

**Problem Solutions to Problems Marked With a * in
Logic Computer Design Fundamentals, Ed. 2**

CHAPTER 5

© 2000 by Prentice-Hall, Inc.

5-3.

1000, 0100, 1010, 1101 0110, 1011, 1101, 1110

5-6.

Shifts:	0	1	2	3	4
A	0110	1011	0101	0010	1001
B	0011	0001	0000	0000	0000
C	0	0	1	1	0

5-8.

Replace each AND gate in Figure 5-6 with an AND gate with one additional input and connect this input to the following:

$$S_1 + \bar{S}_0$$

This will force the outputs of all the AND gates to zero, and, on the next clock edge, the register will be cleared if S1 is 0 and S0 is logic one.

Also, replace each direct shift input with this equation: $S_1\bar{S}_0$ This will stop the shift operation from interfering with the load parallel data operation.

5-10.

- a) 1000, 0100, 0010, 0001, 1000
- b) # States = n

5-17.

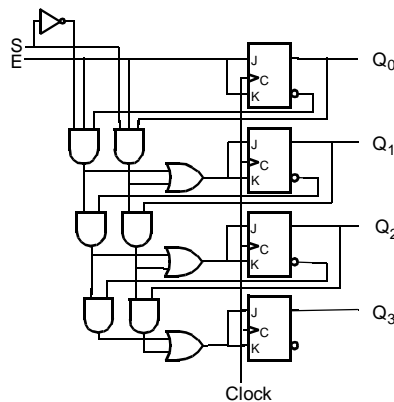
$$Q_0 = \bar{Q}_0 E$$

$$Q_1 = (Q_0 \bar{Q}_1 + \bar{Q}_0 Q_1) E$$

$$Q_2 = (Q_0 Q_1 \bar{Q}_2 + \bar{Q}_1 Q_2 + \bar{Q}_0 Q_2) E$$

$$Q_3 = (\bar{Q}_2 Q_3 + \bar{Q}_1 Q_3 + \bar{Q}_0 Q_3 + Q_0 Q_1 Q_2 \bar{Q}_3) E$$

5-21.



Problem Solutions – Chapter 5

5-24.

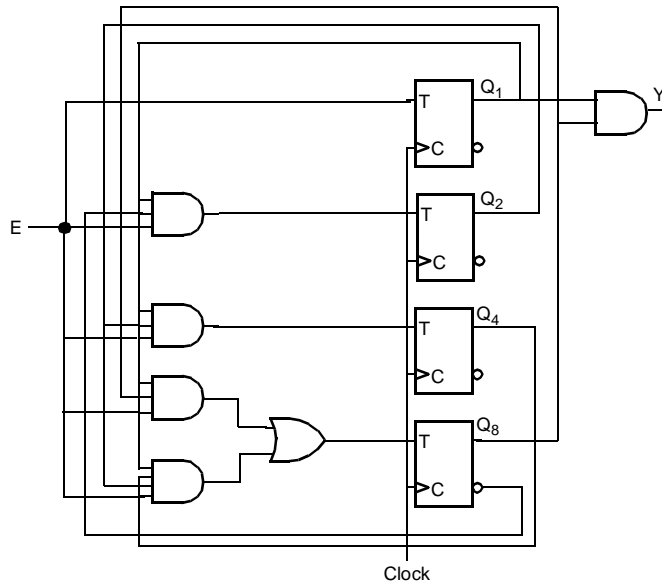
$$T_{Q8} = (Q_1Q_8 + Q_1Q_2Q_4)E$$

$$T_{Q4} = Q_1Q_2E$$

$$T_{Q2} = Q_1\bar{Q}_8E$$

$$T_{Q1} = E$$

$$Y = Q_1Q_8$$



5-26.

Present state			Next state			FF Inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1

a) $J_B = C$
 $K_B = C$
 $J_C = \bar{B}$
 $K_C = 1$

b) $J_A = BC$
 $K_A = C$
 $J_B = \bar{A}C$
 $K_B = C$
 $J_C = 1$
 $K_C = 1$

5-29. (All simulations performed using Xilinx Foundation Series software.)

```

library IEEE;
use IEEE.std_logic_1164.all;

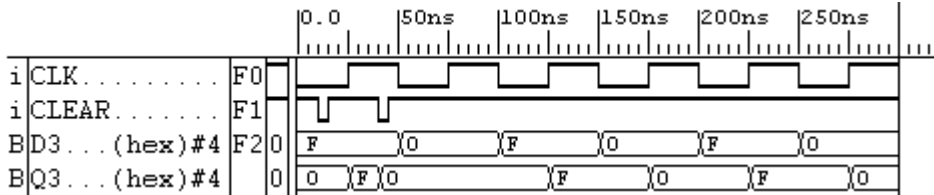
entity reg_4_bit is
  port (
    CLEAR, CLK: in STD_LOGIC;
    D: in STD_LOGIC_VECTOR (3 downto 0);
    Q: out STD_LOGIC_VECTOR (3 downto 0)
  );
end reg_4_bit;

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;

```



5-33.

```

library IEEE;
use IEEE.std_logic_1164.all;

entity ripple_1_bit is
  port (
    RESET, CLK, J, K: in STD_LOGIC;
    Q: out STD_LOGIC
  );
end ripple_1_bit;

architecture ripple_arch of ripple_1_bit is
  signal Q_out: std_logic;
  begin
  process (CLK, RESET)
  begin
    if RESET = '1' then -- asynchronous RESET active
  Low
      Q_out <= '0';
    elsif (CLK'event and CLK='0') then --CLK falling
  edge
      if (J = '1' and K = '1') then
        Q_out <= not Q_out;
      elsif (J = '1' and K = '0') then
        Q_out <= '1';
      elsif (J = '0' and K = '1') then
        Q_out <= '0';
      end if;
    end if;
  end process;
  Q <= Q_out;

end ripple_arch;

-- (Continued in next column)

```

```

library IEEE;
use IEEE.std_logic_1164.all;

entity ripple_4_bit is
  port (
    RESET, CLK: in STD_LOGIC;
    Q: out STD_LOGIC_VECTOR (3 downto 0)
  );
end ripple_4_bit;

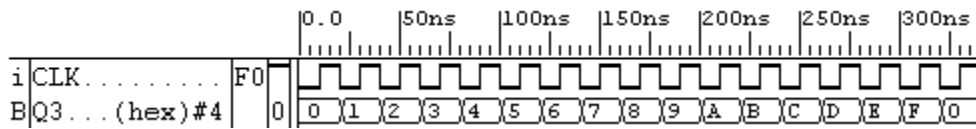
architecture ripple_4_bit_arch of ripple_4_bit is
  component ripple_1_bit
  port (
    RESET, CLK, J, K: in STD_LOGIC;
    Q: out STD_LOGIC
  );
end component ;
  signal logic_1: std_logic;
  signal Q_out: std_logic_vector(3 downto 0);
  begin
    bit0: ripple_1_bit port map(RESET, CLK, logic_1, logic_1,
  Q_out(0));
    bit1: ripple_1_bit port map(RESET, Q_out(0), logic_1, logic_1,
  Q_out(1));
    bit2: ripple_1_bit port map(RESET, Q_out(1), logic_1, logic_1,
  Q_out(2));
    bit3: ripple_1_bit port map(RESET, Q_out(2), logic_1, logic_1,
  Q_out(3));

    logic_1 <= '1';
    Q <= Q_out;

end ripple_4_bit_arch;

```

Problem Solutions – Chapter 5

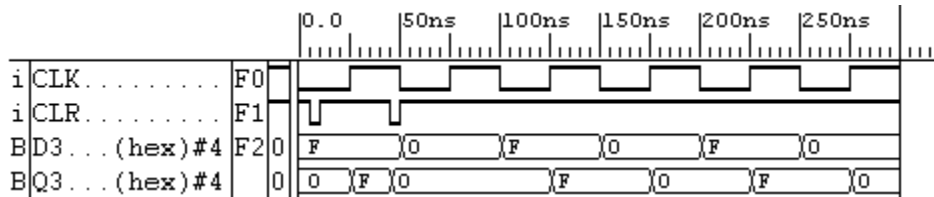


5-35.

```

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
    
```



5-39.

```

module jk_1_bit (J, K, CLK, CLR, Q);
input J, K, CLK, CLR;
output Q;
reg Q;

always @(negedge CLK or posedge CLR)
begin
if (CLR)
    Q <= 1'b0;
else if ((J == 1'b1) && (K == 1'b1))
    Q <= ~Q;
else if (J == 1'b1 && K == 1'b0)
    Q <= 1'b1;
else if (J == 1'b0 && K == 1'b1)
    Q <= 1'b0;
end
endmodule
    
```

```

module reg_4_bit (CLK, CLR, Q);
input CLK, CLR;
output [3:0] Q;
reg [3:0] Q;

jk_1_bit
g1(1'b1, 1'b1, CLK, CLR, Q[0]),
g2(1'b1, 1'b1, Q[0], CLR, Q[1]),
g3(1'b1, 1'b1, Q[1], CLR, Q[2]),
g4(1'b1, 1'b1, Q[2], CLR, Q[3]);

endmodule
    
```

// (continued in next column)

