Solutions to Problems Marked with a * in
Logic and Computer Design Fundamentals, 4th Edition

Chapter 7

7-2. *

1001 1001	
1100 0011	
1000 0001	AND
1101 1011	OR
0101 1010	XOR

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.

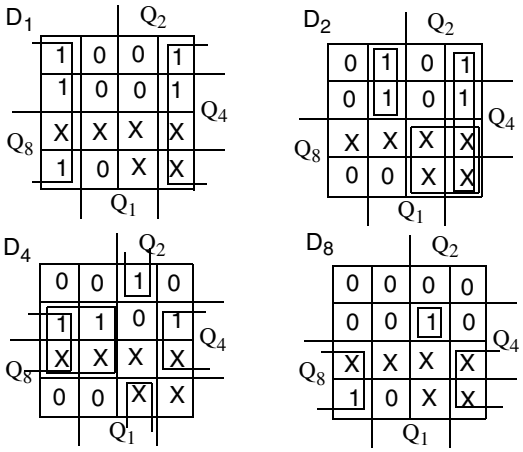
7-6.*

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

b) # States = *n*

7-13.*

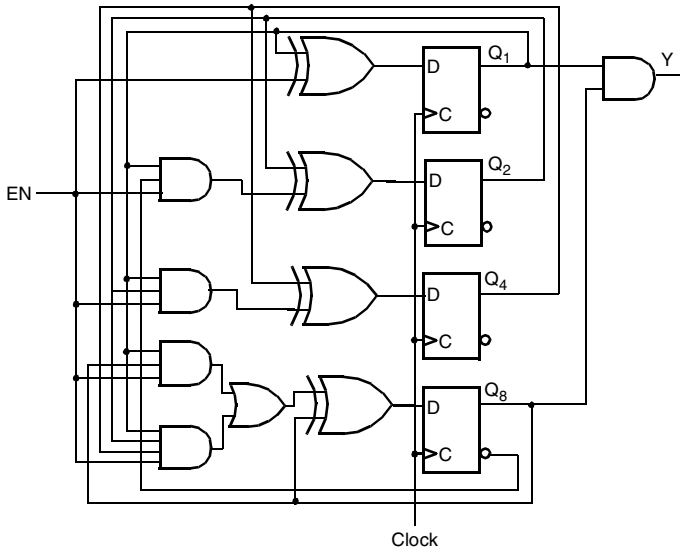
The equations given on page 364-5 can be manipulated into SOP form as follows: $D_1 = \overline{Q}_1$, $D_2 = Q_2 \oplus Q_1\overline{Q}_8 = Q_1\overline{Q}_2\overline{Q}_8 + \overline{Q}_1Q_2 + Q_2Q_8$, $D_4 = Q_4 \oplus Q_1Q_2 = Q_1Q_2\overline{Q}_4 + \overline{Q}_1Q_4 + \overline{Q}_2Q_4$, $D_8 = Q_8 \oplus (Q_1Q_8 + Q_1Q_2Q_4) = \overline{Q}_8(Q_1Q_8+Q_1Q_2Q_4) + Q_8(\overline{Q}_1 + \overline{Q}_8)(\overline{Q}_1 + \overline{Q}_2 + \overline{Q}_4) = Q_1Q_2Q_4\overline{Q}_8 + \overline{Q}_1 Q_8$. 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) $D_B = C$

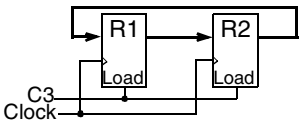
$D_C = \overline{B}\,\overline{C}$

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

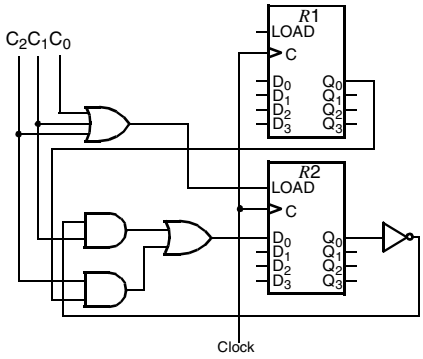
$D_B = \overline{A}\,\overline{B}C + B\overline{C}$

$D_C = \overline{C}$

7-17.*

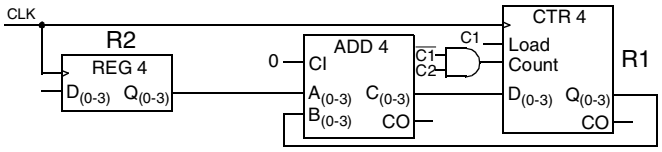


7-19.*

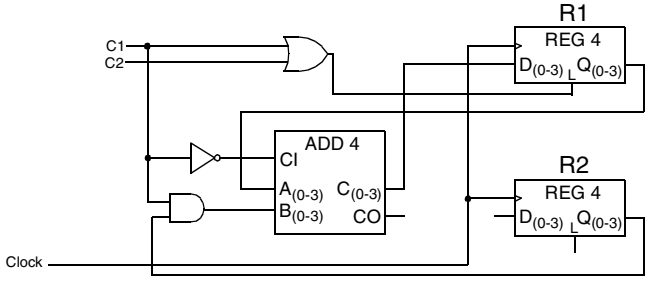


7-24.*

a)



b)



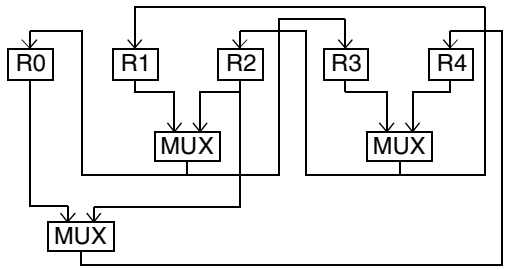
7-27.*

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

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



7-30.*

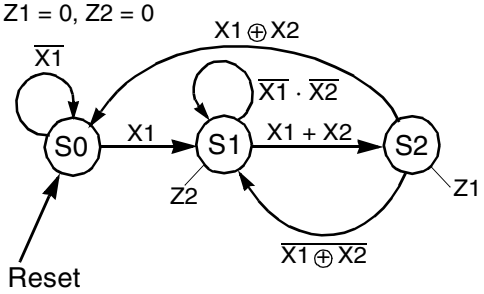
0101, 1010, 0101, 1010, 1101, 0110, 0011, 0001, 1000

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.*

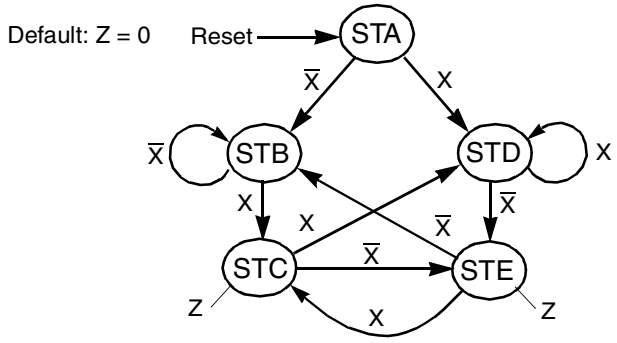
Default: Z1 = 0, Z2 = 0



7-33.*

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

7-36.*



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	
STB	0	1	0	$\overline{X}Y$	1	0	0	
	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

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

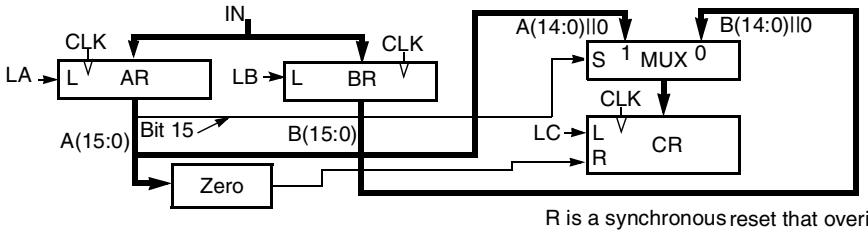
$$D_B = AW$$

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

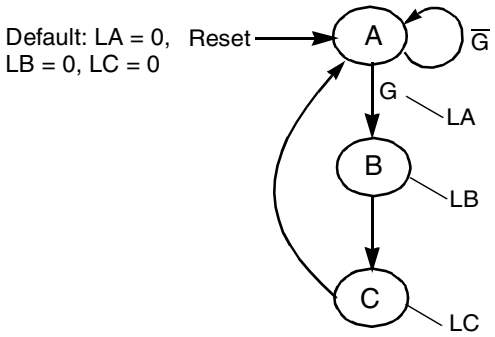
$$Z = B\,\overline{X}\,\overline{Y} + C$$

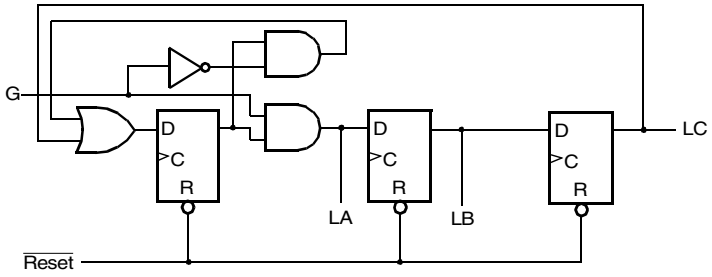
The implementation consists of the logic represented by the above equations and three D flip-flops with Reset connected to S on the first flip-flop and to R on the other two flip-flops.

7-46.*



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





7-48.*

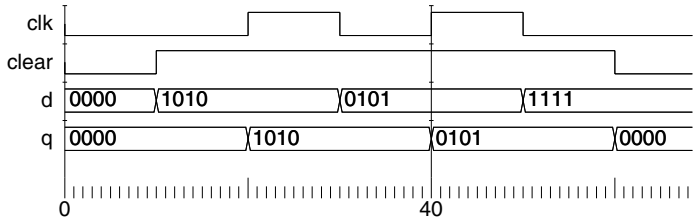
```
library IEEE;
use IEEE.std_logic_1164.all;

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;
```

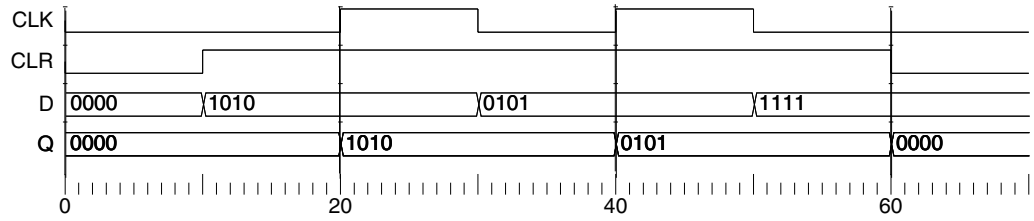


7-51.*

```
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
```



7-53.*

```
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
```