ESO to VHDL Testbench

The Testbench

The testbench is a VHDL file which is used by ISE Simulator to test a VHDL component.

ISE is a programming an simulation tool to develop XILINX FPGAs. This work suite include a programminng workspace, a compiler, simulator (ISIM) and much more.

When you program a new component, you also want to test its behavior. But you do not always have an FPGA, so you could use a simulator. For that the simulator knows what input signals to simulate you need a so called tesbench.

A testbench lists the component you want to test e.g. abo. (You have written a vhdl file which behaves like ABO and this component is also called abo). It instantiate this componant as a Unit Under Test (UUT). This component (uut) will be tested with the input and outputs you have specified in a test process.

At first a testbench code example from ABO is shown for better understanding.

```
--/*****************************************************************************/
--/*               G E N E R A T E D    V H D L   C O D E                     */
--/*****************************************************************************/
--/* KIELER - Kiel Integrated Environment for Layout Eclipse RichClient        */
--/*                                                                           */
--/* http://www.informatik.uni-kiel.de/rtsys/kieler/                           */
--/* Copyright 2013 by                                                         */
--/* + Christian-Albrechts-University of Kiel                                  */
--/*   + Department of Computer Science                                        */
--/*     + Real-Time and Embedded Systems Group                                */
--/*                                                                           */
--/* This code is provided under the terms of the Eclipse Public License (EPL).*/
--/*****************************************************************************/

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY abo_tb IS
END abo_tb;

ARCHITECTURE behavior OF abo_tb IS

COMPONENT abo
PORT(
     tick : IN  std_logic;
     reset : IN std_logic;
     --inputs
     A: IN boolean;
     B: IN boolean;
     --outputs
     O1 : OUT boolean;
     O2 : OUT boolean;
     A_out : OUT boolean;
     B_out : OUT boolean
    );
END COMPONENT;

--Inputs
signal A : boolean := false;
signal B : boolean := false;

--Outputs
signal O1 : boolean := false;
signal O2 : boolean := false;
signal A_out : boolean := false;
signal B_out : boolean := false;

--Control
signal reset : std_logic := '0';
signal tick : std_logic := '0';
constant tick_period : time := 100 ns;

BEGIN
```

```
uut: abo PORT MAP(
    tick => tick,
    reset => reset,
    --Inputs
    A => A,
    B => B,
    --Outputs
    O1 => O1,
    O2 => O2,
    A_out => A_out,
    B_out => B_out
);

tick_process: process
begin
    tick <= '0';
    wait for tick_period/2;
    tick <= '1';
    wait for tick_period/2;
end process;

-- Stimulus process
stim_proc: process
begin
    wait for 1 ps;

    --sim Process

    --NEW TRACE
    reset <= '1';
    wait for tick_period;
    reset <= '0';

    -- tick 1
    A <= true;
    B <= false;
    wait for tick_period;
    assert( O1 = true )
        report "1st trace: 1st tick: O1 should have been true"
        severity ERROR;
    assert( O2 = false )
        report "1st trace: 1st tick: O2 should have been false"
        severity ERROR;
    assert( A_out = true )
        report "1st trace: 1st tick: A_out should have been true"
        severity ERROR;
    assert( B_out = true )
        report "1st trace: 1st tick: B_out should have been true"
        severity ERROR;

    -- tick 2
    A <= false;
    B <= false;
    wait for tick_period;
    assert( O1 = true )
        report "1st trace: 2nd tick: O1 should have been true"
        severity ERROR;
    assert( O2 = false )
        report "1st trace: 2nd tick: O2 should have been false"
        severity ERROR;
    assert( A_out = false )
        report "1st trace: 2nd tick: A_out should have been false"
        severity ERROR;
    assert( B_out = false )
        report "1st trace: 2nd tick: B_out should have been false"
        severity ERROR;

    -- tick 3
    A <= false;
    B <= true;
    wait for tick_period;
```

```
        assert( O1 = false )
            report "1st trace: 3rd tick: O1 should have been false"
            severity ERROR;
        assert( O2 = true )
            report "1st trace: 3rd tick: O2 should have been true"
            severity ERROR;
        assert( A_out = false )
            report "1st trace: 3rd tick: A_out should have been false"
            severity ERROR;
        assert( B_out = true )
            report "1st trace: 3rd tick: B_out should have been true"
            severity ERROR;

        --NEW TRACE
        reset <= '1';
        wait for tick_period;
        reset <= '0';

        -- tick 1
        A <= false;
        B <= false;
        wait for tick_period;
        assert( O1 = false )
            report "2nd trace: 1st tick: O1 should have been false"
            severity ERROR;
        assert( O2 = false )
            report "2nd trace: 1st tick: O2 should have been false"
            severity ERROR;
        assert( A_out = false )
            report "2nd trace: 1st tick: A_out should have been false"
            severity ERROR;
        assert( B_out = false )
            report "2nd trace: 1st tick: B_out should have been false"
            severity ERROR;

        -- tick 2
        A <= true;
        B <= false;
        wait for tick_period;
        assert( O1 = false )
            report "2nd trace: 2nd tick: O1 should have been false"
            severity ERROR;
        assert( O2 = true )
            report "2nd trace: 2nd tick: O2 should have been true"
            severity ERROR;
        assert( A_out = true )
            report "2nd trace: 2nd tick: A_out should have been true"
            severity ERROR;
        assert( B_out = true )
            report "2nd trace: 2nd tick: B_out should have been true"
            severity ERROR;
        wait;
    end process;

END;
```

Explanation:

Line 23: ABO component declaration

Here the ABO component is declared, so the testbanch knows which component to test.

Line 38: declaration of local signals

Local signals are needed for internal communication and for interconnection between components and processes.

Line 55: instantiation of uut

Here is ABO instantiated

Line 69: tick process

The tick process simulates the tick. This signal is a kind of a clock signal. Its cycle duration is set in variable tick_period.

Line 79: the simulation process

The main simulation process, here the input signals are set and the output signals are tested according to specifications in ESO file.

Technical View

Therefore the core ESO contains no information about input and output signals the proper SCL model is needed to generate the testbench.

The variable declaration must conform to the used variables in the ESO file, logically.

SCL Model

```
module ABO
input A : boolean;
input B : boolean;
output O1 : boolean = false;
output O2 : boolean = false;
output A_out : boolean;
output B_out : boolean;
{
    fork
        __WaitAB_HandleA_WaitA:
        if A then
            A_out = true;
            B = true;
            B_out = true;
            O1 = true;
            goto __WaitAB_HandleA_DoneA;
        end;
        pause;
        goto __WaitAB_HandleA_WaitA;
        __WaitAB_HandleA_DoneA:
    par
        __WaitAB_HandleB_WaitB:
        pause;
        if ! B then
            goto __WaitAB_HandleB_WaitB;
        end;
        B_out = true;
        O1 = true;
    join;
    O1 = false;
    O2 = true;
}
```

Core ESO File

```
!reset ;
%% A : true
%% O1 : true
%% A_out : true
%% B_out : true
;

%% O1 : true
;

%% B : true
%% O2 : true
%% B_out : true
;

!reset ;
;

%% A : true
%% O2 : true
%% A_out : true
%% B_out : true
;
```

ESO File

```
!reset ;
A
% Output : O1 A_out B_out
;
% Output : O1
;
B
% Output : O2 B_out
;
!reset ;
% Output :
;
A
% Output : O2 A_out B_out
;
```

⚠ SCL file is not realy correct, will be corrected as soon as possible

The lines 23 to 66 (testbench file) were generated from the SCL model file. The model tells the transformation which input and output the abo component must have and the type of these variables (signals in VHDL). So the SCl file is needed for code generation.

The simulation process (starts at line 79) is generated using the core ESO file. How the simulation process is generated will be shown at the following example:

SCL	ESO trace	Testbench

```
module test
input  A ;
input  B : boolean
= false;
input  B_value :
integer = 0;
input  C :boolean =
false;
input  C_value :
boolean = false;
output  D;
output  E : boolean
= false;
output  E_value :
integer = 5;
output  F : boolean
= false;
output  F_value :
boolean = false;
{
    //pause;
}
```

```
!reset;
A C(false)
%Output: D F(false)
;
```

⚠️ normally the core ESO is used, but for better unterstanding we use the normal ESO trace

```
A <= true;
B <= false;
C <= true;
C_value <= false;
wait for tick_period;
assert( D = true )
        report "1st trace: 1st tick: D
should have been true"
        severity ERROR;
assert( E = false )
        report "1st trace: 1st tick: E
should have been false"
        severity ERROR;
assert( F = true )
        report "1st trace: 1st tick: F
should have been true"
        severity ERROR;
assert( F_value = false )
        report "1st trace: 1st tick:
F_value should have been false"
        severity ERROR;
```

Set inputs

Line 2 and 3 in the testbench are setting the inputs. All (!) inputs must be set!

- Pure signal which are present are set to the according value, e.g. *A<=true;*
- Valued signals which are present are set to their according value e.g. *C_value <= false;* and set present *B<=true;*
- ABSENT values (not listed in ESO files inputs) must be set absent
    - pure singals (not listed) e.g. K <= false
    - valued signals e.g. *B <= false*, only the present value will be set, the valued signal is not touched

Wait for the tick to pass by

The code *wait for tick_period;* waits for one tick, so the Hardware can compute the output values.

Test Outputs

After the tick has passed by we must check if the hardware computes the correct outputs. This is done by assertions. Every (!) output must be tested!

- Pure signals: Test the pure output signal according to the current ESO tick, if listed in the tick, e.g. *assert(D = true)*. If it is not specified in the trace test for absence,
- Valued signals: For valued signals, that are specified in the current ESO tick, test the present singal and the valued signal, e.g. *assert(F = true)* a nd *assert(F_value = false)*
- Valued signals, which are not listed in the current tick in the ESO file: test only the present flag for absence (We can say anything about absent valued signals)

If an assertion failed the corresponding error will be printed to a log file. The severity level tells the simulator to go on with the simulation although an error occur.