Lab 4: VHDL Test Bench

TA: Andrew White

Sept 22, Sept 24, Sept 29

Overview

This lab is an introduction VHDL test benches and elaborates further on structural design.

VHDL Test Bench

A VHDL test bench is VHDL code that wraps the VHDL design (referred to as the DUT, the Design Under Test) to be simulated. A VHDL test bench produces input test stimuli for the DUT and can verify the correctness of the design. An example of a simple test bench that only supplies inputs is attached. For information about verifying outputs, lookup VHDL **assert** statements.

More Structural VHDL

Generate Statements

Generate statements provide a convenient way to create multiple instances of concurrent statements, with the most typical use being component instantiation statements. These DO NOT go inside a process as they are concurrent statements and are used in a manner similar to component instantiation statements: inside the architecture body.

Looping Generate Statements: A for-loop style generate statement, useful for connecting repetitive component structures.

Syntax:

```
-- Label is REQUIRED
Label: for i in discrete_range generate
        {concurrent statement}
end generate [label];
```

ex.

```
and_gen : for i in 0 to 7 generate
        -- Instantiates 8 AND gates with inputs/outputs A,B,Z: 8 bit arrays
        AND_I: component c_and
                port map ( input1 => A(i),
                           input2 => B(i),
                           output => Z(i));
end generate;
```

Conditional Generate Statements: A conditionally executed generate statement that controls the execution of the generate statement using an if-conditional style statement. The condition evaluated must be fixed at run-time (ex. the conditional generate statement can be nested inside a looping generate statement and use the looping variable as the condition).

Syntax:

```
        -- Label required
        Label: if boolean_expression generate
                        {concurrent statement}
                end generate [label];
```

Lab Procedure

For this lab, use the attached full adder slice, the inverter from lab 3, and generate statements to create an 8-bit 2’s compliment subtractor. The subtractor should have a 8-bit output. Hint: use linear arrays (type std_logic_vector) and looping generate statements to connect the slices, with conditional generate statements to handle the unique case of the carry input to the LSB slice.

Once the design has been created, write a test bench that gives 6 pairs of inputs (try both positive and negative numbers). Simulate the design by simulating the test bench in Modelsim. To do this, import the test bench and all other VHDL used for the design files into a project in Modelsim (using the same method as in previous labs), select the test bench VHDL source file and click **Compile -> Compile All**. Then, in the **Library** tab, right click the test bench under the *work* library and select **Simulate**. The most important step is to select the test bench VHDL file before compiling. This should be done for all structural, multiple-file designs.

Lab Report

For the lab report, submit a cover sheet, all VHDL source files used (including the test bench) and simulation waveforms. Mark the waveforms with the integer equivalent values for all input combinations and the resulting outputs to verify correctness of design.

```
-- Test bench for c_and.vhd
library IEEE;
use ieee.std_logic_1164.ALL;

library work;
use work.all;


entity test_c_and is
        -- No Port Interface used
end test_c_and;

architecture test_bench of test_c_and is

        -- Declare the Design Under Test
        component c_and
          port (input1 : std_logic_vector(3  downto 0);
                input2 : std_logic_vector(3  downto 0);
                output : out std_logic_vector(3 downto 0));
        end component;

        -- Declare signals to be used for assigning input stimuli and view
        -- the output for simulation
        signal inputA : std_logic_vector(3 downto 0);
        signal inputB : std_logic_vector(3 downto 0);
        signal outZ :    std_logic_vector(3 downto 0);

begin

        -- Instantiate the Design Under Test
        and4 : c_and
                port map( input1 => inputA,
                        input2 => inputB,
                        output => outZ);

        test_process : process
        begin

                -- First Give all inputs an initial value
                inputA <= "0001" ;
                inputB <= "0001" ;

                -- Seperate Input test stimuli with
                -- a "wait for X" statement to allow input
                -- values to be assigned and output given in
                -- Simulation
                wait for 10 ns;
```

```
			-- InputB's previous value is retained
			inputA <= "1100" ;
			wait for 10 ns;

			-- InputA's previous value is retained
			inputB <= "0110" ;
			wait for 10 ns;

			inputA <= "1111" ;
			inputB <= "0101" ;
			wait for 10 ns;

			inputA <= "0000" ;
			inputB <= "0000" ;
			wait for 10 ns ;

			inputA <= "0101" ;
			wait for 10 ns;

			inputB <= "1111" ;
			wait for 10 ns;

			inputA <= "1111" ;
			wait for 10 ns;

			inputA <= "0000" ;
			wait for 10 ns;

			-- This wait statement halts the simulation
		    wait;


		end process test_process ;

end test_bench;
```

```
-- c_and.vhd
LIBRARY ieee;
  use ieee.std_logic_1164.ALL;

entity c_and is
  port (input1 : std_logic_vector(3  downto 0);
        input2 : std_logic_vector(3  downto 0);
        output : out std_logic_vector(3 downto 0));
end  c_and;

architecture behavior of c_and is
begin
             P0 : process(input1,input2)
          variable  result : std_logic_vector(3 downto 0);
        begin
          outer :
           for n in 3 downto 0 loop
             result(n) := input1(n) and input2(n);
          end loop outer;
          output <= result;
   end process P0;

end behavior;
```

```
-- add_slice.vhd
-- Full Adder slice
-- Example for CDA 4203 Lab 4

library ieee;
use ieee.std_logic_1164.all;

entity add_slice is
        port ( input1, input2, carry_in : in std_logic;
                carry_out, output : out std_logic);
end add_slice;

architecture behav_example of add_slice is
begin

        process(input1,input2,carry_in)
        begin
            output <= (input1 XOR input2) XOR carry_in ;
            carry_out <= (input1 AND input2) OR (carry_in AND (input1 XOR input2));
        end process;

end behav_example;
```