

ECE 368—Spring 2008, Instructor: Prof. Shantanu Dutt

Lab 3 : Due Thurs Feb. 12 (1st part), Feb. 19 (2nd part)

Objective

The objectives of this lab are: (1) Use a *hierarchical design methodology* to design a ripple-carry adder; (2) Learn and use **generic** and **generate** statements in your design.

In a hierarchical design methodology, the smallest components are designed first (e.g., logic gates) followed by the next bigger components that use the previous components (e.g., full adders [FAs] using logic gates), then the next bigger components (e.g., ripple-carry adders using FAs) and so on.

Relevant VHDL Background

The generic statement

The **generic** statement allows components to be parametrized, mostly with certain timing values like delays and size values like the size of an adder (8-bit, 16-bit, 32-bit, etc.) being designed. This allows for example, the same XOR gate description to be used in different designs but with different delay values based on the technological assumptions of the higher-level designs (e.g., does the XOR gate use transistors of size λ or 4λ , is the feature size 0.25 micron or 0.18 micron). The generic statement is part of the entity description of a module/component. An example is given below:

```
entity xor_gate is
generic (t_prop: Time := 6 ns); -- default delay initialization; can be changed during instantiation
port (x, y :in std_logic; z: out std_logic);
end entity xor_gate;
```

```
architecture behav of xor_gate is
begin
z <= (not(x) and y) or (x and not(y)) after t_prop;
end architecture behav;
```

```
:
entity design1 is
:
end entity design1;
```

```
-- in architecture of a higher level design
architecture struct of design1 is
```

```
component xor_gate is
generic (t_prop: Time := 6 ns); -- default delay initialization; can be changed during instantiation
port (x, y :in std_logic; z: out std_logic);
```

```

end component xor_gate;

-- binding indication
for all: xor_gate use entity work.xor_gate(behav);

signal a, b, c : std_logic;
:
begin

xor1: xor_gate generic map (8 ns) -- using an XOR gate but with a 8 ns delay
portmap (a, b, c);

:
end architecture struct;

```

The generate statement

From RASSP slides (slide 22 at www.cedcc.psu.edu/ee497/frassp_11sld022.htm) there is the following succinct description of the **generate statement**.

“Structural descriptions of large, but highly regular, structures can be tedious. A VHDL GENERATE statement can be used to include as many concurrent VHDL statements (e.g. component instantiation statements) as needed to describe a regular structure easily. In fact, a GENERATE statement may even include other GENERATE statements for more complex devices.. Some common examples include the instantiation and connection of multiple identical components such as half adders to make up a full adder, or exclusive or gates to create a parity tree.”

Example of a generate statement for performing 8-bit bitwise XOR:

```

architecture struct of xor_array is

component xor_gate is
generic (t_prop: Time := 6 ns); -- default delay initialization; can be change d during instantiation
port (x, y :in std_logic; z: out std_logic);
end component xor_gate;

-- binding indication
for all: xor_gate use entity work.xor_gate(behav);

signal A, B, C std_logic_vector(0 to 7);

begin
gen1: for i in 0 to 7 generate
array: xor_gate
generic map (4 ns)
port map (A(i), B(i), C(i));

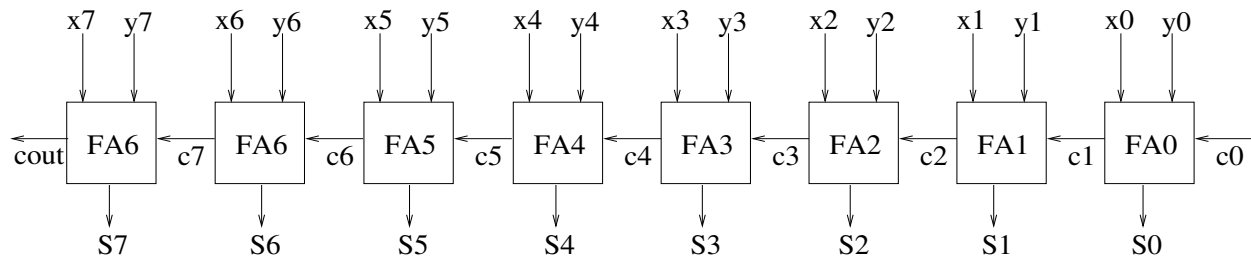
```

```
end generate gen1;
end architecture struct;
```

Problem Statement

To design an n -bit ripple-carry adder structurally and hierarchically in VHDL.

A ripple-carry adder (RCA) is shown below.



Assume that the delay of a k -input AND/OR/NOT gate is $2k$ ns, and that of a k -input XOR/XNOR gate is $3k$ ns.

Part 1

Design Problem

- (1) Define a full adder (FA) structurally in terms of AND/OR/NOT/XOR/XNOR gates, and whatever gates you use you can define behaviorally. Also, describe the gates using the **generic** statement, where the gate delay is the **generic** statement's parameter; that way if the gate delay assumption changes for a different system/circuit in which it is used, you can still use the same gate description but use a different delay value in the **generic map** statement when instantiating the gate in the containing module. **100**
- (2) Calculate the delay of your design/description of the FA. This is the *analytical* delay. **20**

Testbench

Based on the testbench of Lab 2, design your own test bench to exhaustively test (i.e., by giving as inputs **all** input vectors (i.e., input combinations)) the full adder that you've described in VHDL. Instantiate the FA in the test bench, and clearly show correctness of the output result and also the worst-case *simulation* delay of the two output signals.

Definition: A *simulation delay* of an output Z for some input vector I_j , is the delay between the appearance of I_j and the new value of Z , if this new value is different from its previous value (if the new value of Z is the same as its previous value, then this simulation does not contribute to delay estimation)

The worst-case delay across all inputs and outputs that you observe is the *simulation delay* of the FA.

Compare the analytical and simulation delay of the FA.

100

Part 2

Design Problem

- (1) Give a VHDL structural description of an n -bit RCA, using the **generic** and **generate** statements and using full adders as components. The adder size n should be a generic parameter. **100**
- (2) Calculate the delay of your design/description of the n -bit RCA. This is the *analytical* delay. **50**

Test Bench

- (1) Instantiate a 16-bit RCA in a test bench (that will be provided to you) that provides 100 random test inputs, and clearly show correctness of the output result and also the worst-case *simulation* delay of the 4 output signals you think should have the highest delay over all 100 inputs. This is the *simulation delay* of the RCA. **200**
- (2) Compare the analytical and simulation delays you have obtained for your RCA and comment on any discrepancy giving possible reasons for it. **50**