;
   process (CLK, RESET)
   begin
     if (RESET = '1') then
       state <= '0';
     else
        if (CLK'event and ClK = '1') then
           state <= D;
        end if;
     end if;
   end process;
end;

library IEEE;
use IEEE.std_logic_1164.all;
entity counter_4_bit is
    port (
        Clock, Reset, EN: in STD_LOGIC;
        Q: out STD_LOGIC_VECTOR (3 downto 0);
        CO: out STD_LOGIC
        );
end counter_4_bit;

architecture counter_4_bit_arch of counter_4_bit is
component dff
    port(CLK, RESET, D: in std_logic;
       Q: out std_logic
    );
end component ;
signal D_in, C,  Q_out: std_logic_vector(3 downto 0);

begin
       C(0) <= EN;
       C(1) <= C(0) and Q_out(0);
       C(2) <= C(1) and Q_out(1);
       C(3) <= C(2) and Q_out(2);
       CO <= C(3) and Q_out(3);

       D_in(0) <= C(0) xor Q_out(0);
       D_in(1) <= C(1) xor Q_out(1);
       D_in(2) <= C(2) xor Q_out(2);
       D_in(3) <= C(3) xor Q_out(3);

       bit0: dff
             port map (Clock, Reset, D_in(0), Q_out(0));
       bit1: dff
             port map (Clock, Reset, D_in(1), Q_out(1));
       bit2: dff
             port map (Clock, Reset, D_in(2), Q_out(2));
       bit3: dff
             port map (Clock, Reset, D_in(3), Q_out(3));

       Q <= Q_out;

end counter_4_bit_arch;
```



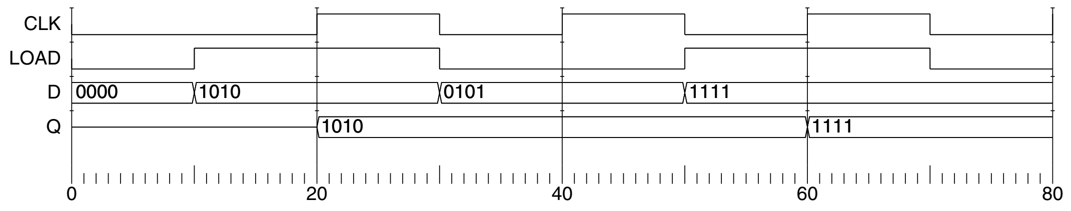
7-51.*

```
module register_4_bit (D, CLK, CLR, Q) ;

input [3:0] D ;
input CLK, CLR ;
output [3:0] Q ;
reg [3:0] Q ;

always @(posedge CLK or negedge CLR)
begin
      if (~CLR)              //asynchronous RESET active low
            Q = 4'b0000;
      else                   //use CLK rising edge
      Q = D;
end
endmodule
```



7-52.

```
module register_4_bit_load (D, CLK, LOAD, Q) ;

input [3:0] D ;
input CLK, LOAD ;
output [3:0] Q ;
reg [3:0] Q ;

always @(posedge CLK)
begin
    if (LOAD)
        Q = D;
end
endmodule
```



7-53.*

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prob_7_53  is
   port (clk, RESET, W, X, Y : in STD_LOGIC;
         Z  : out STD_LOGIC);
end prob_7_53;

architecture process_3 of prob_7_53 is
type state_type is (STA, STB, STC);
signal state, next_state: state_type;
begin

-- Process 1 - state register
state_register: process (clk, RESET)
begin
     if (RESET = '1') then
          state <= STA;
     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 (W, X, Y, state)
begin
     case state is
          when STA =>
-- Continued in next column
               if W = '1' then
                    next_state <= STB;
               else
                    next_state <= STA;
               end if;
          when STB =>
               if X = '0' and Y = '1' then
                    next_state <= STA;
               else
                    next_state <= STC;
               end if;
          when STC =>
                    next_state <= STA;
     end case;
end process;

-- Process 3 - output function
output_func: process (X, Y, state)
begin
     case state is
        when STA =>
          Z <= '0';
          when STB =>
            if X = '0' and Y = '0' then
                 Z <= '1';
               else
                 Z <= '0';
               end if;
          when STC =>
               Z <= '1';
     end case;
 end process;
end process_3;
```

7-54.*

```
// State Diagram in Figure 5-40 using Verilog
module prob_7_54 (clk, RESET, W, X, Y, Z);
input clk, RESET, W, X, Y;
output Z;

reg[1:0] state, next_state;
parameter STA = 2'b00, STB = 2'b01, STC = 2'b10;
reg Z;

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

// Next StateFunction
always@(W or X or Y or state)
begin
    case (state)
    STA: if (W == 1)
            next_state <= STB;
        else
// (continued in the next column)
            next_state <= STA;
    STB: if (X == 0 & Y == 1)
            next_state <= STA;
        else
            next_state <= STC;
    STC:
            next_state <= STA;
    endcase
end

// Output Function
always@(X or Y or state)
begin
    Z <= 0;
    case (state)
        STB:  if (X == 0 & Y == 0)
            Z <= 1;
         else
            Z <= 0;
    STC:
            Z <= 1;
    endcase
end
endmodule
```