4.1 Modeling Delay in VHDL - Propagation Delay

• Signals have a propagation delay of at least ________
• Variables change _______________
4.1 Modeling Delay in VHDL - Delta delay versus real time delay

entity STATEMENTS is
  port(X,Y,Z: in INTEGER; -- Note that entity ports are
  B: out INTEGER); -- always signals.
end STATEMENTS;

architecture PROP_DELAY of STATEMENTS is
  signal AS: INTEGER;
begin
  process (X,Y,Z)
  begin
    AS <= X*Y after 2 ns; --Statement (1)
    B  <= AS+Z after 2 ns; --Statement (2)
  end process;
end PROP_DELAY;

architecture DELTA_DELAY of STATEMENTS is
  signal AS: INTEGER;
begin
  process (X,Y,Z)
  begin
    AS <= X*Y; --Statement (1)
    B  <= AS+Z; --Statement (2)
  end process;
end DELTA_DELAY;

architecture INSTANTANEOUS of STATEMENTS is
begin
  process(X,Y,Z)
  variable AV,BV: INTEGER;
  begin
    AV := X*Y; --Statement (3)
    BV := AV+Z; --Statement (4)
    B <= BV;
  end process;
end INSTANTANEOUS;
4.1 Modeling Delay in VHDL - Delay and Concurrency

• Since logic signals flow in parallel, VHDL models of logic circuits must include provision for ________ of execution.
• All processes execute in parallel - a concurrent statement is a process with an ________ sensitivity list.
• Look at the simulation of the previous example

---

**Simulation of Time Delay**

```vhdl
entity BUFF is
  port(X: in BIT; Z: out BIT);
end BUFF;

architecture  ONE of BUFF is
begin
  process(X)
  variable Y1: BIT;
  begin
    Y1 := X;
    Z <= Y1 after 1 ns;
  end process;
end ONE;

architecture TWO of BUFF is
signal Y2: BIT;
begin
  Y2 <= X ;
  Z <= Y2 ;
end TWO;

architecture THREE of BUFF is
signal Y3: BIT;
begin
  Y3 <= X;
  Z <= Y3 after 1 ns;
end THREE;

architecture FOUR of BUFF is
signal Y4: BIT;
begin
  Y4 <= X after 1 ns;
  Z <= Y4 after 1 ns;
end FOUR;
```

---
4.1 Modeling Delay in VHDL - Inertial and Transport Delay

Z <= I after 10 ns;
Z <= transport I after 1 ns;

• Scheduling Rules
  
<table>
<thead>
<tr>
<th>Scheduling</th>
<th>Transport</th>
<th>Inertial</th>
</tr>
</thead>
<tbody>
<tr>
<td>New before existing</td>
<td>Overwrite existing</td>
<td>Overwrite existing</td>
</tr>
<tr>
<td>New after existing</td>
<td>Append new</td>
<td>If v_{new} = v_{existing}; append new</td>
</tr>
</tbody>
</table>

  Elsif t_{new} - t_{existing} > reject
  append new
  Else
  overwrite existing

• For stimuli, use transport or one statement with commas.
  Z <= '1' after 50 ns;
  Z <= '0' after 100 ns;

4.2 The VHDL Scheduling Algorithm

• This selection details how delay is implemented.

• We need to understand three concepts
  • Driver
  • Transaction
  • Waveform

• Once a transaction becomes current and simulation time then advances, that transaction is deleted.
### Simulation Example

**Signal Drivers**

```vhdl
entity simulation_example is
  port (A, B: bit);
end simulation_example;
architecture test1 of simulation_example is
  signal A, B: bit;
begin
  P1: process(A)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;
  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns; end if;
  end process P2;
end test1;
```

### Another Simulation Example

```vhdl
entity prob is
  port (D: inout bit);
end prob;
architecture PROB of PROB is
  signal A, B, C, E, F: bit;
begin
  P1: process (A, C)
  begin
    B <= A after 3 ns;
    E <= C after 5 ns;
  end process P1;
  C1: C <= A after 10 ns;
  P2: process (C, E)
  begin
    F <= C and E after 4 ns;
  end process P2;
  C2: D <= A or B or C or F after 1 ns;
end PROB;
```

Assume that A changes to '1' at 5 ns.
4.4 Logic Primitives

entity AND2 is
  generic(DEL: TIME);
  port(I1, I2: in BIT; O: out BIT);
end AND2;
architecture DF of AND2 is
begin
  O <= I1 and I2 after DEL;
end DF;

entity BUF is
  generic(DATA_DEL, Z_DEL: TIME);
  port(I, EN: in BIT; O: out BIT);
end BUF;
architecture ALG of BUF is
begin
  process(I, EN)
  begin
    if EN = '1' then
      O <= I after DATA_DEL;
    else
      O <= '1' after Z_DEL;
    end if;
  end process;
end ALG;

entity FULL_ADDER is
  generic(SUM_DEL, CARRY_DEL: TIME);
  port(A, B, CI: in BIT; SUM, COUT: out BIT);
end FULL_ADDER;
architecture DF of FULL_ADDER is
begin
  SUM <= A xor B xor CI after SUM_DEL;
  COUT <= (A and B) or (A and CI) or (B and CI) after CARRY_DEL;
end DF;
4.4 Logic Primitives

entity FOUR_TO_1_MUX is
  generic(DEL: TIME);
  port(IN0, IN1, IN2, IN3: in BIT_VECTOR(3 downto 0);
       SEL: in BIT_VECTOR(1 downto 0);
       O: out BIT_VECTOR(3 downto 0));
end FOUR_TO_1_MUX;

architecture DF of FOUR_TO_1_MUX is
begin
  O <= IN0 after DEL when SEL = "00" else
       IN1 after DEL when SEL = "01" else
       IN2 after DEL when SEL = "10" else
       IN3 after DEL;
end DF;

entity TWO_TO_4_DEC is
  generic(DEL: TIME);
  port(I: in BIT_VECTOR(1 downto 0);
       O: out BIT_VECTOR(3 downto 0));
end TWO_TO_4_DEC;

architecture ALG of TWO_TO_4_DEC is
begin
  process(I)
  begin
    case I is
    when "00" => O <= "0001" after DEL;
    when "01" => O <= "0010" after DEL;
    when "10" => O <= "0100" after DEL;
    when "11" => O <= "1000" after DEL;
    end case;
  end process;
end ALG;
4.4 Logic Primitives

entity FOUR_TO_2_ENC is
  generic(DEL: TIME);
  port(I: in  BIT_VECTOR(3 downto 0);
       O: out BIT_VECTOR(1 downto 0));
end FOUR_TO_2_ENC;

architecture DF of FOUR_TO_2_ENC is
begin
  O <= "00" after DEL when I(0) = '1' else
       "01" after DEL when I(1) = '1' else
       "10" after DEL when I(2) = '1' else
       "11" after DEL;
end DF;

entity SHIFTER is
  generic(DEL: TIME);
  port(DATA_IN: in BIT_VECTOR(3 downto 0);
       SR,SL, IL,IR: in BIT;
       DATA_OUT: out BIT_VECTOR(3 downto 0));
end SHIFTER;

architecture ALG of SHIFTER is
begin
  process(SR,SL,DATA_IN,IL,IR)
  variable CON: BIT_VECTOR(0 to 1);
  begin
    CON := SR&SL;
    case CON is
      when "00" => DATA_OUT <= DATA_IN after DEL;
      when "01" => DATA_OUT <= DATA_IN(2 downto 0) & IL
                  after DEL;
      when "10" => DATA_OUT <= IR & DATA_IN(3 downto 1)
                  after DEL;
      when "11" => DATA_OUT <= DATA_IN after DEL;
    end case;
  end process;
end ALG;
4.4 Logic Primitives

use work.PRIMS.all;
entity ALU is
  generic(DEL: TIME);
  port(A,B: in BIT_VECTOR(3 downto 0); CI: in BIT;
      FSEL: in BIT_VECTOR(1 downto 0);
      F: out BIT_VECTOR(3 downto 0); COUT: out BIT);
end ALU;
architecture ALG of ALU is
begin
  process(A,B,CI,FSEL)
  variable  FV: BIT_VECTOR(3 downto 0);
  variable COUTV: BIT;
  begin
    case FSEL is
      when "00" => F <= A after DEL;
      when "01" => F <= not(A) after DEL;
      when "10" => ADD(A,B,CI,FV,COUTV);
        F <= FV after DEL;
        COUT <= COUTV after DEL;
      when "11" => F <= A and B after DEL;
      end case;
  end process;
end ALG;

package PRIMS is
  function INC(X : BIT_VECTOR) return BIT_VECTOR;
  function DEC(X : BIT_VECTOR) return BIT_VECTOR;
  function INTVAL(VAL : BIT_VECTOR) return INTEGER;
end PRIMS;
package body PRIMS is
  function INC(X : BIT_VECTOR) return BIT_VECTOR is
    variable SUMV: BIT_VECTOR(X'LENGTH-1 downto 0);
    variable CARRY: BIT;
    begin
      SUMV := A;
      CARRY := CIN;
      for I in 0 to SUMV'HIGH loop
        SUMV(I) := (A(I) xor B(I)) xor CARRY;
        CARRY := (A(I) and B(I)) or (A(I) and CARRY)
                   or (B(I) and CARRY);
      end loop;
      SUM := SUMV;
    end function;

4.4 Logic Primitives

function INC(X : BIT_VECTOR) return BIT_VECTOR is
  variable XV: BIT_VECTOR(X'LENGTH-1 downto 0);
begin
  XV := X;
  for I in 0 to XV'HIGH loop
    if XV(I) = '0' then
      XV(I) := '1';
      exit;
    else XV(I) := '0';
    end if;
  end loop;
  return XV;
end INC;

function DEC(X : BIT_VECTOR) return BIT_VECTOR is
  variable XV: BIT_VECTOR(X'LENGTH-1 downto 0);
begin
  XV := X;
  for I in 0 to XV'HIGH loop
    if XV(I) = '1' then
      XV(I) := '0';
      exit;
    else XV(I) := '1';
    end if;
  end loop;
  return XV;
end DEC;

function INTVAL ( VAL: BIT_VECTOR) return INTEGER is
  variable VALV: BIT_VECTOR(VAL'LENGTH - 1 downto 0);
  variable SUM: INTEGER := 0;
begin
  VALV := VAL;
  for N in VALV'LOW to VALV'HIGH loop
    if VALV(N) = '1' then
      SUM := SUM + (2**N);
    end if;
  end loop;
  return SUM;
end INTVAL;
end PRIMS;
4.4 Logic Primitives

```vhdl
entity PLA is
    generic(AND_DEL, OR_DEL: TIME);
    port(X: in BIT_VECTOR(1 to 3);   Z: out BIT_VECTOR(1 to 4));
end PLA;
architecture CONNECTION_MATRIX of PLA is
    signal R: BIT_VECTOR(1 to 4);
    begin
        AND_PLANE: process(X)
            variable RV: BIT_VECTOR(1 to 4);
            type AND_ARRAY is array( 1 to 4, 1 to 3, 1 to 2 ) of BIT;
            variable AND_PL: AND_ARRAY :=
                ((('0','1'),('0','0'),('0','0')), (('0','0'),('1','0'),('1','0')),
                (('1','0'),('1','0'),('0','1')), (('1','0'),('0','1'),('1','0')));
            begin
                for I in 1 to 4 loop
                    RV(I) := '0';
                    for J in 1 to 3 loop
                        if AND_PL(I,J,1) = '1' then RV(I) := RV(I) or X(J);
                        end if;
                        if AND_PL(I,J,2) = '1' then RV(I) := RV(I) or not X(J);
                        end if;
                    end loop;
                    R(I) <= not RV(I) after AND_DEL;
                end loop;
            end process AND_PLANE;
```
4.4 Logic Primitives

**OR_PLANE:**

```vhdl
process(R)
variable ZV: BIT_VECTOR(1 to 4);
type OR_ARRAY is array(1 to 4,1 to 4) of BIT;
variable OR_PLANE: OR_ARRAY :=
    (('1','0','0','0'),('1','0','1','0'),
     ('0','1','0','0'),('0','0','1','1'));
begin
    for I in 1 to 4 loop
        ZV(I) := '0';
        for J in 1 to 4 loop
            if OR_PLANE(I,J) = '1' then ZV(I) := ZV(I) or R(J);
        end loop;
        Z(I) <= ZV(I) after OR_DEL;
    end loop;
end process OR_PLANE;
end CONNECTION_MATRIX;
```

**JKFF:**

```vhdl
entity JKFF is
generic(SRDEL,CLKDEL: TIME);
port(S,R,J,K,CLK: in BIT; Q,QN: inout BIT);
end JKFF;
architecture ALG of JKFF is
begin
    process(CLK,S,R)
    begin
        if S = '1' and R = '0' then
            Q <= '1' after SRDEL;  QN <= '0' after SRDEL;
        elsif S = '0' and R = '1' then
            Q  <= '0' after SRDEL; QN <= '1' after SRDEL;
        elsif CLK'EVENT and CLK = '1' and S='0' and R='0' then
            if J = '1' and K = '0' then
                Q <= '1' after CLKDEL; QN <= '0' after CLKDEL;
            elsif J = '0' and K ='1' then
                Q <= '0' after CLKDEL; QN <= '1' after CLKDEL;
            elsif J = '1' and K = '1' then
                Q <= not Q after CLKDEL; QN <= not QN after CLKDEL;
            end if;
        end if;
    end process;
end ALG;
```
4.4 Logic Primitives

entity REG is
generic(DEL: TIME);
port(RESET, LOAD, CLK: in BIT;
DATA_IN: in BIT_VECTOR(3 downto 0);
Q: inout BIT_VECTOR(3 downto 0));
end REG;

architecture DF of REG is
begin
REG: block(not CLK'STABLE and CLK = '1')
begin
Q <= guarded "0000" after DEL when RESET = '1' else
DATA_IN after DEL when LOAD = '1' else
Q;
end block REG;
end DF;

entity LATCH is
generic(LATCH_DEL:TIME);
port(D: in BIT_VECTOR(7 downto 0);
CLK: in BIT; LOUT: out BIT_VECTOR(7 downto 0));
end LATCH;

architecture DFLOW of LATCH is
begin
LATCH: block(CLK = '1')
begin
LOUT <= guarded D after LATCH_DEL;
end block LATCH;
end DFLOW;
4.4 Logic Primitives

entity SHIFTREG is
  generic(DEL: TIME);
  port(DATA_IN: in BIT_VECTOR(3 downto 0);
       CLK,LOAD,SR,SL: in BIT;
       IL,IR: in BIT;
       Q: inout BIT_VECTOR(3 downto 0));
end SHIFTREG;

architecture DF of SHIFTREG is
begin
  SH:block(not CLK'STABLE and CLK = '1')
  begin
    Q <= guarded DATA_IN after DEL when LOAD = '1' else
         Q(2 downto 0) & IL after DEL when SL='1' and SR='0' else
         IR & Q(3 downto 1) after DEL when SL='0' and SR='1' else
         Q;
  end block SH;
end DF;

entity COUNTER is
  generic(DEL: TIME);
  port(RESET,LOAD,COUNT,UP,CLK: in BIT;
       DATA_IN: in BIT_VECTOR(3 downto 0);
       CNT: inout BIT_VECTOR(3 downto 0));
end COUNTER;

use work.PRIMS.all;
architecture ALG of COUNTER is
begin
  process(CLK)
  begin
    if CLK = '1' then
      if RESET = '1' then
        CNT <= "0000" after DEL;
      elsif LOAD = '1' then
        CNT <= DATA_IN after DEL;
      elsif COUNT = '1' then
        if UP = '1' then
          CNT <= INC(CNT) after DEL;
        else
          CNT <= DEC(CNT) after DEL;
        end if;
      end if;
    end if;
  end process;
end ALG;
4.4 Logic Primitives

use work.PRIMS.all;

entity RAM is
  generic (RDEL, DISDEL: TIME);
  port (DATA: in BIT_VECTOR(3 downto 0);
  ADDRESS: in BIT_VECTOR(4 downto 0);
  RD, WR, NCS: in BIT);
end RAM;

architecture SIMPLE of RAM is
  type MEMORY is array(0 to 31) of BIT_VECTOR(3 downto 0);
begin
  process(RD, WR, NCS, ADDRESS, DATA)
  variable MEM: MEMORY;
  begin
    if NCS='0' then
      if RD='1' then
        DATA <= MEM(INTVAL(ADDRESS)) after RDEL;
      elsif WR='1' then
        MEM(INTVAL(ADDRESS)) := DATA;
      end if;
    else
      DATA <= "1111" after DISDEL;
    end if;
  end process;
end SIMPLE;

4.4 Text I/O Test Bench

use STD.TEXTIO.all;

architecture ONES_CNT1 of TEST_BENCH is
  signal PLAY: BIT;
  signal A: BIT_VECTOR(2 downto 0);
  signal C: BIT_VECTOR(1 downto 0);
  component ONES_CNTA
    port (A: in BIT_VECTOR(2 downto 0); C: out BIT_VECTOR(1 downto 0));
  end component;
  for L1: ONES_CNTA use entity ONES_CNT(ALGORITHMIC);
begin
  L1: ONES_CNTA port map(A, C);
  process
  variable VLINE: LINE;
  variable V1: BIT_VECTOR(2 downto 0);
  variable V2: BIT_VECTOR(1 downto 0);
  file INVECT: TEXT is "TVECT.TEXT";
begin
    PLAY<= '0', '1' after 3 ns;
    wait on PLAY until PLAY = '1';
    while not(ENDFILE(INVECT)) loop
      READLINE(INVECT, VLINE);
      READ(VLINE, V1); READ(VLINE, V2);
      A<= V1;
      wait for 1 ns;
      assert (V2 = C) report "WARNING: C is NOT equal to (V2 or C)" severity WARNING;
    end loop;
  end process;
end ONES_CNT1;