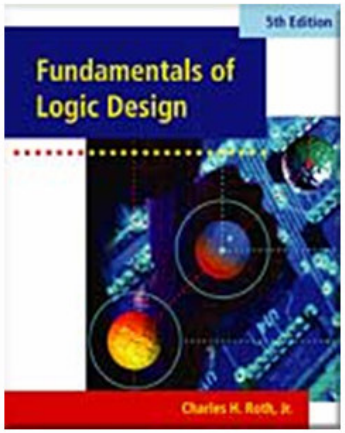
FIGURES FOR

CHAPTER 20

VHDL FOR DIGITAL SYSTEM DESIGN



This chapter in the book includes:

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

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;

3 entity serial is
4    Port ( St : in std_logic;
5           Clk : in std_logic;
6           Xout: out std_logic_vector(3 downto 0));
7 end serial;

8 architecture Behavioral of serial is
9       signal X, Y: std_logic_vector(3 downto 0);
10      signal Sh: std_logic;
11      signal Ci, Ciplus : std_logic;
12      signal Sumi: std_logic;
13      signal State, NextState : integer range 0 to 3;       -- 4 states
```

Figure 20-1a. VHDL Code for Figure 18-1

```
14    begin
15      Sumi <= X(0) xor Y(0) xor Ci;                    -- full adder
16      Ciplus <= (Ci and X(0)) or (Ci and Y(0)) or (X(0) and Y(0));
17      Xout <= X;
18    process (State, St)
19    begin
20      case State is
21        when 0 =>
22          if St = '1' then Sh <= '1'; Nextstate <= 1;
23          else Sh <= '0'; NextState <= 0; end if;
24        when 1 => Sh <= '1'; NextState <= 2;
25        when 2 => Sh <= '1'; NextState <= 3;
26        when 3 => Sh <= '1'; NextState <= 0;
27        end case;
28    end process;
```

Figure 20-1b. VHDL Code for Figure 18-1

```
29    process (clk)
30      begin
31      if clk'event and clk = '0' then
32      State <= Nextstate;        -- update state register
33        if Sh = '1' then
34          X <= Sumi & X(3 downto 1);          -- shift Sumi into X register
35          Y <= Y (0) & Y(3 downto 1);         -- rotate right Y register
36          Ci <= Ciplus; end if;               -- store next carry
37        end if;
38    end process;
39  end Behavioral;
```

Figure 20-1c. VHDL Code for Figure 18-1

```
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 mult4X4 is
6    port (Clk, St: in std_logic;
7       Mplier,Mcand : in std_logic_vector(3 downto 0);
8       Done: out std_logic;
9       Product: out std_logic_vector (7 downto 0));
10  end mult4X4;
```

Figure 20-2a.
Behavioral VHDL Code for Multiplier of Figure 18-6

```
11  architecture behave1 of mult4X4 is
12     signal State: integer range 0 to 9;
13     signal ACC: std_logic_vector(8 downto 0);  --accumulator
14     alias M: std_logic is ACC(0);              --M is bit 0 of ACC
15     begin
16        Product <= ACC (7 downto 0);
17        process (Clk)
18        begin
19           if Clk'event and Clk = '1' then   --executes on rising edge of clock
20           case State is
21           when 0=>                          --initial State
22              if St='1' then
23                 ACC(8 downto 4) <= "00000";        --clear upper ACC
24                 ACC(3 downto 0) <= Mplier;         --load the multiplier
25                 State <= 1;
26              end if;
```

Figure 20-2b.
Behavioral VHDL Code for Multiplier of Figure 18-6

```
27        when 1 | 3 | 5 | 7  =>                --"add/shift" State
28          if M = '1' then                     --Add multiplicand to ACC
29            ACC(8 downto 4) <= ('0'& ACC(7 downto 4)) + Mcand;
30            State <= State+1;
31          else ACC <= '0' & ACC(8 downto 1); --Shift accumulator right
32            State <= State + 2;
33          end if;
34        when 2 | 4 | 6 | 8 =>                 --"shift" State
35          ACC <= '0' & ACC(8 downto 1);       --Right shift
36          State <= State + 1;
37        when 9 =>                             --end of cycle
38          State <= 0;
39        end case;
40        end if;
41      end process;
42      Done <= '1' when State = 9 else '0';
43    end behave1;
```

Figure 20-2c.
Behavioral VHDL Code for Multiplier of Figure 18-6

```
-- command file to test multiplier
view list
add list CLK St State ACC done product
force st 1 2, 0 22
force clk 1 0, 0 10 -repeat 20
force Mcand 1101
force Mplier 1011
run 200
```

Figure 20-3a.
Command File for (13 by 11)

ns	delta	clk	St	State	ACC	done	product
0	+1	1	U	0	UUUUUUUUU	0	UUUUUUUU
2	+0	1	1	0	UUUUUUUUU	0	UUUUUUUU
10	+0	0	1	0	UUUUUUUUU	0	UUUUUUUU
20	+2	1	1	1	000001011	0	00001011
22	+0	1	0	1	000001011	0	00001011
30	+0	0	0	1	000001011	0	00001011
40	+2	1	0	2	011011011	0	11011011
50	+0	0	0	2	011011011	0	11011011
60	+2	1	0	3	001101101	0	01101101
70	+0	0	0	3	001101101	0	01101101
80	+2	1	0	4	100111101	0	00111101
90	+0	0	0	4	100111101	0	00111101
100	+2	1	0	5	010011110	0	10011110
110	+0	0	0	5	010011110	0	10011110
120	+2	1	0	7	001001111	0	01001111
130	+0	0	0	7	001001111	0	01001111
140	+2	1	0	8	100011111	0	00011111
150	+0	0	0	8	100011111	0	00011111
160	+2	1	0	9	010001111	1	10001111
170	+0	0	0	9	010001111	1	10001111
180	+0	1	0	0	010001111	0	10001111
190	+0	0	0	0	010001111	0	10001111
200	+0	1	0	0	010001111	0	10001111

Figure 20-3b.
Simulation Results for (13 by 11)

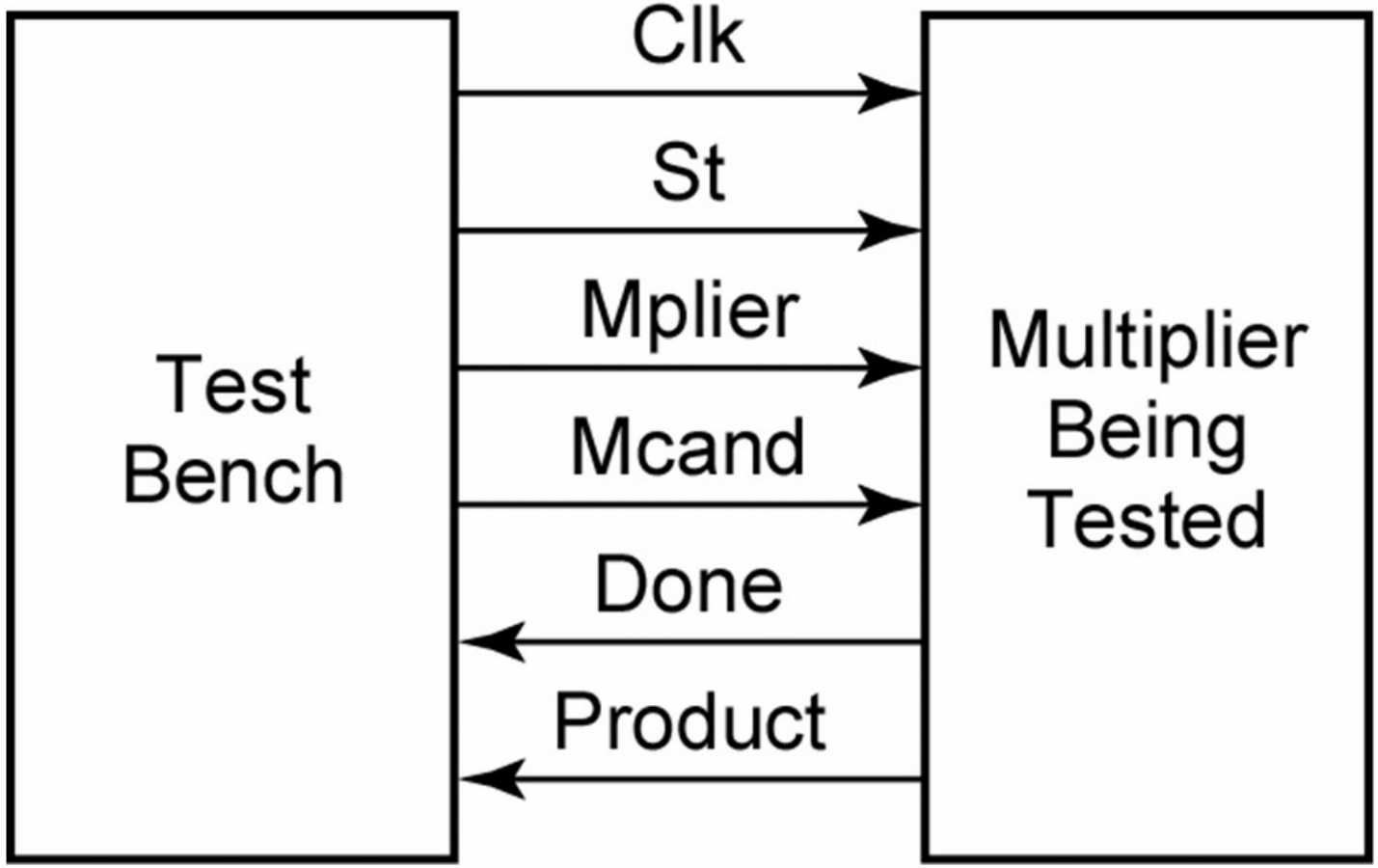


Figure 20-4: **Test Bench for Multiplier**

```
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 testmult is
6 end testmult;
7 architecture test1 of testmult is
8 component mult4X4
9 port (Clk: in std_logic;
10      St: in std_logic;
11      Mplier,Mcand : in std_logic_vector(3 downto 0);
12      Product: out std_logic_vector (7 downto 0);
13      Done: out std_logic);
14  end component;
15  constant N: integer := 6;
16  type arr is array(1 to N) of std_logic_vector(3 downto 0);
17  constant Mcandarr: arr :=
          ("1011", "1101", "0001","1000", "1111", "1101");
18  constant Mplierarr: arr :=
          ("1101", "1011", "0001","1000", "1111","0000");
```

Figure 20-5a. Testbench for Multiplier

```
19  signal CLK: std_logic :='0';
20  signal St, Done: std_logic;
21  signal Mplier, Mcand: std_logic_vector(3 downto 0);
22  signal Product: std_logic_vector(7 downto 0);
23  begin
24    mult1: mult4X4 port map(CLK, St, Mplier, Mcand, Product, Done);
25    CLK <= not CLK after 10 ns;  -- clock has 20 ns period
26    process
27    begin
28      for i in 1 to N loop
29        Mcand <= Mcandarr(i);
30        Mplier <= Mplierarr(i);
31        St <= '1';
32        wait until CLK = '1' and CLK'event;
33        St <= '0';
34        wait until done = '1' ;
35        wait until CLK = '1' and CLK'event;
36      end loop;
37    end process;
38  end test1;
```

Figure 20-5b.
Testbench for Multiplier

```
-- Command file to test multiplier
view list
add list -NOtrigger Mplier Mcand product -Trigger done
run 1320 ns
```

Figure 20-6a. Command File and Simulation of Multiplier

ns	+delta	mcand	mplier	product	done	
0	+0	UUUU	UUUU	UUUUUUUU	U	
0	+1	1011	1101	UUUUUUUU	0	
150	+2	1011	1101	10001111	1	11 x 13 = 143
170	+2	1011	1101	10001111	0	
330	+2	1101	1011	10001111	1	13 x 11 = 143
350	+2	1101	1011	10001111	0	
470	+2	0001	0001	00000001	1	1 x 1 = 1
490	+2	0001	0001	00000001	0	
610	+2	1000	1000	01000000	1	8 x 8 = 64
630	+2	1000	1000	01000000	0	
810	+2	1111	1111	11100001	1	15 x 15 = 225
830	+2	1111	1111	11100001	0	
930	+2	1101	0000	00000000	1	13 x 0 = 0

Figure 20-6b. Command File and Simulation of Multiplier

Figure 20-7a. Two-process VHDL Model for Multiplier

```
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 mult4X4 is
6    port (Clk, St: in std_logic;
7       Mplier,Mcand : in std_logic_vector(3 downto 0);
8       Product: out std_logic_vector (7 downto 0);
9       Done: out std_logic);
10  end mult4X4;

11  architecture control_signals of mult4X4 is
12     signal State, Nextstate: integer range 0 to 9;
13     signal ACC: std_logic_vector(8 downto 0);          --accumulator
14     alias M: std_logic is ACC(0);          --M is bit 0 of ACC
15     signal addout: std_logic_vector(4 downto 0);       -- adder output
                                                          including carry
16     signal Load, Ad, Sh: std_logic;
```

```
17  Begin
18        Product <= ACC (7 downto 0);
19        addout <= ('0' & ACC(7 downto 4)) + Mcand;
            -- uses "+" operator from the ieee._std_logic_unsigned package
20   process(State, St, M)
21   begin
22        Load <='0'; Ad<='0'; Sh <='0'; Done <='0';
23        case State is
24        when 0=>
25                  if St='1' then Load <='1'; Nextstate <= 1;
26                  else Nextstate <= 0; end if;
27        when 1 | 3 | 5 | 7  =>                --"add/shift" State
28                  if M = '1' then Ad <='1';  --Add multiplicand
29                  Nextstate <= State + 1;
30                  else Sh <='1'; Nextstate <= State + 2; end if;
```

Figure 20-7b. Two-process VHDL Model for Multiplier

```
31          when 2 | 4 | 6 | 8 =>      --"shift" State
32            Sh <='1'; Nextstate <= State + 1;
33          when 9 =>  Done <= '1'; Nextstate <= 0;
34          end case;
35        end process;

36        process (CLK)                          --Register update process
37        begin
38          if Clk'event and Clk = '1' then --executes on rising edge of clock
39            if Load = '1' then  ACC(8 downto 4) <= "00000";
40              ACC(3 downto 0) <= Mplier; end if;  --load the multiplier
41            if Ad = '1' then ACC(8 downto 4) <= addout; end if;
42            if Sh = '1' then  ACC <= '0' & ACC(8 downto 1);  end if;
43                                        -- Shift accumulator right
43            State <= Nextstate;
44          end if;
45      end process;
46    end control_signals;
```

Figure 20-7c. Two-process VHDL Model for Multiplier

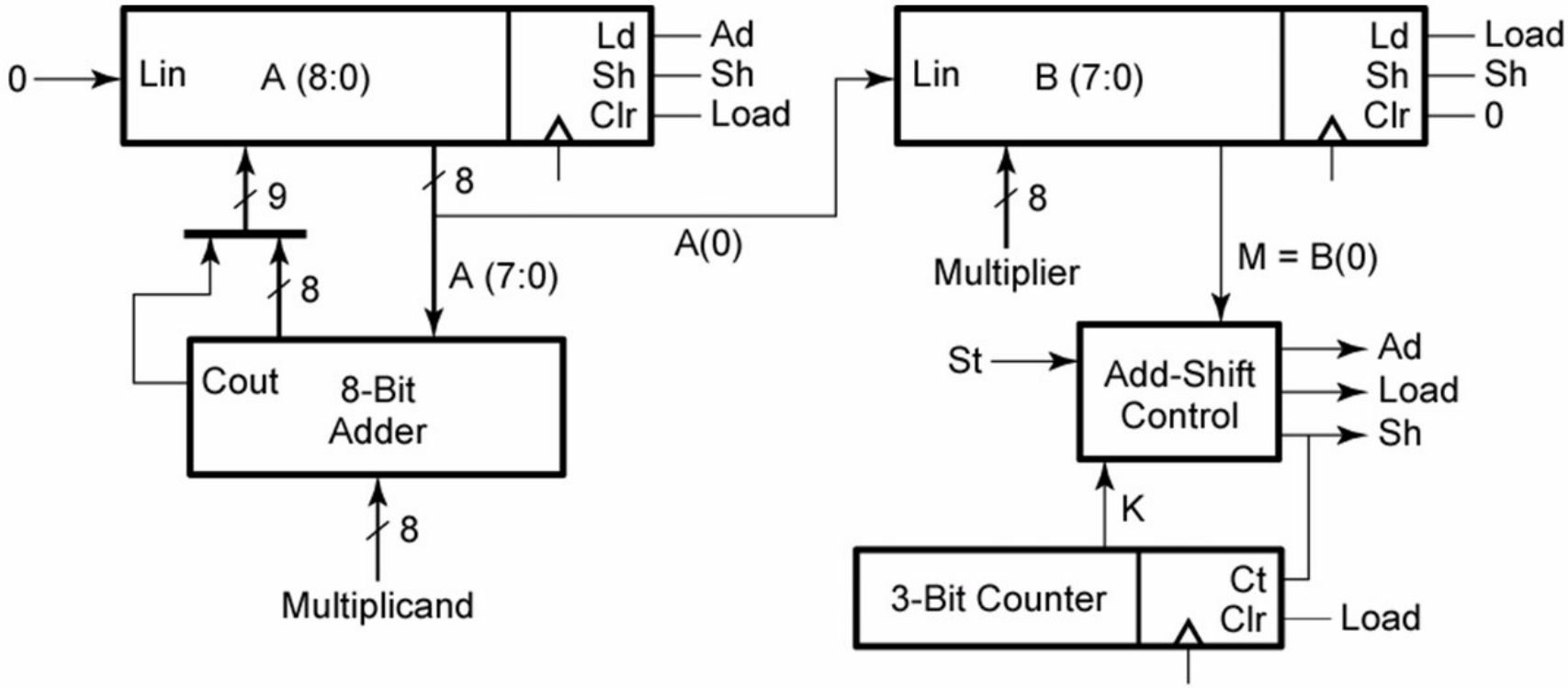


Figure 20-8: **Block Diagram for 8 x 8 Binary Multiplier**

```
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 mult8X8 is
6    Port (CLK, St: in std_logic;
7       Mplier, Mcand : in std_logic_vector(7 downto 0);
8       Done : out std_logic;
9       Product: out std_logic_vector(15 downto 0));
10  end mult8X8;
```

Figure 20-9a. VHDL Code for Multiplier with Shift Counter

```
11  architecture Behavioral of mult8X8 is
12     signal State, NextState: integer range 0 to 3;
13     signal count: std_logic_vector (2 downto 0):="000"; -- 3-bit counter
14     signal A: std_logic_vector (8 downto 0);              -- accumulator
15     signal B: std_logic_vector (7 downto 0);
16     alias M: std_logic is B(0);       -- M is bit 0 of B
17     signal addout: std_logic_vector (8 downto 0);
18     signal K, Load, Ad, Sh: std_logic;
19     begin
20        Product <= A(7 downto 0) & B;  -- 16-bit product is in A and B
21        addout <= '0' & A(7 downto 0) + Mcand;-- adder output is 9 bits
                                                -- including carry
22        K <= '1' when count = 7 else '0';
23        process (St, State, K, M)
24        begin
```

Figure 20-9b. VHDL Code for Multiplier with Shift Counter

```
25  Load <= '0'; Sh <='0'; Ad <='0'; Done <='0';
        -- control signals are '0' by default
26          case State is
27          when 0 =>
28            if St = '1' then Load <= '1'; NextState <= 1;
29            else NextState <= 0; end if;
30          when 1 =>
31            if M = '1' then Ad <= '1'; NextState <= 2;
32            else if K = '0' then Sh <= '1'; NextState <= 1;
33                 else Sh <= '1'; NextState <= 3; end if;
34            end if;
35          when 2 =>
36            if K = '0' then Sh <= '1'; NextState <= 1;
37            else Sh <= '1'; NextState <= 3;  end if;
38          when 3 =>
39            Done <='1'; NextState <= 0;
40          end case;
41     end process;
```

Figure 20-9c. VHDL Code for Multiplier with Shift Counter

```
42      process (clk)
43      begin
44        if clk'event and clk = '1' then
45          if load = '1' then
46            A <= "000000000";  Count <= "000";  -- clear A and counter
47            B <= Mplier;
48          end if;                    -- load multiplier
49          if Ad = '1' then A <= addout; end if;
50          if Sh = '1' then
              A <= '0' & A(8 downto 1);  B <= A(0)& B(7 downto 1);
51              -- right shift A and B
52            count <= count +1;                    -- increment counter
53              -- uses "+" operator from ieee_std_logic_unsigned package
54          end if;
55          State <= NextState;
56        end if;
57      end process;
58  end Behavioral;
```

Figure 20-9d. VHDL Code for Multiplier with Shift Counter

```
add wave clk st state count a b done product
force st 1 2,  0 22
force clk 1 0,  0 10 –repeat 20
force mcand 00001011
force mplier 00001101
run 280
```

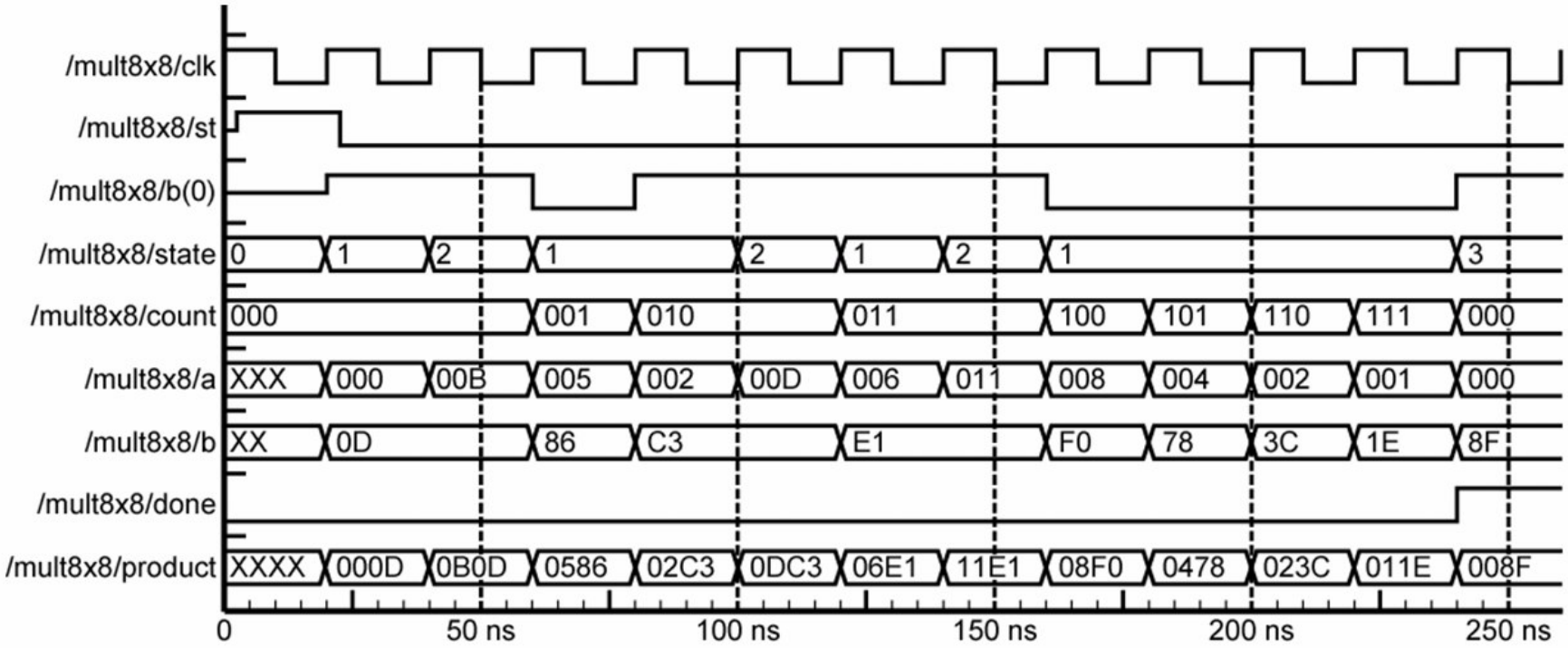


Figure 20-10: **Command File and Simulation of 8 x 8 Multiplier**

```
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 Divider is
6 Port (Dividend_in: in std_logic_vector(7 downto 0);
7       Divisor: in std_logic_vector(3 downto 0);
8       St, Clk: in std_logic;
9       Quotient: out std_logic_vector(3 downto 0);
10        Remainder: out std_logic_vector(3 downto 0);
11        Overflow : out std_logic);
12  end Divider;

13  architecture Behavioral of Divider is
14  signal State, NextState: integer range 0 to 5;
15  signal C, Load, Su, Sh: std_logic;
16  signal Subout : std_logic_vector (4 downto 0);
17  signal Dividend: std_logic_vector (8 downto 0);
```

Figure 20-11a. VHDL Code for Divider

```
18  begin
19     Subout <= Dividend(8 downto 4) - ('0' & divisor);
20     C <= not Subout (4);
21     Remainder <= Dividend (7 downto 4);
22     Quotient <= Dividend  (3 downto 0);

23  State_Graph: process (State, St, C)
24     begin
25        Load <= '0'; Overflow <='0'; Sh <= '0'; Su <= '0';
26     case State is
27        when 0 =>
28           if (St = '1') then Load <='1'; NextState <= 1;
29           else Nextstate <= 0; end if;
30        when 1 =>
31           if (C = '1') then Overflow <='1'; NextState <= 0;
32           else Sh <= '1'; NextState <= 2; end if;
33        when 2 | 3 | 4 =>
34           if (C = '1') then Su <= '1'; NextState <= State;
35           else Sh <= '1'; NextState <= State + 1; end if;
```

Figure 20-11b. VHDL Code for Divider

```
36        when 5 =>
37           if (C = '1') then Su <= '1'; end if;
38           NextState <= 0;
39     end case;
40  end process State_Graph;
41  Update: process (CLK)
42     begin
43     if CLK'event and CLK = '1' then              -- rising edge of CLK
44        State <= NextState;
45        if Load = '1' then Dividend <= '0' & Dividend_in; end if;
46        if Su = '1' then Dividend(8 downto 4) <= Subout; Dividend(0) <= '1';
          end if;
47        if Sh = '1' then Dividend <= Dividend (7 downto 0) & '0'; end if;
48     end if;
49  end process update;
50  end Behavioral;
```

Figure 20-11c. VHDL Code for Divider

```
1 entity DiceGame is
2    port (Rb, Reset, CLK: in bit;
3       Sum: in integer range 2 to 12;
4       Roll, Win, Lose: out bit);
5 end DiceGame;

6 architecture DiceBehave of DiceGame is
7    signal State, Nextstate: integer range 0 to 5;
8    signal Point: integer range 2 to 12;
9    signal Sp: bit;
10   begin
11      process(Rb, Reset, Sum, State)
12      begin
13         Sp <= '0'; Roll <= '0'; Win <= '0'; Lose <= '0';
```

Figure 20-12a VHDL Code for Dice Game Controller

```
14      case State is
15        when 0 =>
            if Rb = '1' then Nextstate <= 1; else nextstate <= 0; end if;
16        when 1 =>
17          if Rb = '1' then Roll <= '1'; Nextstate <= 1;
18          elsif Sum = 7 or Sum = 11 then Nextstate <= 2;
19          elsif Sum = 2 or Sum = 3 or Sum =12 then Nextstate <= 3;
20          else Sp <= '1'; Nextstate <= 4;
21        end if;
22      when 2 => Win <= '1';
23        if Reset = '1' then Nextstate <= 0; else nextstate <= 2; end if;
24      when 3 => Lose <= '1';
25        if Reset = '1' then Nextstate <= 0; else nextstate <= 3; end if;
26      when 4 => if Rb = '1' then Nextstate <= 5; else nextstate <= 4;
                  end if;
```

Figure 20-12b VHDL Code for Dice Game Controller

```
27        when 5 =>
28          if Rb = '1' then Roll <= '1'; Nextstate <= 5;
29            elsif Sum = Point then Nextstate <= 2;
30            elsif Sum = 7 then Nextstate <= 3;
31            else Nextstate <= 4;
32          end if;
33        end case;
34      end process;
35      process(CLK)
36      begin
37        if CLK'event and CLK = '1' then
38          State <= Nextstate;
39          if Sp = '1' then Point <= Sum; end if;
40        end if;
41      end process;
42    end DiceBehave;
```

Figure 20-12c VHDL Code for Dice Game Controller

```
1 entity Counter is
2 port(Clk, Roll: in bit;
3    Sum: out integer range 2 to 12);
4 end Counter;

5 architecture Count of Counter is
6 signal Cnt1,Cnt2: integer range 1 to 6 := 1;
7 begin
8    process (Clk)
9    begin
10       if Clk'event and Clk='1' then
11          if Roll='1' then
12             if Cnt1=6 then Cnt1 <= 1; else Cnt1 <= Cnt1+1; end if;
13             if Cnt1=6 then
14                if Cnt2=6 then Cnt2 <= 1; else Cnt2 <= Cnt2+1; end if;
15             end if;
16          end if;
17       end if;
18 end process;
19 Sum <= Cnt1 + Cnt2;
20 end Count;
```

Figure 20-13 Counter Module for Dice Game

```
1 entity Game is
2    port (Rb, Reset, Clk: in bit;
3       Win, Lose: out bit);
4 end Game;

5 architecture Play1 of Game is
6    component Counter
7       port(Clk, Roll: in bit;
8          Sum: out integer range 2 to 12);
9    end component;
10   component DiceGame
11      port (Rb, Reset, CLK: in bit;
12         Sum: in integer range 2 to 12;
13         Roll, Win, Lose: out bit);
14   end component;
15   signal roll1: bit;
16   signal sum1: integer range 2 to 12;
17   begin
18      Dice: Dicegame port map(Rb,Reset,Clk,sum1,roll1,Win,Lose);
19      Count:  Counter port map(Clk,roll1,sum1);
20   end Play1;
```

Figure 20-14 Main Module for Dice Game

Table 20-1 Synthesis Results (optimized for area)

Device		Multiplier Fig. 20-2	Multiplier Fig. 20-7	Multiplier Fig. 20-9	Divider Fig. 20-11	Dice Game Fig. 20-12 +
	Flip-Flops	13	13	22	12	13
Coolrunner CPLD	Macrocells	18	19	32	18	24
	Product terms	63	61	108	70	72
Spartan FPGA	4-input LUTs	38	32	36	23	31
	CLBs	20	18	19	14	16
Spartan II FPGA	4-input LUTs	30	30	35	30	30
	Slices	16	15	19	16	19