Tutorial: How to make a test bench for a VHDL entity?

Made by :
EL JAOUHARI Fadwa

Each entity in VHDL has its inputs and outputs. To test an entity, one should create different processes for each of its inputs. Two processes are very common : the clock process and the reset process that are going to be discussed further in this tutorial.

Figure 1: Inputs/Outputs of an entity

1. D-latch example:

To introduce the test bench, let us start with a simple D-latch test :
A D Flip Flop (also known as a D Latch) is defined as a memory cell that stores the value on the data line.

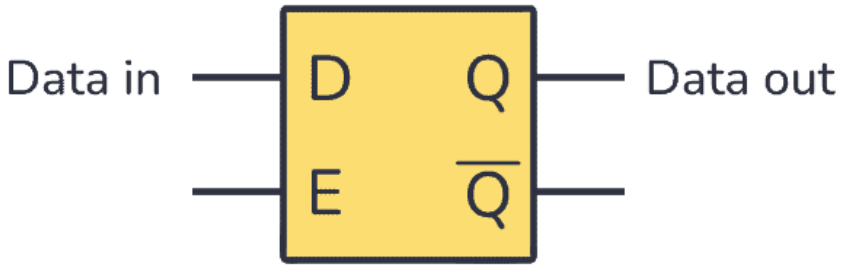

Figure 2 : D Latch

The D latch as shown above is asynchronous and has an enable input. When the E input is 1, the Q output follows the D input. When E is 0, the Q output retains its last value independent of the D input.

To test such an entity one should make sure to give the data some values and also to give different values to the enable input. To test it out with E = 0, wait for a certain time and then test it with E = 1.

The following VHDL code describes the D latch's implementation :

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

entity d_latch_2_top is
    Port ( EN : in  STD_LOGIC;
           D  : in  STD_LOGIC;
           Q  : out STD_LOGIC);
end d_latch_2_top;

architecture Behavioral of d_latch_2_top is
begin

process (EN, D)
begin
    if (EN = '1') then
        Q <= D;
    end if;
end process;

end Behavioral;
```

Figure 3 : VHDL code of D-latch

The following VHDL code is the test bench for the D latch entity :

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

entity d_latch_2_top_tb is
    -- Port ( );
end d_latch_2_top_tb;

architecture Behavioral of d_latch_2_top_tb is

component d_latch_2_top is
    Port ( EN : in  STD_LOGIC;
           D  : in  STD_LOGIC;
           Q  : out STD_LOGIC);
end component;
signal enable, data : std_logic;
signal o: std_logic;
begin
    dut : d_latch_2_top
    port map (EN => enable, D => data, Q => o);


    tb : process
    begin
    data <= '1';
    enable <= '1';
    wait for 50ns;
    data <= '0';
    enable <= '0';
    wait for 50ns;
    data <= '0';
    enable <= '1';
    wait;
    end process;
end Behavioral;
```

Figure 4 : Test bench of D-latch

It is important to note that the test bench code does not have an entity of its own; instead it uses the entity's declaration as a component referred to as the device under test (DUT) or the unit under test (UUT).

Additionally, inputs and outputs are not directly used, they need to be registered in signals defined between "architecture" and "begin" sections alongside the component. These signals should be assigned to each input/output through a function called **"port map"**.

The signals serving as inputs should be tested by giving them values in a stimulus process, for example data signals of type std_logic can be assigned values of 0 or 1.

The "wait for" function is used to delay execution for a specified duration, whereas the "wait" function is used at the end of the process to wait indefinitely.

In conclusion, the test bench should evaluate all possible combinations to validate the correctness of the design.

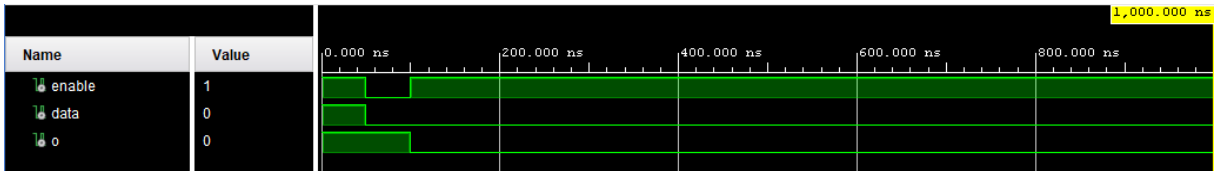

Figure 5 : Simulation of D-latch

This example didn't include all the functions one can use in a test bench. Another important one is **"generic map"**, which is used to assign a value to a generic parameter. For instance, in a clock divider's code, a generic parameter can be defined for the divider, allowing the user to set any value. In this case, a constant should be defined along with the signals that will be assigned to the generic parameter within the "generic map" function.

2. IIR example:

A digital infinite impulse response filter refers to a kind of digital filter that has an impulse response that extends to infinity. That means that the filter's output is dependent not only on the current input but also on previous inputs and previous outputs. IIR filters are characterized by having some form of feedback wherein past outputs are fed back into the filter to create a response that theoretically lasts forever.

To test the IIR's entity, the input should be a sine wave which can be generated by the sine function or by reading a text file containing sine wave data, such as one generated by a Python script.

This design is synchronous and has a reset, thus a clock and reset processes are required:

```
toggle_clk : process
    begin
        clk <= '0';
        wait for T/2;
        clk <= '1';
        wait for T/2;
```

Figure 6 : Clock process

```
reset : process
    begin
        datarsti <= '1';
        wait for T;
        datarsti <= '0';
        wait;
```

Figure 7 : Reset process

To read a file from a text file one can use the **std.textio.all** library that has functions like **file_open** to open a file, **file_close** to close a file, **readline** to read the current line and **read** to extract the value from that line.

An example of a test bench for an IIR filter is given in the Appendix.

Appendix

Code 1 : VHDL code for D-latch

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

entity d_latch_2_top is
    Port ( EN : in  STD_LOGIC;
           D  : in  STD_LOGIC;
           Q  : out STD_LOGIC);
end d_latch_2_top;

architecture Behavioral of d_latch_2_top is
begin

process (EN, D)
begin
    if (EN = '1') then
        Q <= D;
    end if;
end process;

end Behavioral;
```

Code 2 : Test bench for D-latch

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

entity d_latch_2_top_tb is
    -- Port();
end d_latch_2_top_tb;

architecture Behavioral of d_latch_2_top_tb is

component d_latch_2_top is
    Port ( EN : in  STD_LOGIC;
           D  : in  STD_LOGIC;
           Q  : out STD_LOGIC);
end component;
signal enable, data : std_logic;
signal o: std_logic;
begin
    dut : d_latch_2_top
```

```
      port map (EN => enable, D => data, Q => o);


      tb : process
      begin
      data <= '1';
      enable <= '1';
      wait for 50ns;
      data <= '0';
      enable <= '0';
      wait for 50ns;
      data <= '0';
      enable <= '1';
      wait;
    end process;
end Behavioral;
```

Code 3 : Test bench for IIR filter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.math_real.all;
use std.textio.all;
use IEEE.std_logic_textio.all;

entity iir_lpf_real_tb is
--  Port ( );
end iir_lpf_real_tb;

architecture Behavioral of iir_lpf_real_tb is
component iir_lpf_real is
      generic (
      DATA_WIDTH : natural := 32;
      OUTPUT_WIDTH : natural := 64;
      COEFF_WIDTH: natural := 32;
      INTERNAL_SHIFT: natural := 0;
      FRAC_WIDTH: natural := 16;
      READ_SHIFT: natural := 0;
      OUT_SHIFT: natural := 0
      );
      port (
      data_i_i   : in std_logic_vector(DATA_WIDTH-1 downto 0);
      data_en_i  : in std_logic;
      data_clk_i : in std_logic;
      data_rst_i : in std_logic;
      data_i_o   : out std_logic_vector(OUTPUT_WIDTH-1 downto 0);
      data_en_o  : out std_logic;
      data_clk_o : out std_logic;
      data_rst_o : out std_logic;
            a1           : in signed(COEFF_WIDTH-1 downto 0);
      a2          : in signed(COEFF_WIDTH-1 downto 0);
      b0          : in signed(COEFF_WIDTH-1 downto 0);
      b1          : in signed(COEFF_WIDTH-1 downto 0);
      b2          : in signed(COEFF_WIDTH-1 downto 0)
      );
```

```
end component;

constant T : time := 16 ns;
constant DATA_WIDTH : natural := 36;
constant COEFF_WIDTH : natural := 32;
constant FRAC_WIDTH : natural := 20;
constant OUTPUT_WIDTH : natural := 64;
constant INTERNAL_SHIFT : natural := 0;
constant READ_SHIFT : natural := 0;
constant OUT_SHIFT : natural := 0;

signal data_i: std_logic_vector(DATA_WIDTH-1 downto 0);
signal data_o: std_logic_vector(OUTPUT_WIDTH-1 downto 0);
signal clk: std_logic;
signal dataeni, datarsti, dataeno, dataclko, datarsto : std_logic;


constant A1_REAL : real := -1.7525993905846913;
constant A1_INTEGER : integer := integer(A1_REAL * 2.0**(FRAC_WIDTH));
constant a1_coeff : signed(COEFF_WIDTH - 1 downto 0) :=
to_signed(A1_INTEGER, COEFF_WIDTH);

constant A2_REAL : real := 0.9998000100000002;
constant A2_INTEGER : integer := integer(A2_REAL * 2.0**(FRAC_WIDTH));
constant a2_coeff : signed(COEFF_WIDTH - 1 downto 0) :=
to_signed(A2_INTEGER, COEFF_WIDTH);

constant B0_REAL : real := 0.25;
constant B0_INTEGER : integer := integer(B0_REAL * 2.0**(FRAC_WIDTH));
constant b0_coeff : signed(COEFF_WIDTH - 1 downto 0) :=
to_signed(B0_INTEGER, COEFF_WIDTH);


constant B1_REAL : real := 0.5;
constant B1_INTEGER : integer := integer(B1_REAL * 2.0**(FRAC_WIDTH));
constant b1_coeff : signed(COEFF_WIDTH - 1 downto 0) :=
to_signed(B1_INTEGER, COEFF_WIDTH);

constant B2_REAL : real := 0.25;
constant B2_INTEGER : integer := integer(B2_REAL * 2.0**(FRAC_WIDTH));
constant b2_coeff : signed(COEFF_WIDTH - 1 downto 0) :=
to_signed(B2_INTEGER, COEFF_WIDTH);

type Real_Array is array (natural range <>) of real; -- Define a dynamic
array type

begin


dut: iir_lpf_real
generic map (
            DATA_WIDTH => DATA_WIDTH,
            OUTPUT_WIDTH => OUTPUT_WIDTH,
            COEFF_WIDTH => COEFF_WIDTH,
            INTERNAL_SHIFT => INTERNAL_SHIFT,
            FRAC_WIDTH => FRAC_WIDTH,
            READ_SHIFT=> READ_SHIFT,
            OUT_SHIFT => OUT_SHIFT)
port map (data_i_i => data_i, data_en_i => dataeni , data_clk_i => clk,
data_rst_i => datarsti, data_i_o =>data_o, data_en_o =>dataeno,data_clk_o
```

```
=>dataclko, data_rst_o => datarsto ,
          a1=>a1_coeff, a2=>a2_coeff, b0=> b0_coeff, b1=> b1_coeff, b2=>
b2_coeff
          );


tb: process
    --Reading files
    file input_file : text;
    variable current_read_line : line;
    variable val_COMMA : character := ',';
    variable current_read_value : Real_Array(0 to 5000000) := (others =>
0.0);
    variable scaled_value : integer;
    begin
    data_i <= (others => '0');
    dataeni <= '0';

    file_open(input_file, "run29_ch2_amp.txt", READ_MODE);
    while not endfile(input_file) loop
        readline(input_file,current_read_line);
        for jj in 0 to current_read_value'length loop
            report "LENGTH: " & integer'image(current_read_value'length);
            read(current_read_line, current_read_value(jj));
            report "Value: " & real'image(current_read_value(jj));
            read(current_read_line, val_COMMA);
            scaled_value := integer(current_read_value(jj)/2.0**31 *
2.0**FRAC_WIDTH);
            report "Scaled Value: " & integer'image(scaled_value);
            data_i <=
std_logic_vector(to_signed(scaled_value,DATA_WIDTH));
            dataeni <= '1';
            wait until rising_edge(clk);
        end loop;
    end loop;
    file_close(input_file);

    wait;
end process;


toggle_clk : process
      begin
            clk <= '0';
            wait for T/2;
            clk <= '1';
            wait for T/2;
      end process;


stimuli : process
      begin
      datarsti <= '1', '0' after 3*T;
      wait for 2*T;
    assert false report "Test done." severity error;
      wait;
      end process;
end Behavioral;
```

Table of figures