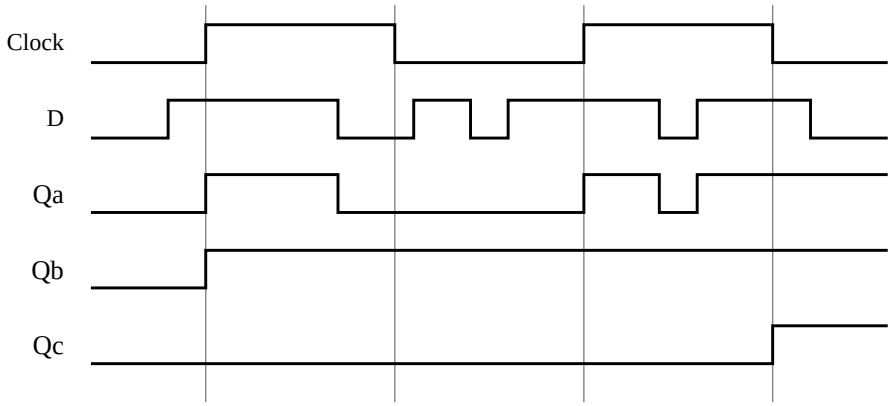
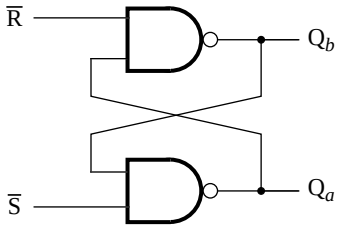
Chapter 7

7.1.



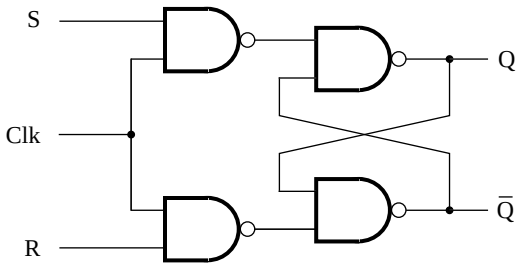
7.2. The circuit in Figure 7.3 can be modified to implement an SR latch by connecting S to the *Data* input and $S + R$ to the *Load* input. Thus the value of S is loaded into the latch whenever either S or R is asserted. Care must be taken to ensure that the *Data* signal remains stable while the *Load* signal is asserted.

7.3.

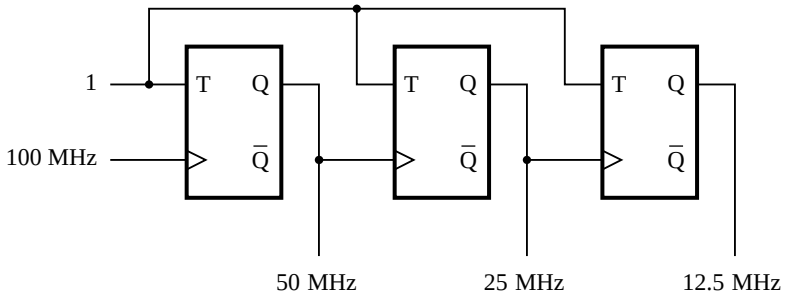


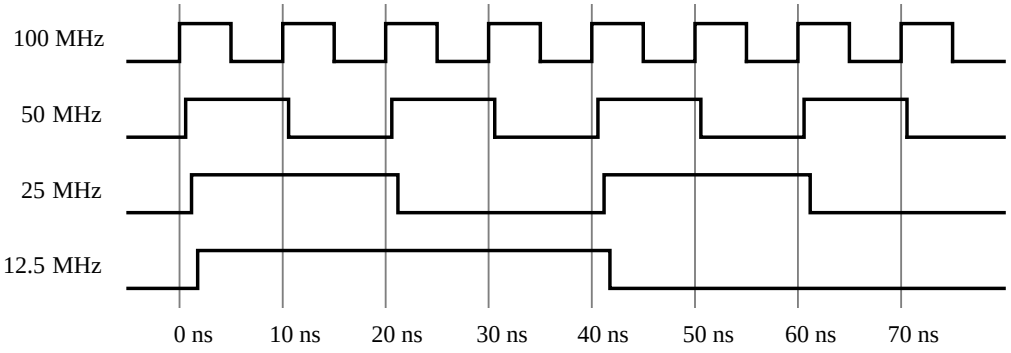
$\overline{S}$	$\overline{R}$	Q_a	Q_b	
1	1	0/1	1/0	(no change)
1	0	0	1	
0	1	1	0	
0	0	1	1	

7.4.

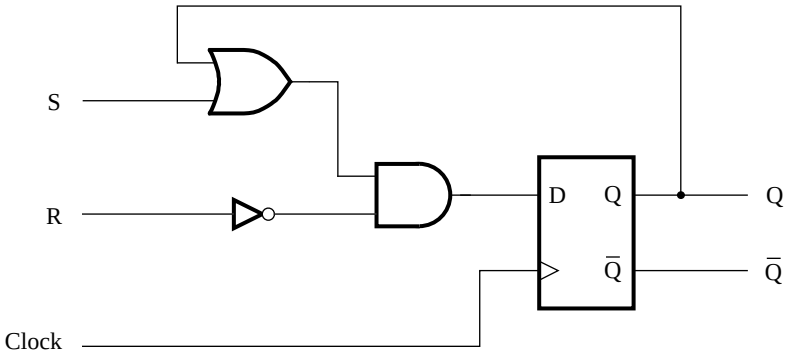


7.5.

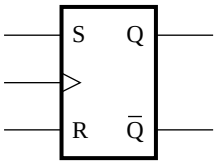




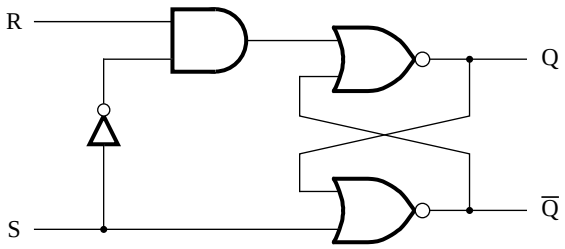
7.6.



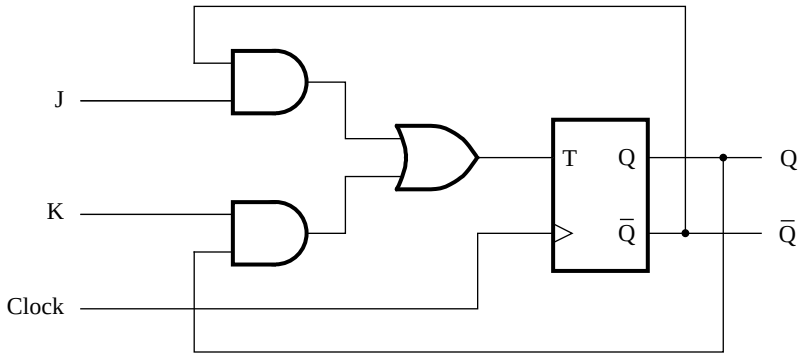
S	R	Q(t + 1)
0	0	Q(t)
0	1	0
1	0	1
1	1	0



7.7.



7.8.



7.9. This circuit acts as a negative-edge-triggered JK flip-flop, in which $J = A, K = B$, *Clock* $= C, \mathrm{Q} = D$, and $\overline{\mathrm{Q}} = E$. This circuit is found in the standard chip called 74LS107A (plus a *Clear* input, which is not shown).

7.10.

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

ENTITY prob7_10 IS
    PORT ( T, Resetn, Clock   : IN     STD_LOGIC ;
           Q                  : OUT  STD_LOGIC ) ;
END prob7_10 ;

ARCHITECTURE Behavior OF prob7_10 IS
    SIGNAL Qint : STD_LOGIC ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Qint <= '0' ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF T = '1' THEN
                Qint <= NOT Qint ;
            ELSE
                Qint <= Qint ;
            END IF ;
        END IF ;
    END PROCESS ;
    Q <= Qint ;
END Behavior ;
```

7.11.

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

ENTITY prob7_11 IS
    PORT ( J, K, Resetn, Clock   : IN     STD_LOGIC ;
           Q                     : OUT  STD_LOGIC ) ;
END prob7_11 ;

ARCHITECTURE Behavior OF prob7_11 IS
    SIGNAL Qint : STD_LOGIC ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Qint <= '0' ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Qint <= ( J AND NOT Qint ) OR ( NOT K AND Qint ) ;
        END IF ;
    END PROCESS ;
    Q <= Qint ;
END Behavior ;
```

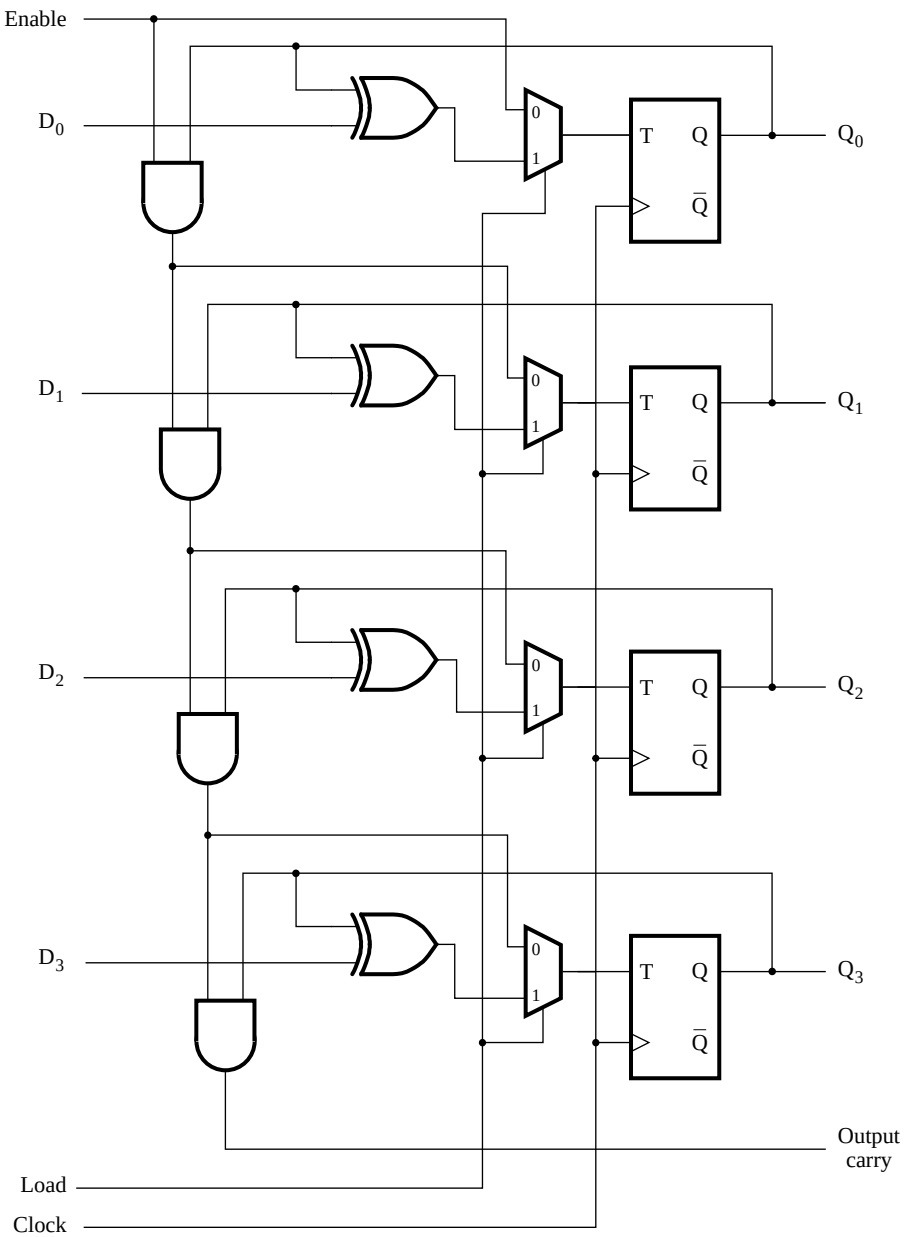
7.13. Let *s* be a binary input that specifies the direction of shifting: if $s = 0$ shifting is done to the right, and if $s = 1$ shifting is done to the left. Also let L be a parallel-load input, and let $R = r_0 r_1 r_2 r_3$ be parallel data. If the inputs to the flip-flops are $d_0 \ldots d_3$ and the outputs are $q_0 \ldots q_3$, then the circuit for the universal shift register can be represented by the logic expressions

$$\begin{aligned} d_0 &= L \cdot r_0 + \overline{L} \cdot (\overline{s} \cdot 0 + s \cdot q_1) \\ d_1 &= L \cdot r_1 + \overline{L} \cdot (\overline{s} \cdot q_0 + s \cdot q_2) \\ d_2 &= L \cdot r_2 + \overline{L} \cdot (\overline{s} \cdot q_1 + s \cdot q_3) \\ d_3 &= L \cdot r_3 + \overline{L} \cdot (\overline{s} \cdot q_2 + s \cdot 0) \end{aligned}$$

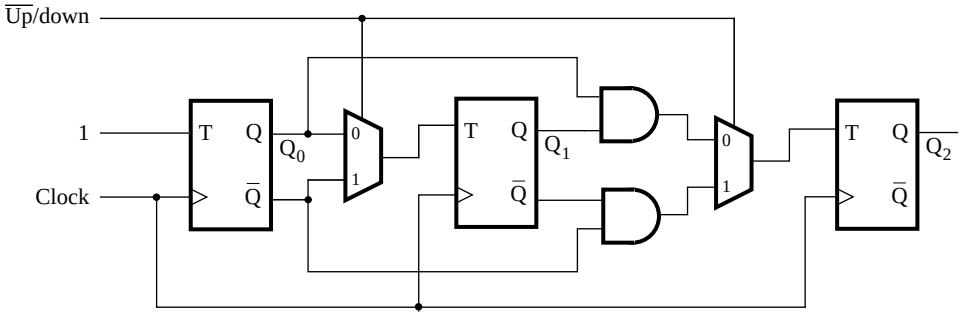
7.14.

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

ENTITY prob7_14 IS
    GENERIC ( N : INTEGER := 4 ) ;
    PORT ( R          : IN     STD_LOGIC_VECTOR (0 TO N−1) ;
           S          : IN     STD_LOGIC_VECTOR (1 DOWNTO 0) ;
           L, Clock   : IN     STD_LOGIC ;
           Q          : BUFFER STD_LOGIC_VECTOR (0 TO N−1) ) ;
END prob7_14 ;

ARCHITECTURE Behavior OF prob7_14 IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF L = '1' THEN
            Q <= R ;
        ELSE
            CASE S IS
                WHEN '0' => Q <= '0' & Q(0 TO N−2) ;          - - right shift
                WHEN OTHERS => Q <= Q(1 TO N−1) & '0' ; - - left shift
            END CASE ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

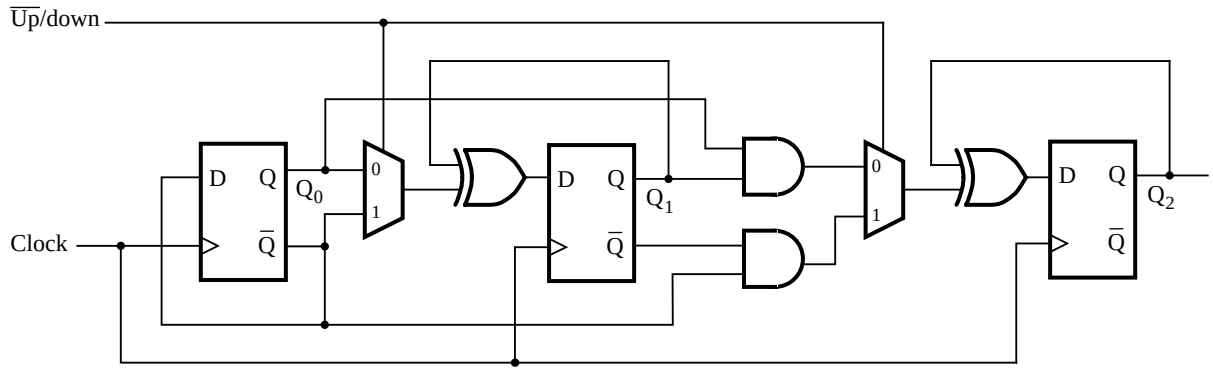
7.15.


Enable
D_0
D_1
D_2
D_3
T
Q
$\overline{Q}$
0
1
Q_0
Q_1
Q_2
Q_3
Output carry
Load
Clock

7.16.


$\overline{Up}$/down
1
Clock
T
Q
$\overline{Q}$
0
1
Q_0
Q_1
Q_2

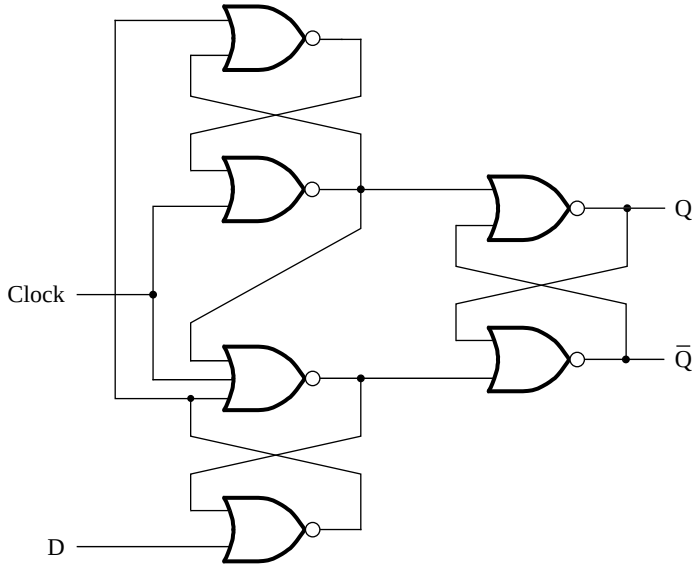
7.17.



7.18. The counting sequence is $000, 001, 010, 111$.

7.19. The circuit in Figure P7.4 is a master-slave JK flip-flop. It suffers from a problem sometimes called *ones-catching*. Consider the situation where the Q output is low, *Clock* = 0, and $J = K = 0$. Now let *Clock* remain stable at 0 while *J* change from 0 to 1 and then back to 0. The master stage is now set to 1 and this value will be incorrectly transferred into the slave stage when the clock changes to 1.

7.20. Repeated application of DeMorgan's theorem can be used to change the positive-edge triggered D flip-flop in Figure 7.11 into the negative-edge D triggered flip-flop:



7.21.

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

ENTITY prob7_21 IS
    PORT ( R                    : IN     STD_LOGIC_VECTOR(23 DOWNTO 0) ;
           Clock, Resetn, L, U  : IN     STD_LOGIC ;
           Q                    : BUFFER STD_LOGIC_VECTOR(23 DOWNTO 0) ) ;
END prob7_21 ;

ARCHITECTURE Behavior OF prob7_21 IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0') ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF L = '1' THEN
                Q <= R ;
            ELSIF U = '1' THEN
                Q <= Q+1 ;
            ELSE
                Q <= Q-1 ;
            END IF ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.22.

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

ENTITY prob7_22 IS
    GENERIC ( N : INTEGER := 4 ) ;
    PORT ( Clock, Resetn, E  : IN     STD_LOGIC ;
           Q                 : OUT    STD_LOGIC_VECTOR ( N-1 DOWNTO 0) ) ;
END prob7_22 ;

ARCHITECTURE Behavior OF prob7_22 IS
    SIGNAL Count : STD_LOGIC_VECTOR ( N-1 DOWNTO 0 ) ;
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Count <= (OTHERS => '0') ;

    . . . con't
```

```
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF E = '1' THEN
                Count <= Count + 1 ;
            ELSE
                Count <= Count ;
            END IF ;
        END IF ;
    END PROCESS ;
    Q <= Count ;
END Behavior ;
```

7.23.

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

ENTITY prob7_23 IS
    PORT ( R                : IN     INTEGER RANGE 0 TO 11 ;
           Clock, Resetn, L : IN     STD_LOGIC ;
           Q                : BUFFER INTEGER RANGE 0 TO 11 ) ;
END prob7_23 ;

ARCHITECTURE Behavior OF prob7_23 IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= 0 ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF L = '1' THEN
                Q <= R ;
            ELSE
                IF Q = 11 THEN
                    Q <= 0 ;
                ELSE
                    Q <= Q + 1 ;
                END IF ;
            END IF ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.24. The longest delay in the circuit is the from the output of FF_0 to the input of FF_3. This delay totals 5 ns. Thus the minimum period for which the circuit will operate reliably is

$$T_{min} = 5\,\text{ns} + t_{su} = 8\,\text{ns}$$

The maximum frequency is

$$F_{max} = 1/T_{min} = 125\,\text{MHz}$$

7.25.

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

ENTITY prob7_25 IS
    PORT ( Clock, Clear   : IN          STD_LOGIC ;
           BCD0, BCD1 : BUFFER  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END prob7_25 ;

ARCHITECTURE Structure OF prob7_25 IS
    COMPONENT fig7_25
        PORT ( D                    : IN          STD_LOGIC_VECTOR(3 DOWNTO 0) ;
               Clock, Enable, Load  : IN          STD_LOGIC ;
               Q                    : BUFFER  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
    END COMPONENT ;
    SIGNAL Load0, Load1 : STD_LOGIC ;
    SIGNAL Enab0, Enab1 : STD_LOGIC ;
    SIGNAL Zero         : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
BEGIN
    Zero  <= "0000" ;
    Enab0 <= '1' ;
    Enab1 <= BCD0(0) AND BCD0(3) ;

    Load0 <= Enab1 OR Clear ;
    Load1 <= (BCD1(0) AND BCD1(3)) OR Clear ;

    cnt0: fig7_25 PORT MAP ( Clock => Clock, Load => Load0, Enable => Enab0,
                             D => Zero, Q => BCD0 ) ;
    cnt1: fig7_25 PORT MAP ( Clock => Clock, Load => Load1, Enable => Enab1,
                             D => Zero, Q => BCD1 ) ;
END Structure ;
```

7.26.

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

ENTITY prob7_26 IS
    PORT ( Clock, Resetn  : IN          STD_LOGIC ;
           Q              : BUFFER  STD_LOGIC_VECTOR(0 TO 7) ) ;
END prob7_26 ;

ARCHITECTURE Behavior OF prob7_26 IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= "00000000" ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= (NOT Q(7)) & Q(0 TO 6) ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.27.

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

ENTITY prob7_27 IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT ( Clock, Start  : IN      STD_LOGIC ;
           Q             : BUFFER  STD_LOGIC_VECTOR(0 TO N−1) ) ;
END prob7_27 ;

ARCHITECTURE Behavior OF prob7_27 IS
BEGIN
    PROCESS ( Clock, Start )
    BEGIN
        IF Start = '1' THEN
            Q <= (OTHERS => '0') ;
            Q(0) <= '1' ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            GenBits: FOR i IN 1 TO N−1 LOOP
                Q(i) <= Q(i−1) ;
            END LOOP ;
            Q(0) <= Q(N−1) ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.28.

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

ENTITY prob7_28 IS
    PORT ( Clock, Reset  : IN      STD_LOGIC ;
           Data          : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Q             : BUFFER  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END prob7_28;

ARCHITECTURE Behavior OF prob7_28 IS
BEGIN
    PROCESS ( Clock, Reset )
    BEGIN
        IF Reset = '1' THEN
            Q <= "0000" ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= Q + Data ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

7.32.

	T_1	T_2	T_3
(Swap): I_4	$R_{out} = X, T_{in}$	$R_{out} = Y, R_{in} = X$	$T_{out}, R_{in} = Y,$ *Done*

Since the processor now has five operations a 3-to-8 decoder is needed to decode the signals f_2, f_1, f_0. The SWAP operation is represented by the code

$$I_4 = f_2\overline{f}_1\overline{f}_0$$

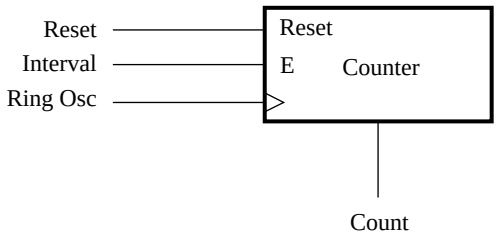
New expressions are needed for R_{in} and R_{out} to accommodate the SWAP operation:

$$\begin{aligned} Rk_{in} &= (I_0 + I_1) \cdot T_1 \cdot X_k + (I_2 + I_3) \cdot T_3 \cdot X_k + I_4 \cdot T_2 \cdot X_k + I_4 \cdot T_3 \cdot Y_k \\ Rk_{out} &= I_1 \cdot T_1 \cdot Y_k + (I_2 + I_3) \cdot (T_1 X_k + T_2 Y_k) + I_4 \cdot T_1 X_k + I_4 \cdot T_2 Y_k \end{aligned}$$

The control signals for the temporary register, T, are

$$\begin{aligned} T_{in} &= T_1 I_4 \\ T_{out} &= T_3 I_4 \end{aligned}$$

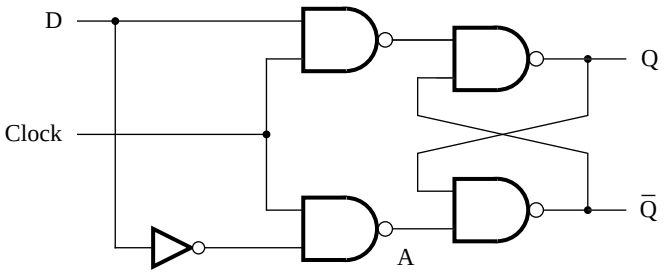
7.33. (a) $\text{Period} = 2 \times n \times t_p$
(b)

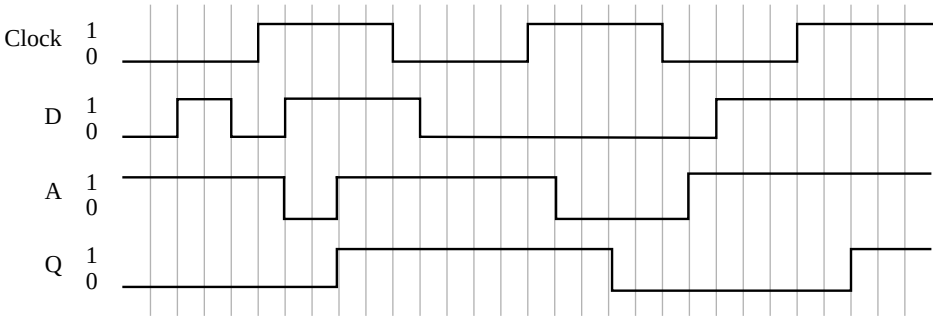


The counter tallies the number of pulses in the 100 ns time period. Thus

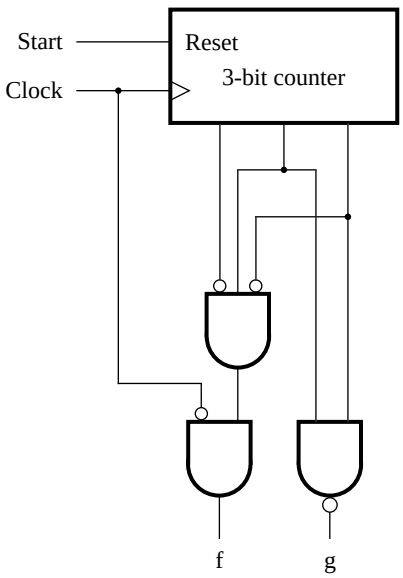
$$t_p = \frac{100\,\text{ns}}{2 \times Count \times n}$$

7.34.





7.35.



7.36. The four-digit BCD counter may be specified as follows:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY prob7_36 IS
    PORT ( Clock, Clear, E : IN  STD_LOGIC ;
           BCD3, BCD2, BCD1, BCD0 : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0)) ;
END prob7_36;
ARCHITECTURE Behavior OF prob7_36 IS
    SIGNAL Carry : STD_LOGIC_VECTOR(4 DOWNTO 1) ;
    COMPONENT digit
        PORT ( Clock, Clear, E : IN STD_LOGIC ;
               BCD_digit : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) ;
               nine : OUT STD_LOGIC ) ;
    END COMPONENT ;
BEGIN
stage0: digit PORT MAP(Clock, Clear, E, BCD0, Carry(1)) ;
stage1: digit PORT MAP(Clock, Clear, (Carry(1) AND E), BCD1, Carry(2)) ;
stage2: digit PORT MAP(Clock, Clear, (Carry(2) AND Carry(1) AND E), BCD2, Carry(3)) ;
stage3: digit PORT MAP(Clock, Clear, (Carry(3) AND Carry(2) AND Carry(1) AND E), BCD3, Carry(4)) ;
END Behavior ;

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY digit IS
    PORT ( Clock, Clear, E : IN  STD_LOGIC ;
           BCD_digit : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           nine : OUT STD_LOGIC ) ;
END digit ;
ARCHITECTURE Spec OF digit IS
BEGIN
    PROCESS (Clock)
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN
            IF Clear = '1' THEN
            BCD_digit <= "0000" ;
            ELSIF E = '1' THEN
                IF BCD_digit = "1001" THEN
                    BCD_digit <= "0000" ;
                ELSE
                    BCD_digit <= BCD_digit + '1' ;
                END IF ;
            END IF ;
        END IF ;
    END PROCESS ;
    PROCESS (BCD_digit)
    BEGIN
        IF BCD_digit = "1001" THEN
            nine <= '1' ;
        ELSE
            nine <= '0' ;
        END IF ;
    END PROCESS ;
END Spec ;
```

7.37. If the 2-to-1 multiplexer is implemented as shown in Figure 6.1, then

$$\begin{aligned} t_{mux} &= t_{NOT} + t_{AND} + t_{OR} \\ &= 1.1 + 1.2 + 1.2 \\ &= 3.5 \text{ ns} \end{aligned}$$

For the circuit in Figure 7.25 have

$$\begin{aligned} T_{min} &= t_{c\text{Q}} + 3(t_{AND}) + t_{XOR} + t_{MUX} + t_{su} \\ &= 1.0 + 3(1.2) + 1.2 + 3.5 + 0.6 \\ &= 9.9 \text{ ns} \end{aligned}$$

Therefore, F_{max} = 1/9.9 ns = 101.01 MHx.

7.38. Assume that there are 4 registers in Figure 7.60. Then, 5-to-1 multiplexers are needed. If the multiplexers are implemented using an AND-OR circuit similar to Figure 6.2*c*, then each AND gate has 4 inputs and the OR gate has 5 inputs. Hence

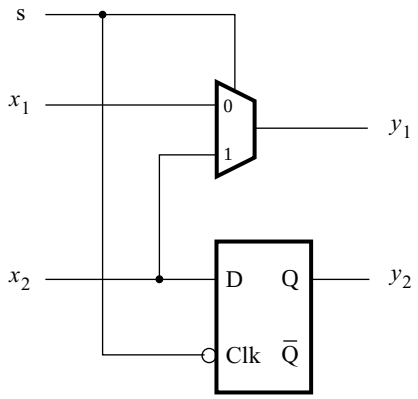
$$\begin{aligned} t_{MUX} &= t_{NOT} + t_{AND} + t_{OR} \\ &= 1.1 + 1.4 + 1.5 \\ &= 4.0 \text{ ns} \end{aligned}$$

Then

$$\begin{aligned} T_{min} &= t_{register} + t_{MUX} \\ &= t_{c\text{Q}} + t_{su} + t_{MUX} \\ &= 1.0 + 0.6 + 4.0 \\ &= 5.6 \text{ ns} \end{aligned}$$

Therefore, F_{max} = 1/5.6 ns = 178.57 MHz.

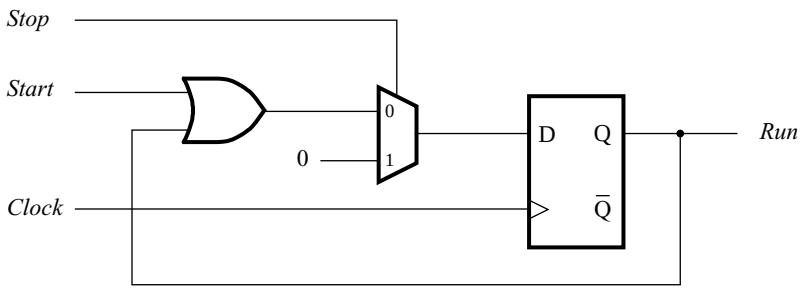
7.39. (a) The following circuit may be synthesized from the VHDL code in Figure P7.9:



(b) To specify a crossbar switch include in the ELSIF clause the statement

```
y2 <= x1;
```

7.40. (a) The following circuit can be used:



(b) The circuit can be specified with the code:

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

ENTITY prob7_40 IS
    PORT ( Clock, Start, Stop : IN  STD_LOGIC ;
           Run :  BUFFER  STD_LOGIC ) ;
END prob7_40;

ARCHITECTURE Behavior OF prob7_40 IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF Stop = '1' THEN
            Run <= '0' ;
        ELSE
            Run <= Start OR Run ;
        END IF ;
    END PROCESS ;
END Behavior ;
```