Chapter 7

7.1. 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.

7.2. 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. 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.

7.4. 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.
7.5.

![Circuit Diagram]

7.6.

![Circuit Diagram]

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>$Q(t+1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$Q(t)$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
7.7.

![Diagram of R-S flip-flop circuit]

7.8.

![Diagram of JK flip-flop circuit]

7.9. As the circuit in Figure P7.2 is drawn, it is not a useful flip-flop circuit, because setting \( C = 0 \) results in both of the circuit outputs being set to 0. Consider the slightly modified circuit shown below:

![Modified JK flip-flop circuit diagram]

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

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
      IF T = '1' THEN
        Qint <= NOT Qint;
      ELSE
        Qint <= Qint;
      END IF;
    END IF;
  END PROCESS;
  Q <= Qint;
END Behavior;
7.13. Let $S = s_1 s_0$ be a binary number that specifies the number of bit-positions to shift by. Also let $L$ be a parallel-load input, and let $R = r_3 r_2 r_1 r_0$ 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 barrel-shifter can be represented by the logic expressions

$$
D_3 = L \cdot R_3 + \overline{L} \cdot (\overline{s_1 s_0} q_3)
$$
$$
D_2 = L \cdot R_2 + \overline{L} \cdot (\overline{s_1} s_0 q_2 + s_1 s_0 q_3)
$$
$$
D_1 = L \cdot R_1 + \overline{L} \cdot (\overline{s_1} s_0 q_1 + \overline{s_1} s_0 q_2 + s_1 s_0 q_3)
$$
$$
D_0 = L \cdot R_0 + \overline{L} \cdot (\overline{s_1} s_0 q_0 + \overline{s_1} s_0 q_1 + \overline{s_1} s_0 q_2 + \overline{s_1} s_0 q_3)
$$

7.14. LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY prob7_14 IS
  PORT ( R : IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
         Shift : IN STD_LOGIC_VECTOR (1 DOWNTO 0) ;
         L, Clock : IN STD_LOGIC ;
         Q : BUFFER STD_LOGIC_VECTOR (3 DOWNTO 0) ) ;
END prob7_14 ;

ARCHITECTURE Behavior OF prob7_14 IS
BEGIN
  PROCESS ( Clock )
  BEGIN
    WAIT UNTIL Clock'EVENT AND Clock = '1' ;
    IF L = '1' THEN
      Q <= R ;
    ELSE
      CASE Shift IS
        WHEN "10" => Q <= "00" & Q(3 DOWNTO 2) ;
        WHEN "01" => Q <= "0" & Q(3 DOWNTO 1) ;
        WHEN OTHERS => Q <= Q ;
      END CASE ;
    END IF ;
  END PROCESS ;
END Behavior ;
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 onescatching. 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 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

$$F_{max} = \frac{1}{T_{min}} = 125\, \text{MHz}$$
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 ®g7_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: ®g7_25 PORT MAP ( Clock => Clock, Load => Load0, Enable => Enab0,
                           D => Zero, Q => BCD0 ) ;
  cnt1: ®g7_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 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 ;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY prob7_29 IS
    PORT ( Clock, Reset : IN STD_LOGIC;
    Q : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END prob7_29;

ARCHITECTURE Structural OF prob7_29 IS
    BEGIN
        cnt: lpm_counter
        GENERIC MAP ( lpm_width => 32)
        PORT MAP ( clock => Clock, aclr => Reset, q => Q );
    END Structural;

7.33.

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rout</td>
<td>X, T1</td>
<td>R1n = Y, R1n = X</td>
<td>T1out, R1n = Y, Done</td>
</tr>
</tbody>
</table>

(Swap): \( I_4 \begin{align*}
R_{out} & = X, T_{1n} \\
R_{out} & = Y, R_{1n} = X \\
T_{out}, R_{1n} & = Y, Done
\end{align*} \)

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

\[ I_4 = f_2 \overline{f_1} f_0 \]

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

\[
\begin{align*}
R_{k_{in}} &= (I_0 + I_1) \cdot T_1 \cdot X_k + (I_2 + I_3) \cdot T_3 \cdot X_k + I_4 \cdot T_2 \cdot X_k + I_4 \cdot T_3 \cdot Y_k \\
R_{k_{out}} &= I_1 \cdot T_1 \cdot Y_k + (I_2 + I_3) \cdot (T_1 X_k + T_2 Y_k) + I_4 \cdot T_1 X_k + I_4 \cdot T_2 Y_k
\end{align*}
\]

The control signals for the temporary register, \( T \), are

\[
\begin{align*}
T_{1n} &= T_1 I_4 \\
T_{out} &= T_3 I_4
\end{align*}
\]

7.34. (a) Period = \( 2 \times n \times t_p \)
(b)

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

\[
t_p = \frac{100 \text{ ns}}{2 \times \text{Count} \times n}
\]
7.35.

![Logic diagram](image)

Clock waveforms:
- Clock
- D
- A
- Q

7.36.

![3-bit counter diagram](image)