VHDL Modeling for Synthesis
Hierarchical Design

Textbook Section 4.8: Add and Shift Multiplier
“Add and shift” binary multiplication

Multiplicand $\rightarrow 1\ 1\ 0\ 1\ (13)$
Multiplier $\rightarrow 1\ 0\ 1\ 1\ (11)$

1 1 0 1
1 1 0 1
1 1 0 1

Shift & add

1 0 0 1 1 1
0 0 0 0

Shift & add

1 0 0 1 1 1
1 1 0 1

Shift & add

1 0 0 0 1 1 1 1 (143)
System Example: 8x8 multiplier

- **Multiplicand** (M)
- **Multiplier** (Q)
- **Adder** (ADR)
- **Accumulator** (A)
- **Multiplier** (Q)
- **Controller** (C)

**Controller outputs in red**

- LoadM
- LoadA
- ShiftA
- ClearA
- LoadQ
- ShiftQ

**States:**
- Start Clock
- Q₀
- Done
Roth example: Block diagram with control signals

FIGURE 4-25: Block Diagram for Binary Multiplier

[Diagram of a block diagram with control signals and a binary multiplier, labeled with inputs such as 'Load', 'Sh', 'Ad', 'Clk', 'Done', 'St', 'M', and outputs including 'Product', 'ACC', 'Multiplier', and 'Cm 4-Bit adder'.]
“Add and shift” multiply algorithm (Moore model)

A <- 0
M <- Multiplicand
Q <- Multiplier
CNT <- 0

A <- A + MQ(0)
A:Q <- right shift
CNT <- CNT + 1

Q(0)
1
ADD
Ad = 1

A:Q <- right shift
CNT <- CNT + 1

No
CNT = 4
Yes

START
DONE <- 1

HALT

Load=1
Ad = 1
Sh=1
**Example:**  $6 \times 5 = 110 \times 101$

<table>
<thead>
<tr>
<th>M</th>
<th>A</th>
<th>Q</th>
<th>CNT</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>0000</td>
<td>101</td>
<td>0</td>
<td>INIT Multiplicand-&gt;M, 0-&gt;A, Multiplier-&gt;Q, CNT=0</td>
</tr>
<tr>
<td></td>
<td>+ 110</td>
<td></td>
<td></td>
<td>ADD (Since Q0=1) A = A+M</td>
</tr>
<tr>
<td>0110</td>
<td>101</td>
<td>0</td>
<td></td>
<td>Shift A:Q, CNT+1=1 (CNT not 3 yet) (skip ADD, since Q0 = 0)</td>
</tr>
<tr>
<td>0011</td>
<td>010</td>
<td>1</td>
<td></td>
<td>Shift A:Q, CNT+1=2 (CNT not 3 yet)</td>
</tr>
<tr>
<td>0001</td>
<td>101</td>
<td>2</td>
<td></td>
<td>ADD (Since Q0 = 1) A = A+M</td>
</tr>
<tr>
<td></td>
<td>+ 110</td>
<td></td>
<td></td>
<td>Shift A:Q, CNT+1=2 (CNT= 3)</td>
</tr>
<tr>
<td>0111</td>
<td>101</td>
<td>2</td>
<td></td>
<td>HALT Done = 1</td>
</tr>
<tr>
<td>0011</td>
<td>110</td>
<td>3</td>
<td></td>
<td>P = 30</td>
</tr>
</tbody>
</table>

$P = 30$
Timing considerations

Be aware of register/flip-flop setup and hold constraints

Controller outputs change after its state changes

Register/Controller inputs set up before rising edge of clock

State/Registers change on rising edge of clock

Controller output: LoadR
Revised multiply algorithm

A <- 0
M <- Multiplicand
Q <- Multiplier
CNT <- 0

A <- A + MQ(0)
A:Q <- right shift
CNT <- CNT + 1

CNT = 4
?

No
Yes

Extra state needed before testing CNT
Control algorithm #1 state diagram

FIGURE 4-26: State Graph for Binary Multiplier Control
Control algorithm #2 – with bit counter
(Mealy model)

FIGURE 4-28: Multiplier Control with Counter

M = LSB of shifted multiplier
K = 1 after n shifts
Example – showing the counter

<table>
<thead>
<tr>
<th>Time</th>
<th>State</th>
<th>Counter</th>
<th>Product Register</th>
<th>St</th>
<th>M</th>
<th>K</th>
<th>Load</th>
<th>Ad</th>
<th>Sh</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>$S_0$</td>
<td>00</td>
<td>0000000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$S_0$</td>
<td>00</td>
<td>0000000000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$S_1$</td>
<td>00</td>
<td>0000010111</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$S_2$</td>
<td>00</td>
<td>0110110111</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$S_1$</td>
<td>01</td>
<td>001101101</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$t_5$</td>
<td>$S_2$</td>
<td>01</td>
<td>100111101</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$t_6$</td>
<td>$S_1$</td>
<td>10</td>
<td>010011110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$t_7$</td>
<td>$S_1$</td>
<td>11</td>
<td>001001111</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_8$</td>
<td>$S_2$</td>
<td>11</td>
<td>100011111</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$t_9$</td>
<td>$S_3$</td>
<td>00</td>
<td>010001111</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Multiplier – Top Level

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MultTop is
  port ( Multiplier: in std_logic_vector(3 downto 0);
         Multiplicand: in std_logic_vector(3 downto 0);
         Product:    out std_logic_vector(7 downto 0);
         Start:      in std_logic;
         Clk:        in std_logic;
         Done:       out std_logic);
end MultTop;

architecture Behavioral of MultTop is
  use work.mult_components.all;  -- component declarations
  -- internal signals to interconnect components
  signal Mout,Qout: std_logic_vector (3 downto 0);
  signal Dout,Aout: std_logic_vector (4 downto 0);
  signal Load,Shift,AddA: std_logic;
library ieee;
use ieee.std_logic_1164.all;
package mult_components is
component Controller -- Multiplier controller
    generic (N: integer := 2);
    port ( Clk: in   std_logic; --rising edge clock
            Q0: in   std_logic; --LSB of multiplier
            Start: in   std_logic; --start algorithm
            Load: out std_logic; --Load M,Q; Clear A
            Shift: out std_logic; --Shift A:Q
            AddA: out std_logic; --Adder -> A
            Done: out std_logic ); -- Algorithm completed
end component;
component AdderN -- N-bit adder, N+1 bit output
    generic (N: integer := 4);
    port( A,B: in std_logic_vector(N-1 downto 0);
        S: out std_logic_vector(N downto 0) );
end component;
component RegN -- N-bit register with load/shift/clear
    generic (N: integer := 4);
    port ( Din: in  std_logic_vector(N-1 downto 0); --N-bit input
            Dout: out std_logic_vector(N-1 downto 0); --N-bit output
            Clk: in  std_logic; --rising edge clock
            Load: in  std_logic; --Load enable
            Shift: in  std_logic; --Shift enable
            Clear: in  std_logic; --Clear enable
            SerIn: in  std_logic ); --Serial input
end component;
Multiplier – Top Level (continued)

begin

C: Controller generic map (2) -- Controller with 2-bit counter
port map (Clk,Qout(0),Start,Load,Shift,AddA,Done);

A: AdderN generic map (4) -- 4-bit adder; 5-bit output includes carry
port map (Aout(3 downto 0),Mout,Dout);

M: RegN generic map (4) -- 4-bit Multiplicand register
port map (Multiplicand,Mout,Clk,Load,'0','0','0');

Q: RegN generic map (4) -- 4-bit Multiplier register
port map (Multiplier,Qout,Clk,Load,Shift,'0',Aout(0));

ACC: RegN generic map (5) -- 5-bit Accumulator register
port map (Dout,Aout,Clk,AddA,Shift,Load,'0');

Product <= Aout(3 downto 0) & Qout; -- 8-bit product

end Behavioral;
Generic N-bit shift/load register entity

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

department RegN is
   generic (N: integer := 4);
   port (  Din: in  std_logic_vector(N-1 downto 0);  --N-bit input
          Dout: out std_logic_vector(N-1 downto 0);  --N-bit output
          Clk:  in  std_logic; --Clock (rising edge)
          Load: in  std_logic; --Load enable
          Shift: in std_logic; --Shift enable
          Clear: in std_logic; --Clear enable
          SerIn: in std_logic --Serial input
    );
end RegN;
Generic N-bit register architecture

architecture Behavioral of RegN is

  signal Dinternal: std_logic_vector(N-1 downto 0); -- Internal state

begin

  process (Clk)
  begin
    if (rising_edge(Clk)) then
      if (Clear = '1') then
        Dinternal <= (others => '0'); -- Clear
      elsif (Load = '1') then
        Dinternal <= Din; -- Load
      elsif (Shift = '1') then
        Dinternal <= SerIn & Dinternal(N-1 downto 1); -- Shift
      end if;
    end if;
  end process;

  Dout <= Dinternal; -- Drive outputs**

end Behavioral;

** With this inside the process, extra FFs were synthesized
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity AdderN is
  generic (N: integer := 4);
  port( A: in std_logic_vector(N-1 downto 0); -- N bit Addend
       B: in std_logic_vector(N-1 downto 0); -- N bit Augend
       S: out std_logic_vector(N downto 0) -- N+1 bit result, includes carry);
end AdderN;

architecture Behavioral of AdderN is
begin
  S <= std_logic_vector(('0' & UNSIGNED(A)) + UNSIGNED(B));
end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Controller is
  generic (N: integer := 2); -- # of counter bits
  port ( Clk: in std_logic; -- Clock (use rising edge)
         Q0: in std_logic; -- LSB of multiplier
         Start: in std_logic; -- Algorithm start pulse
         Load: out std_logic; -- Load M,Q and Clear A
         Shift: out std_logic; -- Shift A:Q
         AddA: out std_logic; -- Load Adder output to A
         Done: out std_logic -- Indicate end of algorithm
    );
end Controller;
Multiplier Controller - Architecture

architecture Behavioral of Controller is

type states is (HaltS, InitS, QtempS, AddS, ShiftS);
signal state: states := HaltS;
signal CNT: unsigned(N-1 downto 0);

begin

-- Moore model outputs to control the datapath
Done  <= '1' when state = HaltS else '0';   -- End of algorithm
Load  <= '1' when state = InitS else '0';    -- Load M/Q, Clear A
AddA  <= '1' when state = AddS else '0';    -- Load adder to A
Shift <= '1' when state = ShiftS else '0';  -- Shift A:Q

QtempS included for correct timing
Controller – State transition process

```
process(clk)
begin
    if rising_edge(Clk) then
        case state is
            when HaltS =>
                if Start = '1' then
                    state <= InitS;  -- Start pulse applied?
                    state <= InitS;  -- Start the algorithm
                end if;
            when InitS =>
                state <= QtempS;  -- Test Q0 at next clock**
            when QtempS =>
                if (Q0 = '1') then
                    state <= AddS;  -- Add if multiplier bit = 1
                else
                    state <= ShiftS;  -- Skip add if multiplier bit = 0
                end if;
            when AddS =>
                state <= ShiftS;  -- Shift after add
            when ShiftS =>
                if (CNT = 2**N - 1) then
                    state <= HaltS;  -- Halt after 2^N iterations
                else
                    state <= QtempS;  -- Next iteration of algorithm: test Q0 **
                end if;
        end case;
    end if;
end process;
```
Controller – Iteration counter

process(Clk)
begin
  if rising_edge(Clk) then
    if state = InitS then
      CNT <= to_unsigned(0,N); -- Reset CNT in InitS state
    elsif state = ShiftS then
      CNT <= CNT + 1; -- Count in ShiftS state
    end if;
  end if;
end process;
Multiplier test bench (main process)

Clk <= not Clk after 10 ns; -- 20ns period clock

process
begin
    for i in 15 downto 0 loop      -- 16 multiplier values
        Multiplier <= std_logic_vector(to_unsigned(i,4));
    end loop;

    for j in 15 downto 0 loop  -- 16 multiplicand values
        Multiplicand <= std_logic_vector(to_unsigned(j,4));
        Start <= '0', '1' after 5 ns, '0' after 40 ns;  -- 40 ns Start pulse
        wait for 50 ns;
        wait until Done = '1'; -- Wait for completion of algorithm
        assert (to_integer(UNSIGNED(Product)) = (i * j)) – Check Product
            report "Incorrect product" severity NOTE;
        wait for 50 ns;
    end loop;
end process;
Simulation results
Behavioral model (non-hierarchical)

FIGURE 4-27: Behavioral Model for 4 × 4 Binary Multiplier

-- This is a behavioral model of a multiplier for unsigned
-- binary numbers. It multiplies a 4-bit multiplicand
-- by a 4-bit multiplier to give an 8-bit product.

-- The maximum number of clock cycles needed for a
-- multiply is 10.

library IEEE;
use IEEE.numeric_bit.all;

entity mult4X4 is
  port(Clk, St: in bit;
       Mpltr, Mcand: in unsigned(3 downto 0);
       Done: out bit);
end mult4X4;

architecture behave1 of mult4X4 is
signal State: integer range 0 to 9;
signal ACC: unsigned(8 downto 0); -- accumulator
alias M: bit is ACC(0); -- M is bit 0 of ACC
begin
  process(Clk)
  begin
    if Clk'event and Clk = '1' then -- executes on rising edge of clock
      case State is
        when 0 => -- initial State
          if St = '1' then
            ACC(8 downto 4) <= "00000"; -- begin cycle
            ACC(3 downto 0) <= Mpltr; -- load the multiplier
            State <= 1;
          end if;
        end case;
  end process;
end;

Behavioral model (continued)

```vhdl
when 1 | 3 | 5 | 7 => -- "add/shift" State
    if M = '1' then -- add multiplicand
        ACC(8 downto 4) <= '0' & ACC(7 downto 4) + Mcand;
        State <= State + 1;
    else
        ACC <= '0' & ACC(8 downto 1); -- shift accumulator right
        State <= State + 2;
    end if;
when 2 | 4 | 6 | 8 => -- "shift" State
    ACC <= '0' & ACC(8 downto 1); -- right shift
    State <= State + 1;
when 9 => -- end of cycle
    State <= 0;
end case;
end if;
end process;
Done <= '1' when State = 9 else '0';
end behave1;
```
### Array multiplier (combinational)

**TABLE 4-3: Four-bit Multiplier Partial Products**

<table>
<thead>
<tr>
<th></th>
<th>(X_3)</th>
<th>(X_2)</th>
<th>(X_1)</th>
<th>(X_0)</th>
<th>(Y_3)</th>
<th>(Y_2)</th>
<th>(Y_1)</th>
<th>(Y_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_3)</td>
<td>(X_0)</td>
<td>(X_1)</td>
<td>(X_2)</td>
<td>(X_3)</td>
<td>(Y_0)</td>
<td>(Y_1)</td>
<td>(Y_2)</td>
<td>(Y_3)</td>
</tr>
<tr>
<td>(Y_3)</td>
<td>(Y_2)</td>
<td>(Y_1)</td>
<td>(Y_0)</td>
<td>(Y_3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Multiplier**
- **Partial product 0**
- **Partial product 1**
- **First row carries**
- **Partial product 2**
- **Second row carries**
- **Partial product 3**
- **Third row carries**
- **Final product**
Array multiplier circuit

FIGURE 4-29: Block Diagram of $4 \times 4$ Array Multiplier
FIGURE 4-30: VHDL Code for $4 \times 4$ Array Multiplier

```vhdl
entity Array_Mult is
    port(X, Y: in bit_vector(3 downto 0);
         P: out bit_vector(7 downto 0));
end Array_Mult;

architecture Behavioral of Array_Mult is
    signal C1, C2, C3: bit_vector(3 downto 0);
    signal S1, S2, S3: bit_vector(3 downto 0);
    signal XY0, XY1, XY2, XY3: bit_vector(3 downto 0);
    component FullAdder
        port(X, Y, Cin: in bit;
             Cout, Sum: out bit);
    end component;
    component HalfAdder
        port(X, Y: in bit;
             Cout, Sum: out bit);
    end component;
    begin
        XY0(0) <= X(0) and Y(0); XY1(0) <= X(0) and Y(1);
        XY0(1) <= X(1) and Y(0); XY1(1) <= X(1) and Y(1);
        XY0(2) <= X(2) and Y(0); XY1(2) <= X(2) and Y(1);
        XY0(3) <= X(3) and Y(0); XY1(3) <= X(3) and Y(1);

        XY2(0) <= X(0) and Y(2); XY3(0) <= X(0) and Y(3);
        XY2(1) <= X(1) and Y(2); XY3(1) <= X(1) and Y(3);
        XY2(2) <= X(2) and Y(2); XY3(2) <= X(2) and Y(3);
        XY2(3) <= X(3) and Y(2); XY3(3) <= X(3) and Y(3);

        FA1: FullAdder port map (XY0(2), XY1(1), C1(0), C1(1), S1(1));
        FA2: FullAdder port map (XY0(3), XY1(2), C1(0), C1(2), S1(2));
        FA3: FullAdder port map (S1(2), XY2(1), C2(0), C2(1), S2(1));
        FA4: FullAdder port map (S1(3), XY2(2), C2(1), C2(2), S2(2));
        FA5: FullAdder port map (C1(3), XY2(3), C2(2), C2(3), S2(3));
        FA6: FullAdder port map (S2(2), XY3(1), C3(0), C3(1), S3(1));
        FA7: FullAdder port map (S2(3), XY3(2), C3(1), C3(2), S3(2));
        FA8: FullAdder port map (C2(3), XY3(3), C3(2), C3(3), S3(3));
        HA1: HalfAdder port map (XY0(1), XY1(0), C1(0), S1(0));
        HA2: HalfAdder port map (XY1(3), C1(2), C1(3), S1(3));
        HA3: HalfAdder port map (S1(1), XY2(0), C2(0), S2(0));
        HA4: HalfAdder port map (S2(1), XY3(0), C3(0), S3(0));

        P(0) <= XY0(0); P(1) <= S1(0); P(2) <= S2(0);
        P(3) <= S3(0); P(4) <= S3(1); P(5) <= S3(2);
        P(6) <= S3(3); P(7) <= C3(3);
    end Behavioral;

-- Full Adder and half adder entity and architecture descriptions
-- should be in the project
```