

Using *Test Benches* for Digital Systems V&V

S. Chiusano S. Di Carlo G. Di Natale

Politecnico di Torino

Silvia.Chiusano@polito.it

dicarlo@polito.it

dinatale@polito.it

Index

1.	Application to combinational circuits.....	2
1.1.	The case study: a comparator (Example #6.3.4).....	2
1.2.	Test bench	4
1.2.1.	Basic Structure	4
1.2.2.	Test Bench: Sol. #1	5
1.2.3.	Test Bench: Sol. #2	8
1.2.4.	Test Bench: Sol. #3	11
1.2.4.1.	Package my_function.....	15
1.3.	Simulation Steps for VHDL code validation.....	17
1.3.1.	Using the Test bench Sol.#1	17
1.3.2.	Using the Test bench Sol.#2	17
1.3.3.	Using the Test bench Sol.#3	18
2.	Extension to sequential circuits.....	19
2.1.	I/O signals in sequential circuits	19
2.2.	Example	19
2.3.	Test Bench	19
2.3.1.	Basic Structure	19
2.3.2.	Remarks	22
2.4.	Simulation Steps for VHDL code validation.....	22

1. Application to combinational circuits

1.1. The case study: a comparator (Example #6.3.4)

A comparator gets 2 n -bit binary numbers, A and B , and provides in output the result of the comparison between A and B . A control input $UM/\sim 2C$ specifies whether the input operands are unsigned or signed, respectively. The result of the comparison is displayed using the following output signals:

- $AgtB$, asserted iff $A > B$
- $AltB$, asserted iff $A < B$
- $AeqB$, asserted iff $A = B$
- $Al e B$, asserted iff $A \leq B$
- $AneB$, asserted iff $A \neq B$
- $AgeB$, asserted iff $A \geq B$.

A possible VHDL description of the comparator is the following one:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity comparator is
  generic (inp_size: integer:=16);
  port(
    A, B : in std_logic_vector(inp_size-1 downto 0);
    UM_n2C : in std_logic ;
    AgtB, AltB, AeqB, AleB, AneB, AgeB : out std_logic) ;
end comparator;

architecture beh of comparator is
begin
  process(A,B, UM_n2C)
    variable AeqB_v, AgtB_v, AltB_v: std_logic;
  begin
    if A=B then
      AeqB_v := '1';
      AgtB_v := '0';
      AltB_v := '0';
    else
      AeqB_v := '0';
      if (UM_n2C = '0') then -- unsigned operands
        if (unsigned(A)> unsigned(B)) then
          AgtB_v := '1';
          AltB_v := '0';
        else
          AgtB_v := '0';
          AltB_v := '1';
        end if;
      else -- signed operands
        if (signed(A) > signed(B)) then
          AgtB_v := '1';
          AltB_v := '0';
        else
          AgtB_v := '0';
          AltB_v := '1';
        end if;
      end if;
    end if;
    AleB <= AeqB_v or AltB_v;
    AneB <= not AeqB_v;
    AgeB <= AeqB_v or AgtB_v;
    AeqB <= AeqB_v;
    AgtB <= AgtB_v;
    AltB <= AltB_v;
  end process;
end beh;

```

1.2. Test bench

1.2.1. Basic Structure

The test bench mainly contains:

- The instantiation of the component to be simulated (in this example, the comparator);
- One process (in the example, STIMULUS1), to specify the input values for the component, and possibly check the correctness of the generated output values.

```

library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;

entity TESTBENCH is
end TESTBENCH;

architecture stimulus of TESTBENCH is
component COMPARATOR is
    generic (inp_size: integer := 16);
    port (
        A, B : in unsigned(inp_size downto 0);
        UM_n2C : in std_logic ;
        AgtB, AltB, AeqB, AleB, AneB, AgeB : out std_logic
    );
end component;

-- Top level signals go here...
signal A, B : unsigned(inp_size downto 0);
signal UM_n2C : std_logic ;
signal AgtB, AltB, AeqB, AleB, AneB, AgeB : std_logic;

begin
    DUT: COMPARATOR generic map (16)
        port map ( A, B, UM_n2C , AgtB, AltB, AeqB, AleB, AneB,
AgeB );

    STIMULUS1: process
    begin

        wait;          -- Suspend simulation
    end process STIMULUS1;

end stimulus;

```

In the following, three different possible implementations of the test bench process (STIMULUS1) will be presented.

1.2.2. Test Bench: Sol. #1

- The input values are explicitly specified within the test bench process (STIMULUS1);
- The correctness of the output values is evaluated analyzing the output waveforms displayed using the tool VWaves.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use std.textio.all;

entity testbench is
end testbench;

architecture stimulus of testbench is
component COMPARATOR
  generic (inp_size: integer);
  port (
    A, B : in std_logic_vector(inp_size downto 0);
    UM_n2C : in std_logic ;
    AgtB, AltB, AeqB, AleB, AneB, AgeB : out std_logic
  );
end component;

constant N: integer := 16;

-- Top level signals go here...
signal A, B : std_logic_vector(N-1 downto 0);
signal UM_n2C : std_logic ;
signal AgtB, AltB, AeqB, AleB, AneB, AgeB : std_logic;

begin
  DUT: COMPARATOR generic map (N)
    port map ( A, B, UM_n2C , AgtB, AltB, AeqB, AleB, AneB, AgeB
);

  STIMULUS1: process
    constant period: time := 20 ns;
    begin

----consider the case of unsigned inputs
    UM_n2c <= '0';

----fix A to zero and assign different values to B
    A <= conv_std_logic_vector(0, N);
    B <= conv_std_logic_vector(0, N);
    wait for period;
    B <= conv_std_logic_vector(1, N);
    wait for period;
    B <= conv_std_logic_vector(2**(N-1), N);
                                --middle      range      value
    ((2**N)/2)
    wait for period;
    B <= conv_std_logic_vector((2**N)-1, N); --max number
    wait for period;

----fix A to max number and assign different values to B
    A <= conv_std_logic_vector((2**N)-1, N); --max number
    B <= conv_std_logic_vector((2**N)-1, N); --max number
    wait for period;

```

```

    B <= conv_std_logic_vector((2**N)-2, N); --max number -1
    wait for period;
    B <= conv_std_logic_vector(0, N);
    wait for period;

    ----fix A to middle range value and assign different values to B
    A <= conv_std_logic_vector(2**(N-1), N);
        --middle range value ((2**N)/2)
    B <= conv_std_logic_vector(2**(N-1), N);
        --middle range value ((2**N)/2)
    wait for period;
    B <= conv_std_logic_vector((2**(N-1))+1, N);
        --middle range value+1
    wait for period;
    B <= conv_std_logic_vector((2**(N-1))-1, N);
        --middle range value-1
    wait for period;
    B <= conv_std_logic_vector((2**N)-1, N);--max number
    wait for period;
    B <= conv_std_logic_vector(0, N);
    wait for period;

    ----consider the case of unsigned inputs
    UM_n2c <= '1';

    ----assign different positive/negative values to A and B
    A <= conv_std_logic_vector(0, N);
    B <= conv_std_logic_vector(0, N);
    wait for period;

    B <= conv_std_logic_vector(-1, N);
    wait for period;
    B <= conv_std_logic_vector(+1, N);
    wait for period;

    B <= conv_std_logic_vector(2**(N-1)-1, N); --max positive num-
ber
    wait for period;
    B <= conv_std_logic_vector(-(2**(N-1)), N);---max negative num-
ber
    wait for period;

    A <= conv_std_logic_vector(-1, N);
    B <= conv_std_logic_vector(-2, N);
    wait for period;
    A <= conv_std_logic_vector(-2, N);
    B <= conv_std_logic_vector(-1, N);

    wait;          -- Suspend simulation
    end process STIMULUS1;

end stimulus;

```

1.2.3. Test Bench: Sol. #2

- The input values and the corresponding expected outputs are stored inside the test bench, using an array;
- The test bench process (STIMULUS1) reads from the array the input values, and applies them to the comparator;
- The correctness of the output values is evaluated inside the test bench process. Using an assert statement, the output values generated by the comparator are compared with the expected one, stored into the array;
- According to the severity level (NOTE, WARNING, ERROR, FAILURE) of the assertion, when an error is encountered the simulation can be interrupted.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use std.textio.all;

entity testbench is
end testbench;

architecture stimulus_array of testbench is
component COMPARATOR
  generic (inp_size: integer);
  port (
    A, B : in std_logic_vector(inp_size downto 0);
    UM_n2C : in std_logic ;
    AgtB, AltB, AeqB, AleB, AneB, AgeB : out std_logic
  );
end component;

----A record is defined, containing all the input/output
----signals of the component to be verified

type test_record is record
  UM_n2C: std_logic;
  A: integer;
  B: integer;
  OUTPUT: std_logic_vector (5 downto 0);
  --AgtB, AltB, AeqB, AleB, AneB, AgeB
end record;

---- An array of records is defined
type test_array is array(positive range <>) of test_record;

----The array stores the input values and the corresponding ex-
pected output values
constant test_vectors: test_array := (
  ('0',0,0,"001101"),
  ('0',0,1,"010110"),
  ('0',0,32768,"010110"),
  ('0',0,65535,"010110"),

  ('0',65535,65535,"001101"),
  ('0',65535,65534,"100011"),
  ('0',65535,0,"100011"),

  ('0',32768,32768,"001101"),
  ('0',32768,32769,"010110"),
  ('0',32768,32767,"100011"),
  ('0',32768,65535,"010110"),
  ('0',32768,0,"100011"),

  ('1',0,0,"001101"),
  ('1',0,1,"010110"),
  ('1',0,-1,"100011"),
  ('1',0,32767,"010110"),

```

```

        ('1',0,-32768,"100011"),
        ('1',-1,-2,"100011"),
        ('1',-2,-1,"010110")
    );
    -- Top level signals go here...
    signal A, B : std_logic_vector(15 downto 0);
    signal UM_n2C : std_logic ;
    signal AgtB, AltB, AeqB, AleB, AneB, AgeB : std_logic;

    begin
        DUT: COMPARATOR generic map (16)
            port map ( A, B, UM_n2C , AgtB, AltB, AeqB, AleB, AneB, AgeB
        );

        STIMULUS1: process
            constant period: time := 20 ns;
            variable output: std_logic_vector(5 downto 0);
            begin
                for index in test_vectors'range loop
                    ----The input values are applied to the component
                    A <= conv_std_logic_vector (test_vectors(index).A,16);
                    B <= conv_std_logic_vector (test_vectors(index).B,16);
                    UM_n2C <= test_vectors(index).UM_n2C;
                    wait for period;

                    ----The output values generated by the component are
                    ----compared with the expected values
                    output := AgtB & AltB & AeqB & AleB & AneB & AgeB;
                    assert (output = test_vectors(index).output)
                        report "Output does not match!"
                        severity FAILURE;

                end loop;
                wait; -- Suspend simulation
            end process STIMULUS1;

        end stimulus_array;
    
```

1.2.4. Test Bench: Sol. #3

- The input values and the corresponding expected outputs are stored into two distinct files;
- First, the test bench process (STIMULUS1) reads, from the input file `input.txt`, the input values, and applies them to the comparator;
- Then, the test bench process (STIMULUS1) writes, on the output file `output.txt`, the output values generated by the comparator;
- The correctness of the output values is evaluated externally to the test bench process: the generated output file is compared with a file of expected output values, defined by the user.

```

use ieee.std_logic_1164.all;
library std;
use std.textio.all;
library work;
use work.my_function.all;

entity TESTBENCH is
end TESTBENCH;

architecture stimulus_file of TESTBENCH is
component COMPARATOR
generic (inp_size: integer);
port (
    A, B : in std_logic_vector(inp_size downto 0);
    UM_n2C : in std_logic ;
    AgtB, AltB, AeqB, AleB, AneB, AgeB : out std_logic
);
end component;

constant PERIOD: time := 20 ns;

-- Top level signals go here...
signal A, B : std_logic_vector(15 downto 0);
signal UM_n2C : std_logic ;
signal AgtB, AltB, AeqB, AleB, AneB, AgeB : std_logic;

begin
    DUT: COMPARATOR generic map (16)
        port map ( A, B, UM_n2C , AgtB, AltB, AeqB, AleB, AneB,
AgeB );

    STIMULUS1: process
        -- For PeakVHDL:
        -- File infile: text;
        -- File outfile: text;

        ---- An input file is declared for reading the input values
        File infile : text is in "input.txt";

        ----An output file is declared for writing the output values
        File outfile: Text is out "output.txt";

        ----Temporal variables for converting single characters and string
of characters read from
        ----the input file to std_logic and std_logic_vector values
        variable i_A, i_B : String (1 to 16);
        variable i_UM_n2C : Character ;

        ----A "line" data type is declared from reading/writing data
from/on the files
        variable ptr,ptrout : line;
        variable space : character;

    begin
        --For PeakVHDL:

```

```
--file_open (infile,"in",READ_MODE);
--file_open (outfile,"out",WRITE_MODE);
while not (endfile(infile)) loop

----A line is read from the input file
    readline (infile, ptr);

----From the line the values i_UM_n2C, i_A, i_B are read
----In this case the values are interlieved by a white space char-
acter
    read (ptr, i_UM_n2C);
    read (ptr, space);
    read (ptr, i_A);
    read (ptr, space);
    read (ptr, i_B);

----The read values are converted into std_logic and
std_logic_vector
----data types and they are applied as inputs for the comparator

    A <= to_std_logic_vector (i_A);
    B <= to_std_logic_vector (i_B);
    UM_n2C <= to_std_logic (i_UM_n2C);
    wait for PERIOD;

----The outputs generated by the comparator are first
----converted into characters and then written on the file
    write (ptrout, to_char(AgtB));
    write (ptrout, to_char(AltB));
    write (ptrout, to_char(AeqB));
    write (ptrout, to_char(AleB));
    write (ptrout, to_char(AneB));
    write (ptrout, to_char(AgeB));
    writeline (outfile, ptrout);

    end loop;
    wait;
end process stimulus1;
end stimulus_file;
```

In this specific case, the input file `input.txt` is the following one:

```
0 0000000000000000 0000000000000000
0 0000000000000000 0000000000000001
0 0000000000000000 1000000000000000
0 0000000000000000 1111111111111111
0 1111111111111111 1111111111111111
0 1111111111111111 1111111111111110
0 1111111111111111 0000000000000000
0 1000000000000000 1000000000000000
0 1000000000000000 1000000000000001
0 1000000000000000 0111111111111111
0 1000000000000000 1111111111111111
0 1000000000000000 0000000000000000
1 0000000000000000 0000000000000000
1 0000000000000000 0000000000000001
1 0000000000000000 1111111111111111
1 0000000000000000 0111111111111111
1 0000000000000000 1000000000000000
1 1111111111111111 1111111111111110
1 1111111111111110 1111111111111111
```

The generated output file `output.txt` is instead the following one:

```
001101
010110
010110
010110
001101
100011
100011
001101
010110
100011
010110
100011
001101
010110
100011
010110
100011
100011
010110
```

The functions for the conversion of data types used inside the test bench are not standard functions. They are described into the package `my_function`, and are the following ones:

```
function to_char(A: std_logic) return Character;
function to_string(A: std_logic_vector) return String;
function to_std_logic (B: character) return std_logic;
function to_std_logic_vector (B: String) return
std_logic_vector;
```

1.2.4.1. Package my_function

```

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

package my_function is
    function to_char(A: std_logic) return Character;
    function to_string(A: std_logic_vector) return String;
    function to_std_logic (B: character) return std_logic;
    function to_std_logic_vector (B: String) return
std_logic_vector;
end my_function;

package body my_function is
    function to_char(A: std_logic) return Character is
begin
    case A is
        when '0' | 'L' => return '0';
        when '1' | 'H' => return '1';
        when 'Z' => return 'Z';
        when OTHERS => return 'X';
    end case;
end;

    function to_string(A: std_logic_vector) return String is
    variable stmp: string(A'left+1 downto 1);
begin
    for i in A'reverse_range loop
        if A(i)='1' then
            stmp(i+1) := '1';
        elsif A(i)='0' then
            stmp(i+1) := '0';
        else
            stmp(i+1) := 'X';
        end if;
    end loop;
    return stmp;
end;

    function to_std_logic (B: character) return std_logic is
begin
    case B is
        when '0' => return '0';
        when '1' => return '1';
        when OTHERS => return 'X';
    end case;
end;

    function to_std_logic_vector (B: string) return
std_logic_vector is
    variable res: std_logic_vector (B'range);
begin
    for i in B'range loop
        case B(i) is
            when '0' => res(i) := '0';
            when '1' => res(i) := '1';

```

```
                when OTHERS => res(i) := 'X';
                end case;
            end loop;
            return res;
        end;
    end my_function;
```

1.3. Simulation Steps for VHDL code validation

The simulation steps to validate the VHDL system description can be mainly summarized as follows:

1. Identify the system functionalities to be verified. For each of them identify the appropriate input values to be applied to the system, and the expected output values;
2. Write the corresponding Test bench;
3. Run the whole simulation;
4. Verify if the output values are the expected ones;
5. If they are not, identify the first occurrence of input values that generates wrong output values;
6. Run again the simulation, suspending it immediately before the application of that input value. The following methods can be adopted to stop the simulation in the appropriate instant of time:
 - a. insert a breakpoint on the VHDL statement where that input value is assigned (applicable only in the case of Test bench #1, where the input values are explicitly defined inside the Test bench);
 - b. run the simulation until the time instant before the application of that input value;
 - c. insert a breakpoint on the input signal. As a condition to suspend the simulation, specify the input value preceding the one that causes the wrong output. In this way, the simulation will be interrupted immediately before processing the interested input value.
7. Go on with the simulation, proceeding step by step on the VHDL code. The error will be localized analyzing the assignment performed on variables and signals.

The three Test Benches mainly differ in the way to identify the input value that causes the generation of the wrong output value.

1.3.1. Using the Test bench Sol.#1

In this case the input value that generates a wrong output value is identified analyzing the output waveforms inside Vwaves.

1.3.2. Using the Test bench Sol.#2

In this case the input value that generates a wrong output value is identified by means of assert statements on the output values inserted into the Test bench.

1.3.3. Using the Test bench Sol.#3

In this case the input value that generates a wrong output value is identified comparing the file of expected outputs and the file of output values generated during the simulation.

2. Extension to sequential circuits

2.1. I/O signals in sequential circuits

The sequential circuits include the following input and output signals:

- a **clock signal**;
- an **asynchronous reset signal**;
- the remaining signals include input and output **data** signals, and possible input **control** signals and output **status** signals.

2.2. Example

Let consider as an example a sequential component having the following entity

```
entity conv is
  port (
    clock, reset : in std_logic;
    enable       : in std_logic;
    din          : in std_logic;
    dout         : out std_logic;
    err          : out std_logic);
end conv;
```

In the example:

- `clock` is the **clock signal**;
- `reset` is the **asynchronous reset signal**;
- `din` is the input **data** signals, `dout` the output data signals, `enable` the **control signal** and `err` the **status signal**.

2.3. Test Bench

2.3.1. Basic Structure

The test bench mainly contains:

1. The **instantiation** of the component to be simulated;
2. One process generating the **clock signal**;
3. One process generating the **asynchronous reset signal**;

4. One process specifying the remaining **inputs** for the component, and possibly checking the correctness of the generated output values. The inputs can be specified using the Test Bench #1, #2, or #3.

The Test Bench for the sequential circuit reported above is the following one:

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;

entity TESTBENCH is
end TESTBENCH;

architecture stimulus of TESTBENCH is

component conv is
  port(
    clock,reset :in std_logic;
    enable      :in std_logic;
    din         :in std_logic;
    dout        :out std_logic;
    err         :out std_logic );
  end conv;

-- Top level signals go here...
signal clock,reset,enable,din,dout,err: std_logic;

constant clock_period1: 10ns;
constant clock_period2: 20ns;

begin

  DUT: conv port map (clock, reset, enable, din, dout, err);

  PROC_CLOCK: process
    begin
      clock<='0';
      wait for clock_period1;
      clock<='1';
      wait for clock_period2;
    end process PROC_CLOCK;

  PROC_RESET: process
  begin
    ----reset the system at the beginning
      reset<='0';
      wait for 4 ns;
      reset<='1';
      wait for 10 ns;
    ---for experimental reason, further reset the component
    ---during the simulation
```

```
        reset<='0';
        wait for 5850 ns;
        reset<='1';
    end process PROC_RESET;

    STIMULUS1: process
    begin

        ----Define "din" and "enable"
        ----and possible verify "dout" and "err"
        ----using Test Bench #1, #2, #3

        wait;

    end process STIMULUS1;
end stimulus;
```

- When `clock_period2= clock_period1`, the clock signal has duty cycle 50%.

2.3.2. Remarks

When defining the input signals (data, control and status signals) of the component, always guarantee that the signals are stable before the rising edge of the clock. Otherwise the simulation results are not predictable.

2.4. Simulation Steps for VHDL code validation

The simulation steps to validate the VHDL system description are the ones previously described for combinational circuits.