Introduction to VHDL
Lecture D

Prof. K. J. Hintz

Department of Electrical
and
Computer Engineering
George Mason University
Basic VHDL
RASSP Education & Facilitation
Module 10

Version 1.6

Copyright © 1995, 1996 RASSP E&F

All rights reserved. This information is copyrighted by the RASSP E&F Program and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the RASSP E&F Program is prohibited. All information contained herein may be duplicated for non-commercial educational use provided this copyright notice is included. No warranty of any kind is provided or implied, nor is any liability accepted regardless of use.

FEEDBACK:
The RASSP E&F Program welcomes and encourages any feedback that you may have including any changes that you may make to improve or update the material. You can contact us at feedback@rassp.scra.org or

http://rassp.scra.org/module-request/FEEDBACK/feedback-on-modules.html
VHDL uses a simulation cycle to model the stimulus and response nature of digital hardware.
All VHDL signal assignment statements prescribe an amount of time that must transpire before the signal assumes its new value.

This prescribed delay can be in one of three forms:

- **Transport** -- prescribes propagation delay only
- **Inertial** -- prescribes minimum input pulse width and propagation delay
- **Delta** -- the default if no delay time is explicitly specified
Transport Delay

λ Delay must be explicitly specified by user

µ Keyword “TRANSPORT” must be used

λ Signal will assume its new value after specified delay

```
-- TRANSPORT must be specified
Output <= TRANSPORT NOT Input AFTER 10 ns;
```
Inertial Delay

λ Provides for specification of input pulse width, i.e. ‘inertia’ of output, and propagation delay:

\[ \text{target} \leq \left[ \text{REJECT time_expression} \right] \text{INERTIAL waveform}; \]

λ Inertial delay is default and REJECT is optional:

\[ \text{Output} \leq \text{NOT Input AFTER 10 ns;} \]

--- Propagation delay and minimum pulse width are 10ns
Example of gate with ‘inertia’ smaller than propagation delay

e.g. Inverter with propagation delay of 10ns which suppresses pulses shorter than 5ns

Output <= REJECT 5ns INERTIAL NOT Input AFTER 10ns;

Note that _REJECT_ feature is new to VHDL 1076-1993
Delta Delay

- Default signal assignment propagation delay if no delay is explicitly prescribed
  - VHDL signals assignment cannot take place immediately
  - Delta is an infinitesimal VHDL time unit so that all signal assignments can result in signals assuming their values at some future time
  - E.g. `Output <= NOT Input;`  -- Output assumes new value in one delta cycle

- Supports a model of concurrent VHDL process execution
  - Order in which processes are executed by simulator does not affect simulation output
What is the behavior of C?

NAND gate evaluated first:
- IN: 1->0
- A: 0->1
- B: 1->0
- C: 0->0

AND gate evaluated first:
- IN: 1->0
- A: 0->1
- C: 0->1
- B: 1->0
- C: 1->0
Delta Delay
An Example with Delta Delay

What is the behavior of C?

\[ \text{IN: } 1 \rightarrow 0 \]

Using delta delay scheduling

<table>
<thead>
<tr>
<th>Time</th>
<th>Delta</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ns</td>
<td>1</td>
<td>( \text{IN: } 1 \rightarrow 0 ) eval INVERTER</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>( \text{A: } 0 \rightarrow 1 ) eval NAND, AND</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>( \text{B: } 1 \rightarrow 0 ) eval AND</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>( \text{C: } 1 \rightarrow 0 )</td>
</tr>
</tbody>
</table>

1 ns
**Signals and Variables**

This example highlights the difference between signals and variables.

**ARCHITECTURE test1 OF mux IS**
  **SIGNAL x : BIT := '1';**
  **SIGNAL y : BIT := '0';**
  BEGIN
    PROCESS (in_sig, x, y)
    BEGIN
      x <= in_sig XOR y;
      y <= in_sig XOR x;
    END PROCESS;
  END test1;

**ARCHITECTURE test2 OF mux IS**
  **SIGNAL y : BIT := '0';**
  BEGIN
    PROCESS (in_sig, y)
    VARIABLE x : BIT := '1';
    BEGIN
      x := in_sig XOR y;
      y <= in_sig XOR x;
    END PROCESS;
  END test2;

Assuming a 1 to 0 transition on `in_sig`, what are the resulting values for `y` in the both cases?
VHDL Objects
Signals vs Variables

A key difference between variables and signals is the assignment delay

```vhdl
ARCHITECTURE sig_ex OF test IS
    SIGNAL a, b, c, out_1, out_2 : BIT;
BEGIN
    PROCESS (a, b, c, out_1)
    BEGIN
        out_1 <= a NAND b;
        out_2 <= out_1 XOR c;
    END PROCESS;
END sig_ex;
```

<table>
<thead>
<tr>
<th>Time</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>out_1</th>
<th>out_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1+d</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1+2d</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
ARCHITECTURE var_ex OF test IS
  SIGNAL a, b, c, out_4 : BIT;
BEGIN
  PROCESS (a, b, c)
  VARIABLE out_3 : BIT;
  BEGIN
    out_3 := a NAND b;
    out_4 <= out_3 XOR c;
  END PROCESS;
END var_ex;

<table>
<thead>
<tr>
<th>Time</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>out_3</th>
<th>out_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1+d</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Simulation Cycle Revisited
Sequential vs Concurrent Statements

- VHDL is inherently a **concurrent language**
  - All VHDL **processes** execute concurrently
  - Concurrent signal assignment statements are actually **one-line processes**

- VHDL statements **execute sequentially within a process**

- Concurrent processes with **sequential execution within a process** offers **maximum flexibility**
  - Supports various levels of abstraction
  - Supports modeling of concurrent and sequential events as observed in real systems
Examples

- Build a library of logic gates
  - AND, OR, NAND, NOR, INV, etc.
- Include sequential elements
  - DFF, Register, etc.
- Include tri-state devices
- Use 4-valued logic
  - ‘X’, ‘0’, ‘1’, ‘Z’
  - Encapsulate global declarations in a package
PACKAGE resources IS

    TYPE level IS ('X', '0', '1', 'Z'); -- enumerated type

    TYPE level_vector IS ARRAY (NATURAL RANGE <>) OF level;
    -- type for vectors (buses)

    SUBTYPE delay IS TIME; -- subtype for gate delays

    -- Function and procedure declarations go here

END resources;
Two Input AND Gate Example

USE work.resources.all;

ENTITY and2 IS
   GENERIC(trise : delay := 10 ns;
            tfall : delay := 8 ns);
   PORT(a, b : IN level;
        c : OUT level);
END and2;

ARCHITECTURE behav OF and2 IS

   BEGIN
      one : PROCESS (a,b)
      BEGIN
         IF (a = '1' AND b = '1') THEN
            c <= '1' AFTER trise;
         ELSIF (a = '0' OR b = '0') THEN
            c <= '0' AFTER tfall;
         ELSE
            c<= 'X' AFTER (trise+tfall)/2;
         END IF;
      END PROCESS one;

   END behav;
And Gate Simulation Results
Tri-State Buffer Example

USE work.resources.all;

ENTITY tri_state IS

    GENERIC(trise : delay := 6 ns;
              tfall : delay := 5 ns;
              thiz : delay := 8 ns);

    PORT(a : IN level;
         e : IN level;
         b : OUT level);

END tri_state;

ARCHITECTURE behav OF tri_state IS

BEGIN

    one : PROCESS (a,e)

    BEGIN

        IF (e = '1' AND a = '1') THEN
            -- enabled and valid data
            b <= '1' AFTER trise;
        ELSIF (e = '1' AND a = '0') THEN
            b <= '0' AFTER tfall;
        ELSIF (e = '0') THEN
            -- disabled
            b <= 'Z' AFTER thiz;
        ELSE
            -- invalid data or enable
            b <= 'X' AFTER (trise+tfall)/2;
        END IF;

    END PROCESS one;

END behav;
Tri-State Buffer Simulation Results
D Flip Flop Example

USE work.resources.all;

ENTITY dff IS
    GENERIC(tprop : delay := 8 ns;
             tsu   : delay := 2 ns);
    PORT(d  : IN level;
          clk : IN level;
          enable : IN level;
          q  : OUT level;
          qn  : OUT level);
END dff;

ARCHITECTURE behav OF dff IS
    BEGIN
        one : PROCESS (clk)
        BEGIN
            -- check for rising clock edge
            IF ((clk = '1' AND clk'LAST_VALUE = '0')
                AND enable = '1') THEN -- ff enabled
                -- first, check setup time requirement
                IF (d'ABLE(tsu)) THEN
                    -- check valid input data
                    IF (d = '0') THEN
                        q <= '0' AFTER tprop;
                        qn <= '1' AFTER tprop;
                    ELSIF (d = '1') THEN
                        q <= '1' AFTER tprop;
                        qn <= '0' AFTER tprop;
                    ELSE -- else invalid data
                        q <= 'X';
                        qn <= 'X';
                    END IF;
                ELSE -- else violated setup time requirement
                    q <= 'X';
                    qn <= 'X';
                END IF;
            ELSE -- else violated setup time requirement
                q <= 'X';
                qn <= 'X';
            END IF;
        END IF;
    END PROCESS one;
END behav;
D Flip Flop Simulation Results