Advanced Digital Circuit Design - Test Benches for Combinational Circuits

Prof. Dr. Berna Örs Yalçın

Istanbul Technical University
Faculty of Electrical and Electronics Engineering
Department of Electronics and Communication Engineering
siddika.ors@itu.edu.tr

Test Benches

- A test bench is an HDL program for applying stimulus to an HDL model and observe its response.
- A good test bench should be written to exercise as many functional features of a circuit as possible.
- **process begin** ... **end process;** phrase is used to provide stimulus to a circuit to be tested.
- An **process begin** ... **end process;** phrase is executed only once, starting at time 0.
- An **process begin** ... **end process;** block includes operations that are delayed by a given number of time units.

The **process** Block

```
process
begin
        A = 0; B=0;
        wait for 10 ns;
        A=1;
         wait for 20 ns;
        A=0; B=1;
end process;
```

Stimulus for Generation for Combinational Circuits

```
process
begin
        D = ''000'';
        for i in 1 to 7 loop
                wait for 10 ns;
                D = D + ''001'';
        end loop;
end process;
```

The above code can be used to generate stimulus for a 3 input combinational circuit and obtain its truth table.

Writing a Test Bench for an Entity under Test

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
use ieee.std_logic_arith.all;
...
ENTITY entity_name_tb IS
END entity_name_tb;
ARCHITECTURE behavioral OF entity_name_tb IS
      // Declare the design entity under test.
      COMPONENT entity_name PORT( ...);
      END COMPONENT;

      // Declare local signals
      SIGNAL input ports
      SIGNAL output ports
      CONSTANT period : time := 10 ns;
BEGIN
      // Instantiate the design module under test.
      UUT: entity_name PORT MAP(...);
      // Generate stimulus
      stim_proc: process BEGIN
            input signal  <= value;
            WAIT FOR period;
            ....
            WAIT;
      end process;
END;
```

- A test bench entity is written as any other VHDL entity but it has no inputs or outputs.
- The signals that are applied as inputs to the entity under test are declared as **signal** data type.
- The outputs of the entity under test are declared in the test module as **signal** data type.

Interaction Between Stimulus and Design Entities

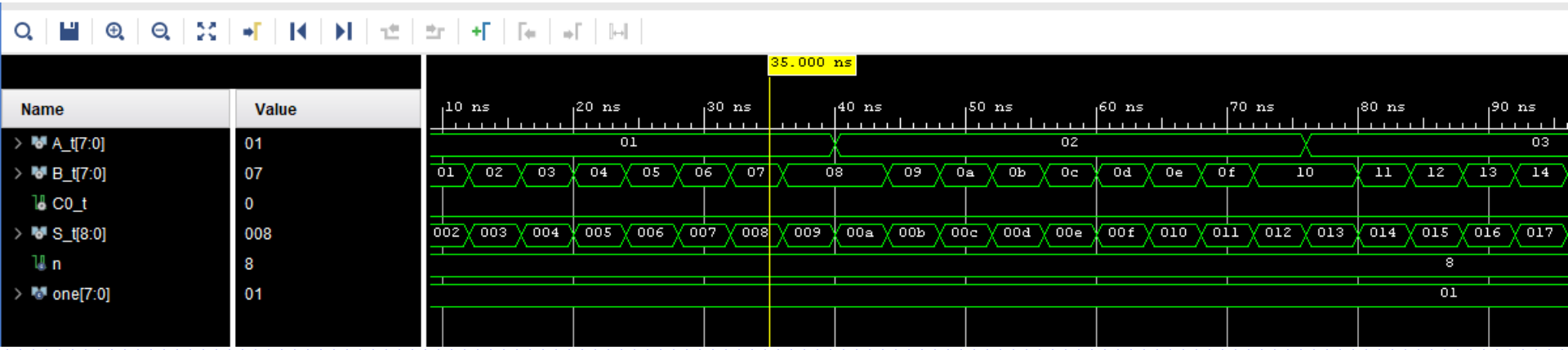
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ripple_carry_adder is
    generic (n : integer := 8);
    Port ( A : in std_logic_vector (n-1 downto 0);
           B : in std_logic_vector (n-1 downto 0);
           C0 : in std_logic;
           S : out std_logic_vector (n downto 0));
end ripple_carry_adder;

architecture Gate_Level of ripple_carry_adder is
    component full_adder is
        Port ( x : in std_logıc;
               y : in std_logic;
               z : in std_logic;
               S : out std_logic;
               C : out std_logic);
    end component;
    signal C : std_logic_vector (n downto 0);
begin
    C(0) <= C0;
    GEN_FA: for i in 0 to n-1 generate
        FA: full_adder Port map(A(i),B(i),C(i),S(i),C(i+1));
    end generate GEN_FA;
    S(n) <= C(n);
end Gate_Level;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ripple_carry_adder_tb is
    generic (n : integer := 8);
end ripple_carry_adder_tb;
architecture Behavioral of ripple_carry_adder_tb is
    component ripple_carry_adder is
        generic (n : integer := 8);
        Port ( A : in std_logic_vector (n-1 downto 0);
               B : in std_logic_vector (n-1 downto 0);
               C0 : in std_logic;
               S : out std_logic_vector (n downto 0));
    end component;
    signal A_t,B_t : std_logic_vector (n-1 downto 0);
    signal C0_t : std_logic;
    signal S_t : std_logic_vector (n downto 0);
    constant one : std_logic_vector (n-1 downto 0) := "00000001";
begin
    DUT: ripple_carry_adder generic map(n)
                        Port map (A_t,B_t,C0_t,S_t);
    process    begin
        A_t <= "00000000";        B_t <= "00000000";
        C0_t <= '0';
        for i in 0 to n-1 loop
            wait for 4 ns;              A_t <= A_t + one;
            for j in 0 to n-1 loop
                wait for 4 ns;                  B_t <= B_t + one;
            end loop;
        end loop;
    end process;
end Behavioral;
```

Simulation Results for the 4-bit Adder Entity



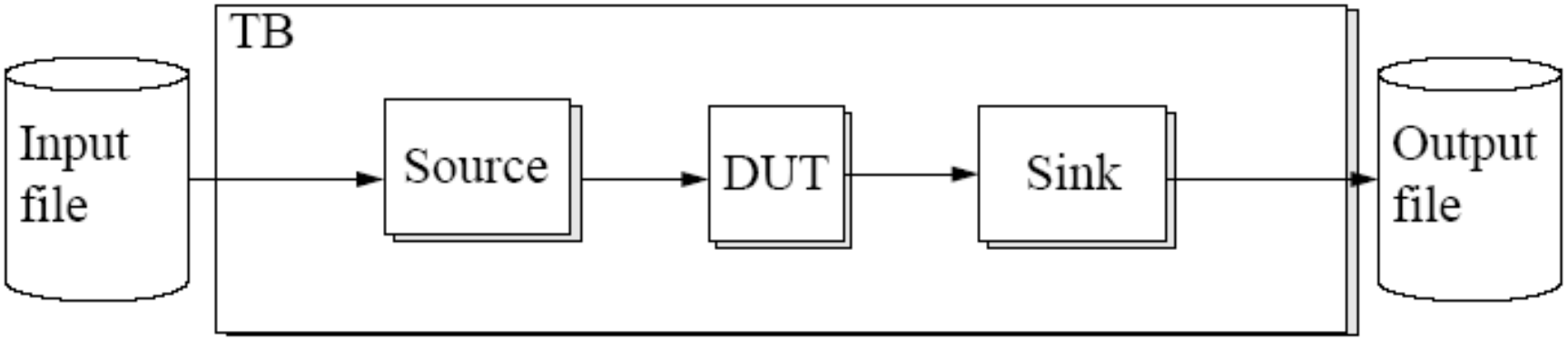
Check results with "assertions"

```
process
   begin
      A_t <= "00000000";       B_t <= "00000000";       C0_t <= '0';
      for i in 0 to n-1 loop
         wait for 4 ns;          A_t <= A_t + one;
         for j in 0 to n-1 loop
            wait for 4 ns;             B_t <= B_t + one;
            assert (S_t = A_t + B_t) report "Error message" severity NOTE;
         end loop;
      end loop;
   end process;
```

- Match data types for S_t, A_t, B_t
- Print "Error message" if assert condition FALSE
  - (condition is not what we expected)
- Specify one of four severity levels:
  - NOTE, WARNING, ERROR, FAILURE
- Simulator allows selection of severity level to halt simulation
  - ERROR generally should stop simulation
  - NOTE generally should not stop simulation

Test Bench with Text-IO

- Stimulus for DUT is read from an input file and modified in the source component
- The response modified is in the sink and written to the output file



Libraries, remember to declare the textio-library!

```
library IEEE;
use IEEE.std_logic_1164.all;
use std.textio.all;
use IEEE.std_logic_textio.all;
```

Test Bench with Text-IO

- Create process and declare the input and output files (VHDL'87)
  FILE file_in : TEXT IS IN *"datain.txt"*;
  FILE file_out : TEXT IS OUT *"dataout.txt"*;
  - File paths are relative to *simulation directory*
- Variables for one line of the input and output files
  VARIABLE line_in : LINE;
  VARIABLE line_out : LINE;
  - Value of variable is updated immediately. Hence, the new value is visible on the same execution of the process (already on the next line)
- Variables for the value in one line
  VARIABLE input_tmp : INTEGER;
  VARIABLE output_tmp : INTEGER;

```
process
      FILE file_in : TEXT IS IN "datain.txt";
      FILE file_out : TEXT IS OUT "dataout.txt";
      VARIABLE line_in : LINE;
      VARIABLE line_out : LINE;
      VARIABLE input_tmp : std_logic_vector(7 downto 0);
      VARIABLE output_tmp : std_logic_vector(8 downto 0);
   begin
   while not endfile(file_in) loop
         readline(file_in, line_in);
         read(line_in, input_tmp);
         A_t<= input_tmp;
         read(line_in, input_tmp);
         B_t<= input_tmp;
         wait for 4 ns;
         write(line_out, S_t);
         writeline(file_out,line_out);
         writeline(OUTPUT,line_out);
      end loop;
      wait;
   end process;
end Behavioral;
```

datain.txt

```
00000000 00000000
00000000 00000001
00000000 00000010
00000000 00000011
00000000 00000100
```

dataout.txt

```
000000000
000000001
000000010
000000011
000000100
```