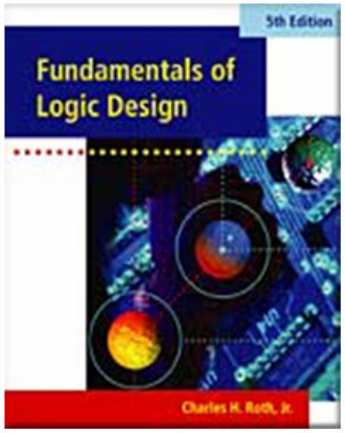
FIGURES FOR

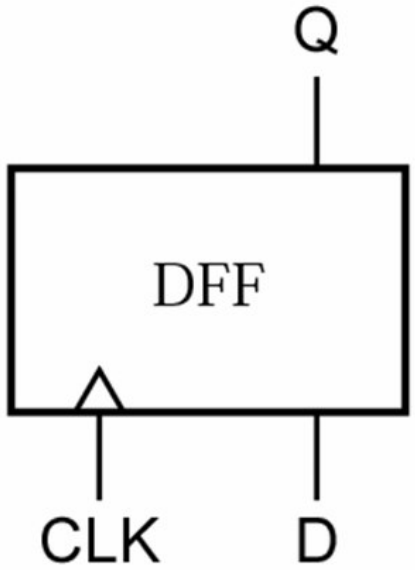
CHAPTER 17

VHDL FOR SEQUENTIAL LOGIC



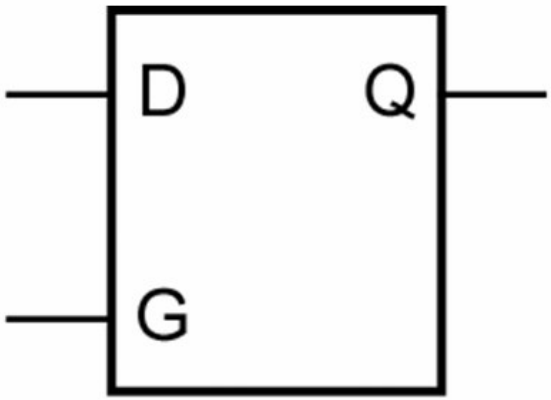
This chapter in the book includes:

Click the mouse to move to the next page.
Use the ESC key to exit this chapter.



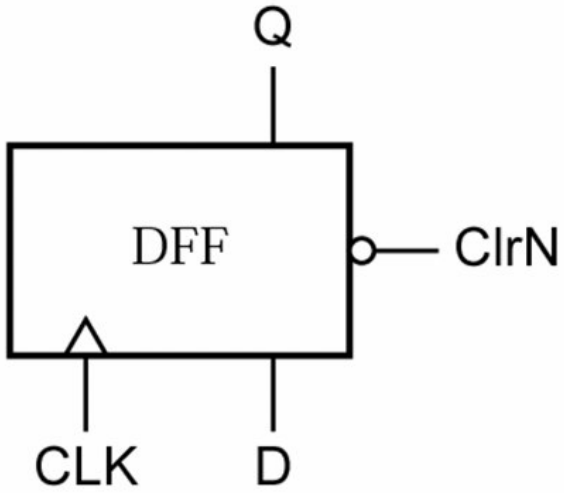

```
process (CLK)
begin
    if CLK'event and CLK = '1' -- rising edge of CLK
        then Q <= D;
    end  if;
end  process;
```

Figure 17-1: **VHDL Code for a Simple D Flip-Flop**



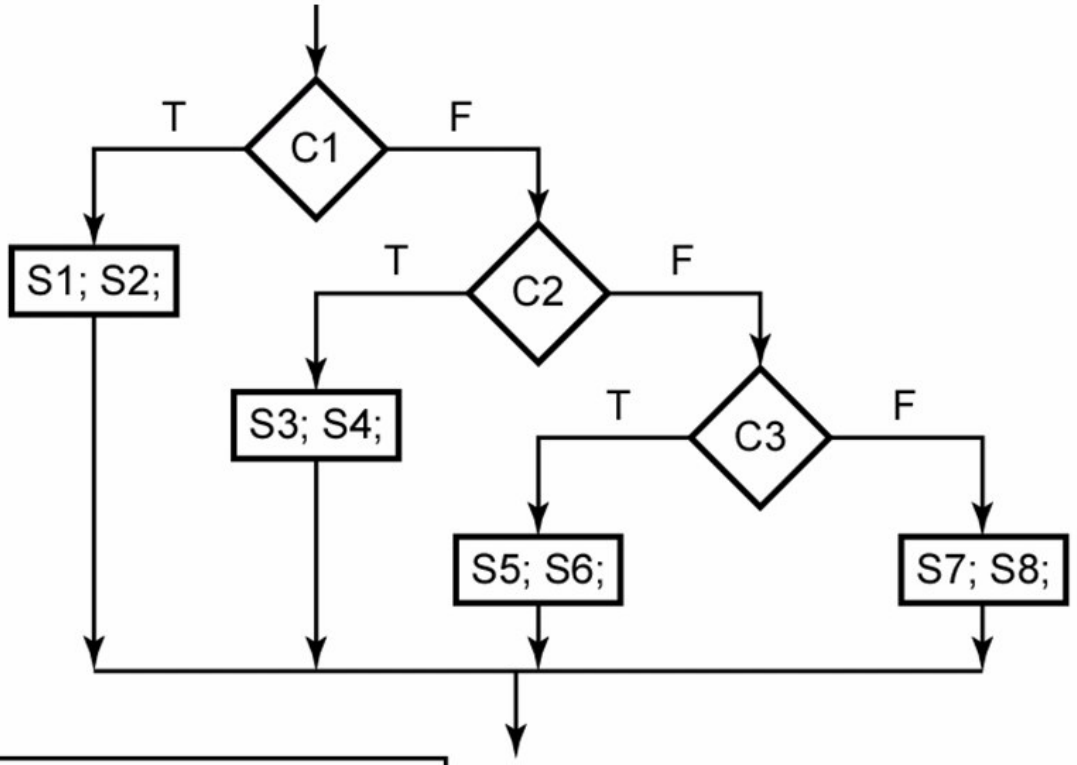

```
process (G,D)
begin
     if G = '1' then Q <= D; end  if;
end  process;
```

Figure 17-2: **VHDL Code for a Transparent Latch**




```
process (CLK, ClrN)
begin
    if ClrN = '0' then Q <= '0';
        else  if CLK'event and CLK = '1'
            then Q <= D;
            end  if;
    end  if;
end  process;
```

Figure 17-3: **VHDL Code for a D Flip-Flop with Asynchronous Clear**




```
if (C1) then S1; S2;
    else if (C2) then S3; S4;
        else if (C3) then S5; S6;
            else S7; S8;
            end if;
        end if;
end if;
```

```
if (C1) then S1; S2;
    elsif (C2) then S3; S4;
    elsif (C3) then S5; S6;
    else S7; S8;
end if;
```

Figure 17-4: **Equivalent Representations of a Flow Chart Using Nested Ifs and Elsifs**

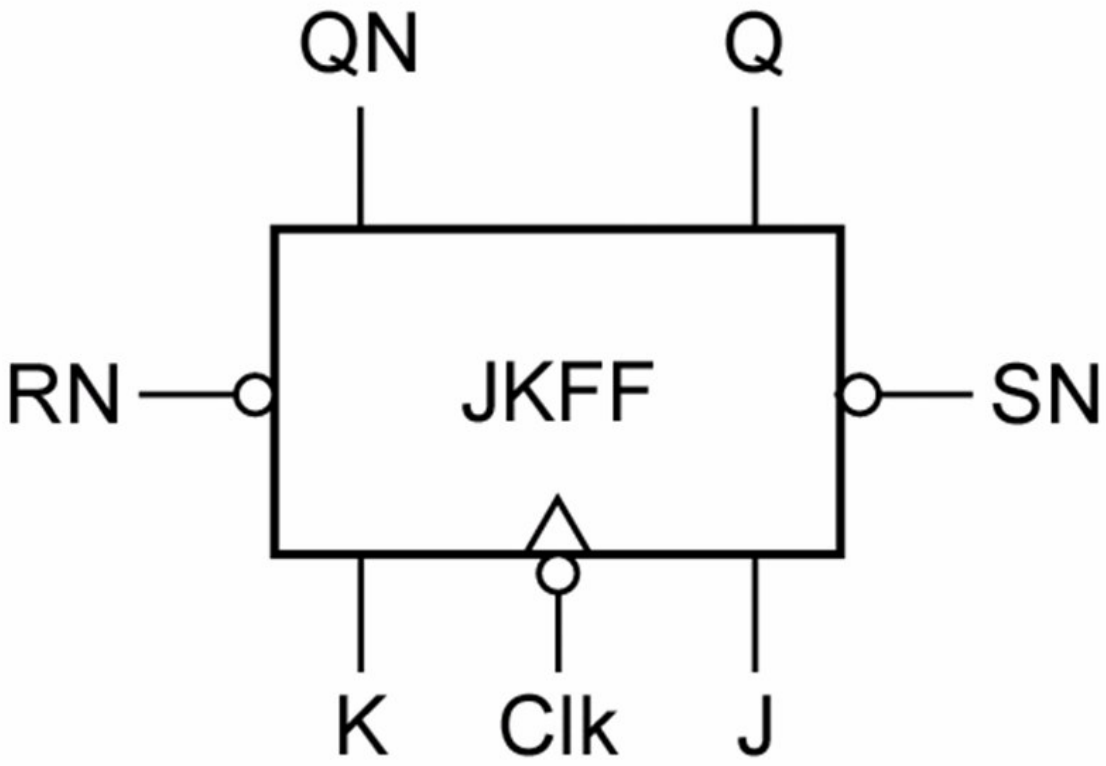
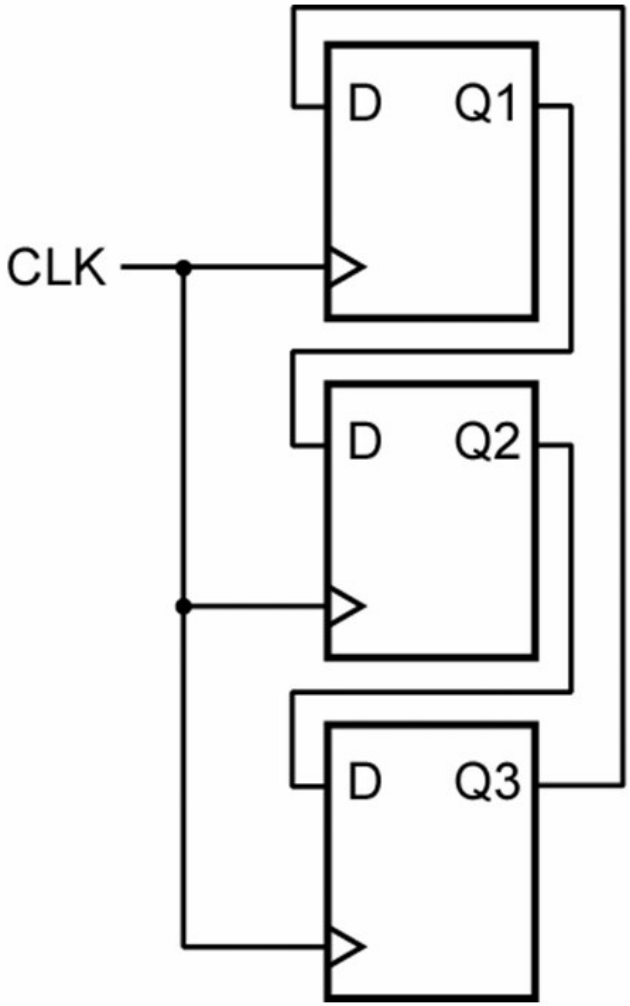


Figure 17-5: **J-K Flip-Flop**

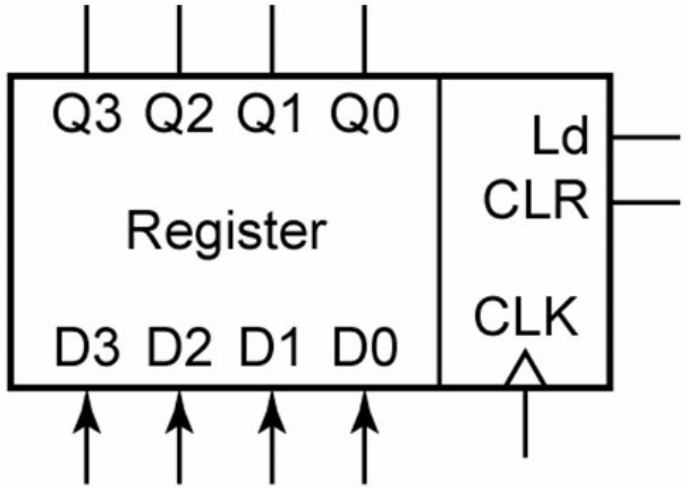
```
1  entity JKFF is
2  port (SN, RN, J, K, CLK: in bit;                    -- inputs
3        Q, QN: out bit);
4  end JKFF;

5  architecture JKFF1 of JKFF is
6  signal Qint: bit;                                   -- internal value of Q
7  begin
8     Q <= Qint; -- output Q and QN to port
9     QN <= not Qint;
10       process (SN, RN, CLK)
11       begin
12          if RN = '0' then  Qint <= '0' after 8 ns;      -- RN='0' will clear the FF
13          elsif SN = '0' then  Qint <= '1' after 8 ns;   -- SN='0' will set the FF
14          elsif CLK'event and CLK = '0' then             -- falling edge of CLK
15             Qint <= (J and not Qint) or (not K and Qint) after 10 ns;
16          end if;
17       end process;
18 end JKFF1;
```

Figure 17-6. J-K Flip-Flop Model



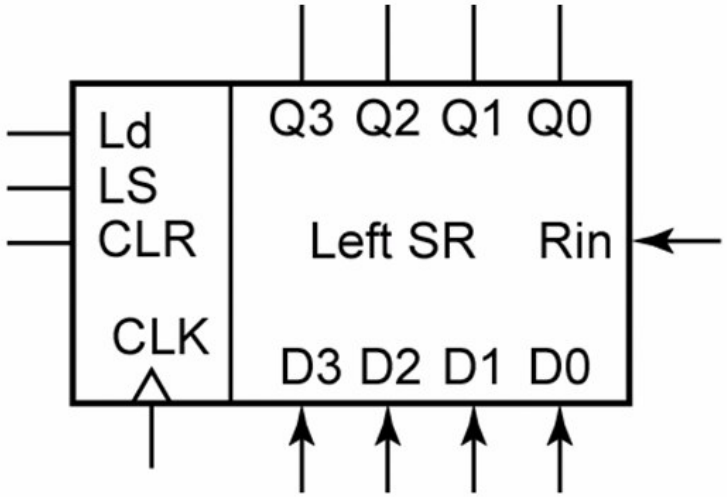

```
process (CLK)
begin
    if CLK'event and CLK = '1' then
        Q1 <= Q3 after 5 ns;
        Q2 <= Q1 after 5 ns;
        Q3 <= Q2 after 5 ns;
    end if;
end process;
```

Figure 17-7: **Cyclic Shift Register**



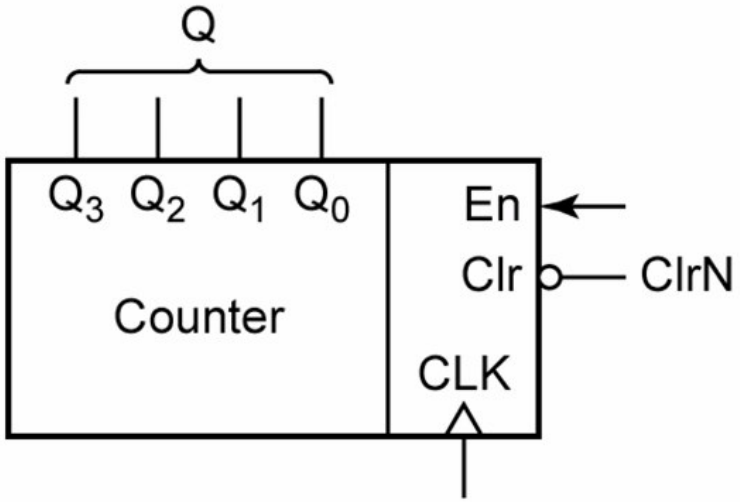

```
process (CLK)
begin
    if CLK'event and CLK = '1' then
        if CLR = '1' then Q <= "0000";
            elsif Ld = '1' then Q <= D;
        end  if;
    end  if;
end  process;
```

Figure 17-8: **Register with Synchronous Clear and Load**




```
process (CLK)
begin
    if CLK'event and CLK = '1' then
      if CLR = '1' then Q <= "0000";
         elsif Ld = '1' then Q <= D;
         elsif LS = '1' then Q <= Q(2 downto 0)& Rin;
      end  if;
    end  if;
end  process;
```

Figure 17-9: **Left-Shift Register with Synchronous Clear and Load**




```
signal Q: std_logic_vector(3 downto 0);
------------
process (CLK)
begin
     if CLK'event and CLK = '1' then
          if ClrN = '0' then Q <= "0000";
               elsif En = '1' then Q <= Q + 1;
          end  if;
     end  if;
end  process;
```

Figure 17-10: **VHDL Code for a Simple Synchronous Counter**

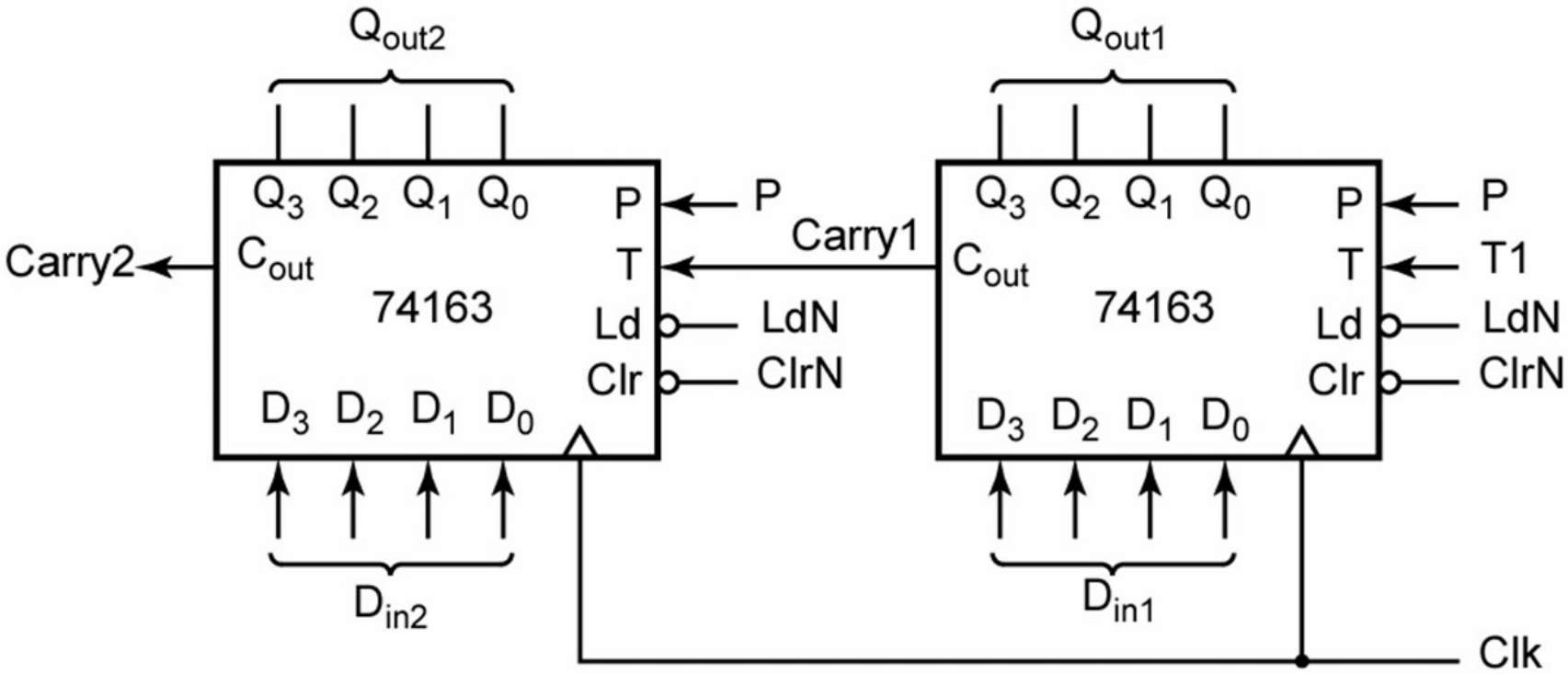


Figure 17-11: **Two 74163 Counters Cascaded to Form an 8-Bit Counter**

Table 17-1 74163 Counter Operation

Control Signals			Next State				
ClrN	LdN	PT	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
0	X	X	0	0	0	0	(clear)
1	0	X	D_3	D_2	D_1	D_0	(parallel load)
1	1	0	Q_3	Q_2	Q_1	Q_0	(no change)
1	1	1	present state + 1				(increment count)

```
-- 74163 FULLY SYNCHRONOUS COUNTER

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5 entity c74163 is
6    port(LdN, ClrN, P, T, ClK: in std_logic;
7         D: in std_logic_vector(3 downto 0);
8         Cout: out std_logic; Qout: out std_logic_vector(3 downto 0) );
9 end c74163;
```

Figure 17-12a. **74163 Counter Model**

```
10  architecture b74163 of c74163 is
11  signal Q: std_logic_vector(3 downto 0);      -- Q is the counter register
12  begin
13     Qout <= Q;
14     Cout <= Q(3) and Q(2) and Q(1) and Q(0) and T;
15     process (Clk)
16     begin
17        if Clk'event and Clk = '1' then            -- change state on rising edge
18           if ClrN = '0' then  Q <= "0000";
19              elsif LdN = '0' then Q <= D;
20              elsif (P and T) = '1' then Q <= Q+1;
21           end if;
22        end if;
23     end process;
24  end b74163;
```

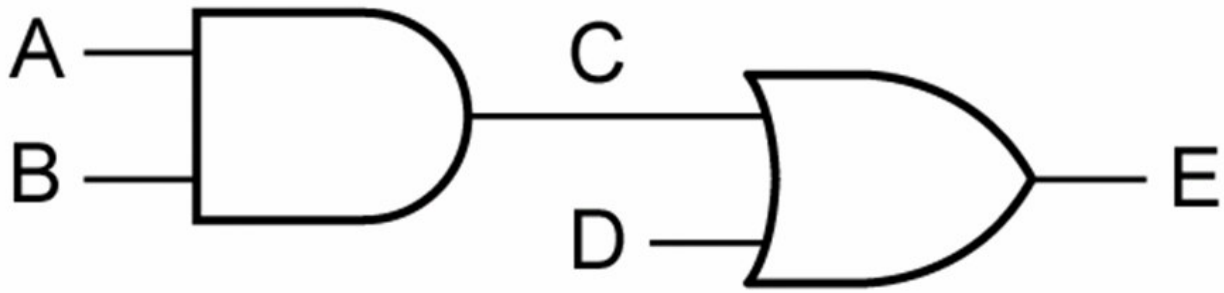
Figure 17-12b. **74163 Counter Model**

```
--Test module for 74163 counter

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;

5 entity c74163test is
6    port(ClrN,LdN,P,T1,Clk: in std_logic;
7       Din1, Din2: in std_logic_vector(3 downto 0);
8       Count: out integer range 0 to 255;
9       Carry2: out std_logic);
10  end c74163test;
```

Figure 17-13a. **VHDL for 8-Bit Counter**

```
11  architecture tester of c74163test is
12    component c74163
13      port(LdN, ClrN, P, T, Clk: in std_logic;
14        D: in std_logic_vector(3 downto 0);
15      Cout: out std_logic; Qout: out std_logic_vector(3 downto 0) );
16    end component;
17    signal Carry1: std_logic;
18    signal Qout1, Qout2: std_logic_vector(3 downto 0);
19  begin
20    ct1: c74163 port map (LdN,ClrN,P,T1,Clk,Din1,Carry1, Qout1);
21    ct2: c74163 port map (LdN,ClrN,P,Carry1,Clk,Din2,Carry2,Qout2);
22    Count <= Conv_integer(Qout2 & Qout1);
23  end tester;
```

Figure 17-13b. **VHDL for 8-Bit Counter**




```
process (A, B, C, D)
begin
      C <= A and B after 5 ns;
      E <= C or D after 5 ns;
end  process;
```

Figure 17-14: **VHDL Code for Gate Circuit**

```
signal sel: bit_vector(0 to 1);
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
sel  <=  A&B;     -- a concurrent statement, outside of the process
process (sel, I0, I1, I2, I3)
begin
   case sel is     -- a sequential statement in the process
      when "00" => F <= I0;
      when "01" => F <= I1;
      when "10" => F <= I2;
      when "11" => F <= I3;
      when others => null; -- required if sel is a std_logic_vector;
                     -- omit if sel is a bit_vector
   end case;
end process;
```

The Case Statement for Figure 10-7

Table 17-2. State Table for Code Converter

PS	NS		Z	
	X = 0	X = 1	X = 0	X = 1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	-	1	-

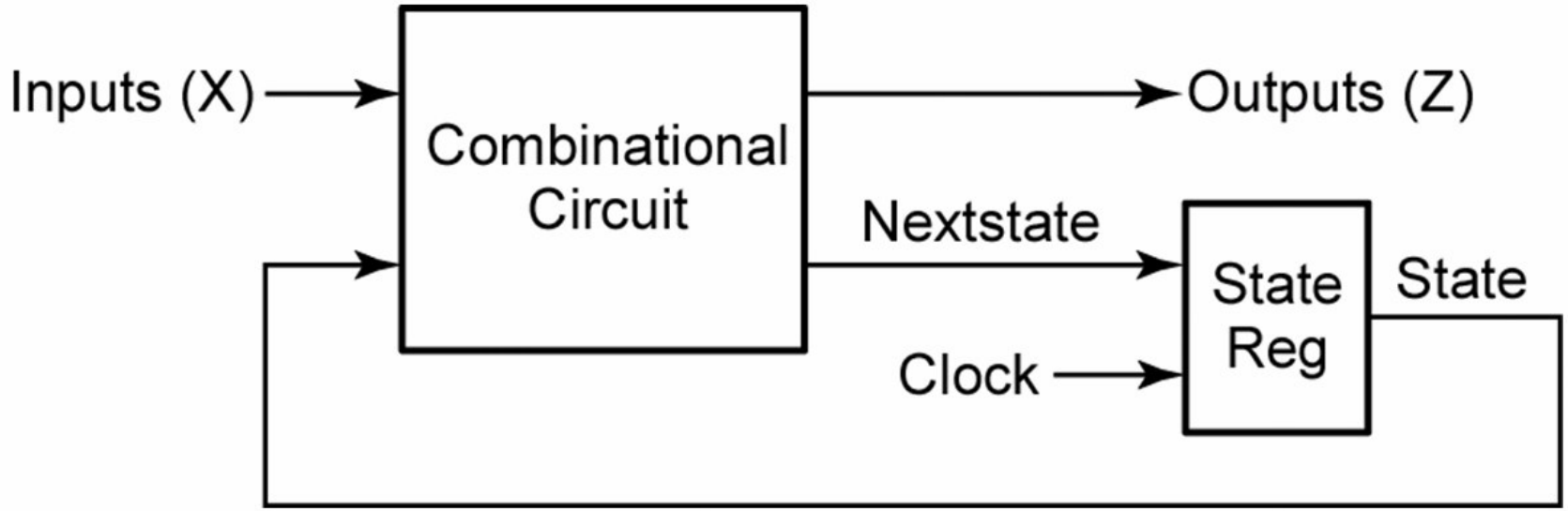


Figure 17-15: **General Model of Mealy Sequential Machine**

```
1 entity SM17_2 is
2    port(X, CLK: in bit;
3       Z: out bit);
4 end SM17_2;

5 architecture Table of SM17_2 is
6    signal State, Nextstate: integer range 0 to 6 := 0;
7 begin
8    process(State,X)                    --Combinational Circuit
9    begin
10       case State is
11       when 0 =>
12          if X='0' then Z<='1'; Nextstate<=1;
13          else Z<='0'; Nextstate<=2; end if;
14       when 1 =>
15          if X='0' then Z<='1'; Nextstate<=3;
16          else Z<='0'; Nextstate<=4; end if;
17       when 2 =>
18          if X='0' then Z<='0'; Nextstate<=4;
19          else Z<='1'; Nextstate<=4; end if;
```

Figure 17-16a: **Behavioral Model for Table 17-2**

```
20        when 3 =>
21          if X='0' then Z<='0'; Nextstate<=5;
22          else Z<='1'; Nextstate<=5; end if;
23        when 4 =>
24          if X='0' then Z<='1'; Nextstate<=5;
25          else Z<='0'; Nextstate<=6; end if;
26        when 5 =>
27          if X='0' then Z<='0'; Nextstate<=0;
28          else Z<='1'; Nextstate<=0; end if;
29        when 6 =>
30          Z<='1'; Nextstate<=0;
31        end case;
32      end process;

33      process(CLK)                    -- State Register
34        begin
35          if CLK'event and CLK='1' then      -- rising edge of clock
36          State <= Nextstate;
37        end if;
38      end process;
39  end Table;
```

Figure 17-16b: **Behavioral Model for Table 17-2**

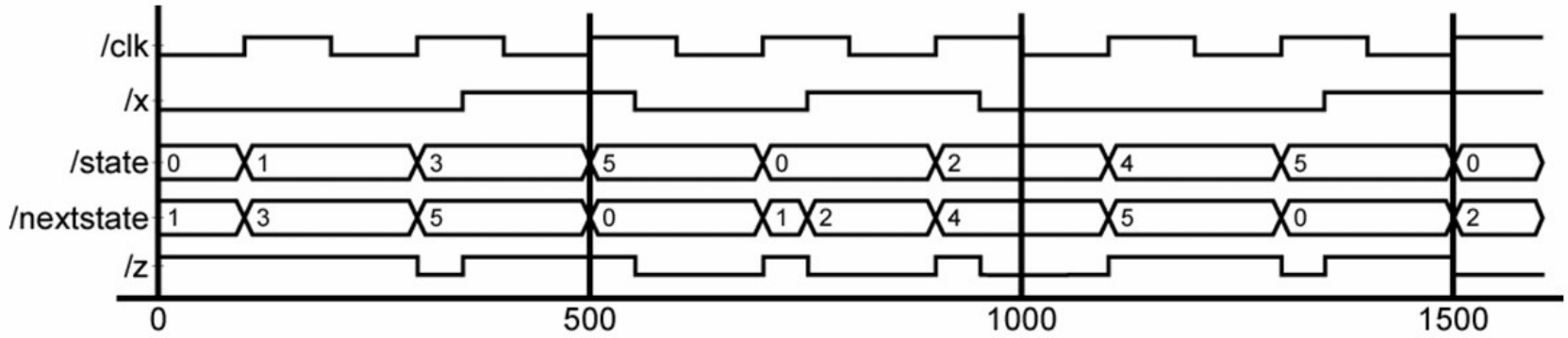


Figure 17-17: **Waveforms for Figure 17-16**

```
1 entity SM1_2 is
2    port(X,CLK: in bit;
3       Z: out bit);
4 end SM1_2;

5 architecture Equations1_4 of SM1_2 is
6    signal Q1,Q2,Q3: bit;
7 begin
8     process(CLK)
9     begin
10        if CLK'event and CLK='1' then   -- rising edge of clock
11           Q1<=not Q2 after 10 ns;
12           Q2<=Q1 after 10 ns;
13           Q3<=(Q1 and Q2 and Q3) or (not X and Q1 and not Q3) or
14                (X and not Q1 and not Q2) after 10 ns;
15        end if;
16   end process;
17     Z<=(not X and not Q3) or (X and Q3) after 20 ns;
18  end Equations1_4;
```

Figure 17-18. Sequential Machine Model Using Equations

```
-- The following is a STRUCTURAL VHDL description
--  of the circuit of Figure 16-4.

1 library BITLIB;
2 use BITLIB.bit_pack.all;

3 entity SM17_1 is
4    port(X,CLK: in bit;
5              Z: out bit);
6 end SM17_1;
```

Figure 17-19a: Structural Model of Sequential Machine

```
7  architecture Structure of SM17_1 is
8     signal A1,A2,A3,A5,A6,D3: bit:='0';
9     signal Q1,Q2,Q3: bit:='0';
10       signal Q1N,Q2N,Q3N, XN: bit:='1';
11    begin
12       I1:   Inverter port map (X,XN);
13       G1: Nand3 port map (Q1,Q2,Q3,A1);
14       G2: Nand3 port map (Q1,Q3N,XN,A2);
15       G3: Nand3 port map (X,Q1N,Q2N,A3);
16       G4: Nand3 port map (A1,A2,A3,D3);
17       FF1: DFF port map (Q2N,CLK,Q1,Q1N);
18       FF2: DFF port map (Q1,CLK,Q2,Q2N);
19       FF3: DFF port map (D3,CLK,Q3,Q3N);
20       G5: Nand2 port map (X,Q3,A5);
21       G6: Nand2 port map (XN,Q3N,A6);
22       G7: Nand2 port map (A5,A6,Z);
23    end Structure;
```

***Figure 17-19b*: Structural Model of Sequential Machine**

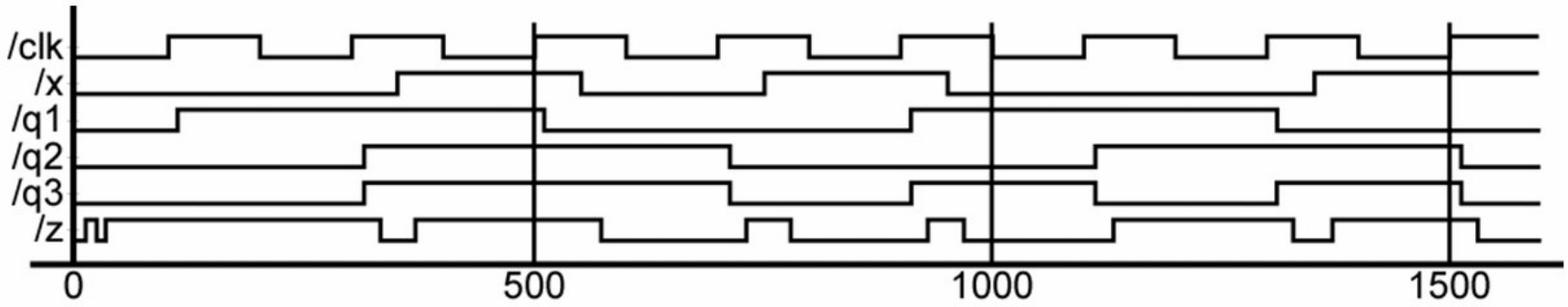


Figure 17-20: **Waveforms for Figure 16-4**

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;

5 entity SM16_5 is
6    Port ( X : in std_logic;
7           CLK : in std_logic;
8           Z : out std_logic);
9 end SM16_5;
```

***Figure 17-21a*: Sequential Machine Using a ROM**

```
10  architecture ROM of SM16_5  is
11  type ROM16X4 is array (0 to 15) of std_logic_vector(0 to 3);
12  constant ROM1: ROM16X4 := ("1001", "1011", "0100", "0101",
13                             "1101", "0000", "1000", "0000",
14                             "0010", "0100", "1100", "1101",
15                             "0110", "1000", "0000", "0000");
16  signal Q, D: std_logic_vector (1 to 3) := "000";
17  signal Index, Romout: std_logic_vector (0 to 3);
18  begin
19  Index <= X&Q;   -- X&Q is a 4-bit vector: X Q1 Q2 Q3
20  Romout <= ROM1 (conv_integer(Index));
      -- this statement reads the output from the ROM
      -- conv_integer converts Index to an Integer
21  Z <= Romout(0);
22  D <= Romout(1 to 3);

23  process (CLK)
24    begin
25      if CLK'event and CLK = '1' then Q <= D; end if;
26    end process;
27  end ROM;
```

***Figure 17-21b*: Sequential Machine Using a ROM**

```
1 entity Table_13_4 is
2    port(X1, X2, CLK: in bit; Z1, Z2: out bit);
3 end Table_13_4;

4 architecture T1 of Table_13_4 is
5 signal State, Nextstate: integer range 0 to 3:= 0;
6 signal X12: bit_vector(0 to 1);
7 begin
8    X12 <= X1&X2;
9    process(State, X12)
10      begin
11         case State is
12         when 0 =>
13            case X12 is
14               when "00" => Nextstate <= 3; Z1 <= '0'; Z2 <= '0';
15               when "01" => Nextstate <= 2; Z1 <= '1'; Z2 <= '0';
16               when "10" => Nextstate <= 1; Z1 <= '1'; Z2 <= '1';
17               when "11" => Nextstate <= 0; Z1 <= '0'; Z2 <= '1';
18               when others => null  -- not required since X is a bit_vector
19            end case;
20         when 1 =>     -- code for state 1 goes here, etc.
```

Figure 17-22 **Partial VHDL Code for the Table of Figure 13-4**

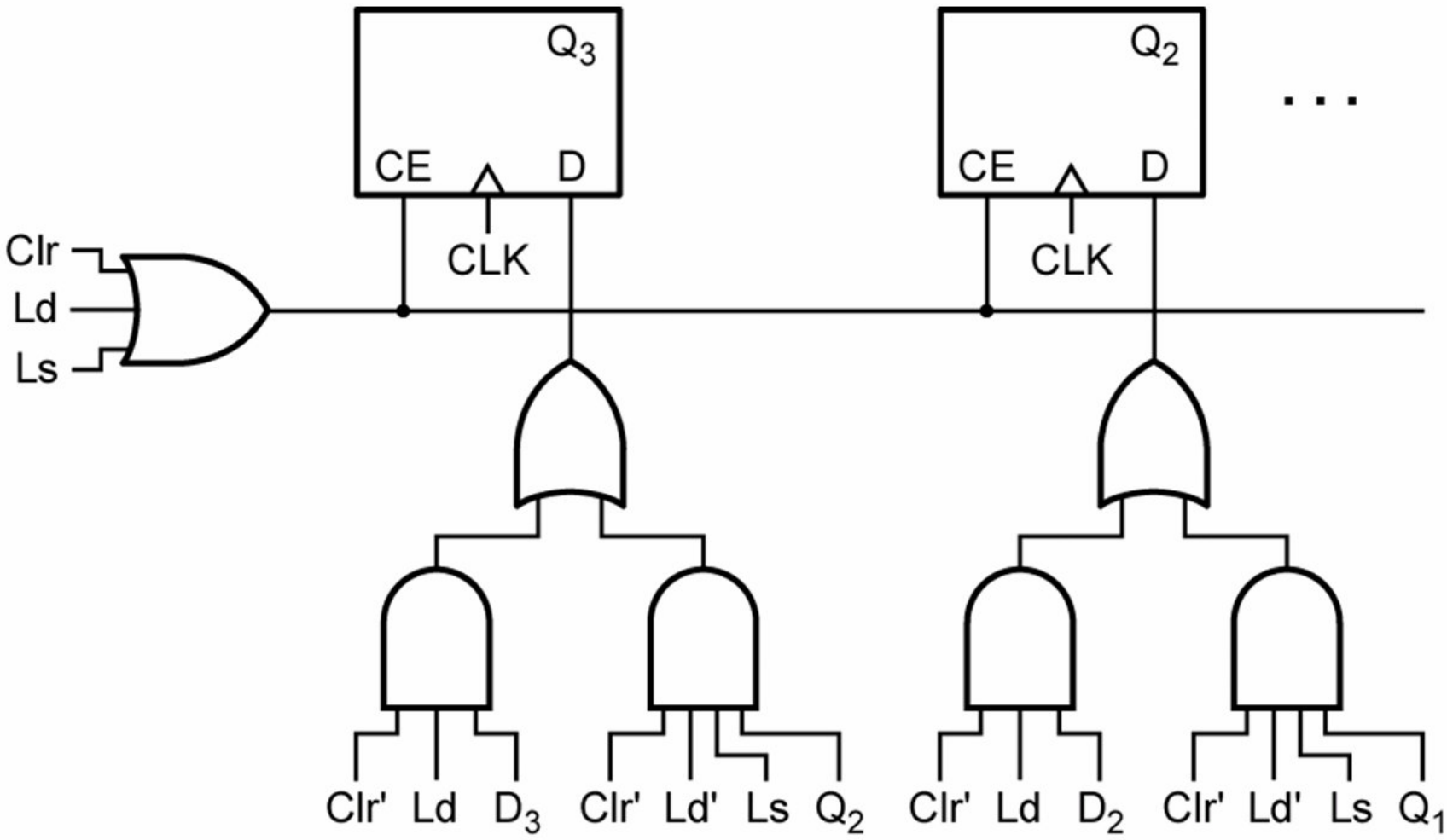


Figure 17-23: **Synthesis of VHDL Code from Figure 17-9**

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_UNSIGNED.ALL;

4 entity adder is
5     Port (B : in std_logic_vector(3 downto 0);
6    Ld, Ad, Cin, CLK : in std_logic;
7     Aout : out std_logic_vector(3 downto 0);
8      Cout : out std_logic);
9 end adder;
```

Figure 17-24a:

VHDL Code for 4-Bit Adder with Accumulator

```
10 architecture Behavioral of adder is
11 signal A : std_logic_vector(3 downto 0);
12 signal Addout : std_logic_vector(4 downto 0);
13 begin
14     Addout <= ('0' & A) + B + Cin;
15     Cout <= Addout(4);
16     Aout <= A;
17 process(CLK)
18 begin
19     if CLK'event and CLK = '1' then
20        if  Ld = '1' then A <= B;
21        elsif Ad = '1'
22           then A <= Addout(3 downto 0);
23        end if;
24     end if;
25  end process;
26  end Behavioral;
```

Figure 17-24b:

VHDL Code for 4-Bit Adder with Accumulator

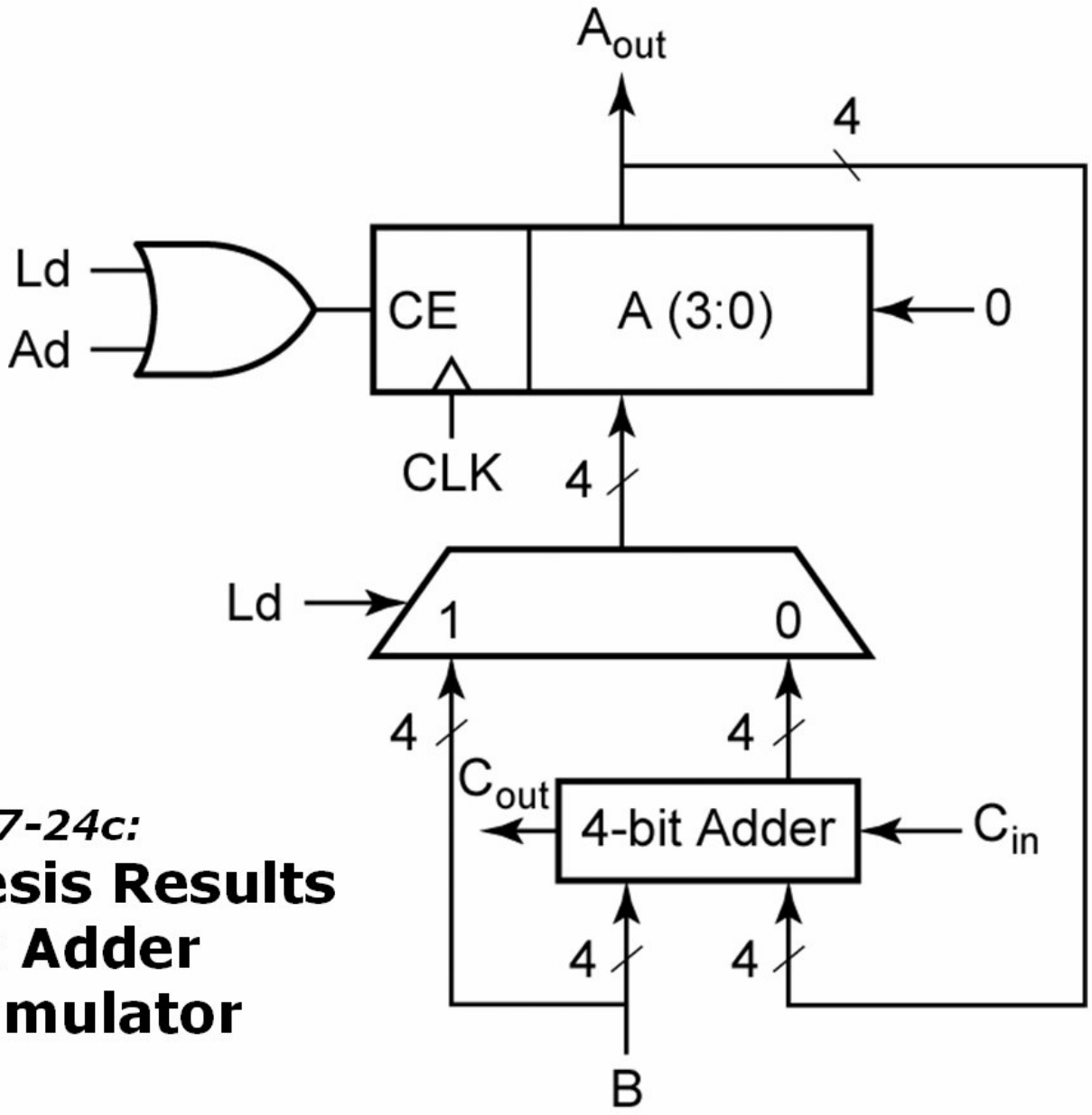


Figure 17-24c:

VHDL Synthesis Results for 4-Bit Adder with Accumulator