Integrated Systems Architectures

General suggestions

1 Test Bench

This document is to be intended as a quick tutorial on HDL test benches. Nowadays, test benches are heavily used in companies to validate every single component. Entire company sections are dedicated to test and validate what comes from the design section.
During this course you will face two different situations: given specifications, you design a component and a test bench is provided to test your architecture; given specifications you must design both the architecture and the test bench.
Teachers will use automatic test bench systems to verify that the designs are correct. Considering this it becomes very important to respect the specifications and interfaces. Architectures with wrong interfaces will be considered as wrong designs.
Whenever you design a component, it is of fundamental importance to test it to ensure that its functioning is compliant to the specifications. In order to verify that a component behaves according to the project requirements, it must be subjected to a test:

- controlled stimuli are applied on the input;
- output outcomes are verified to be compliant.

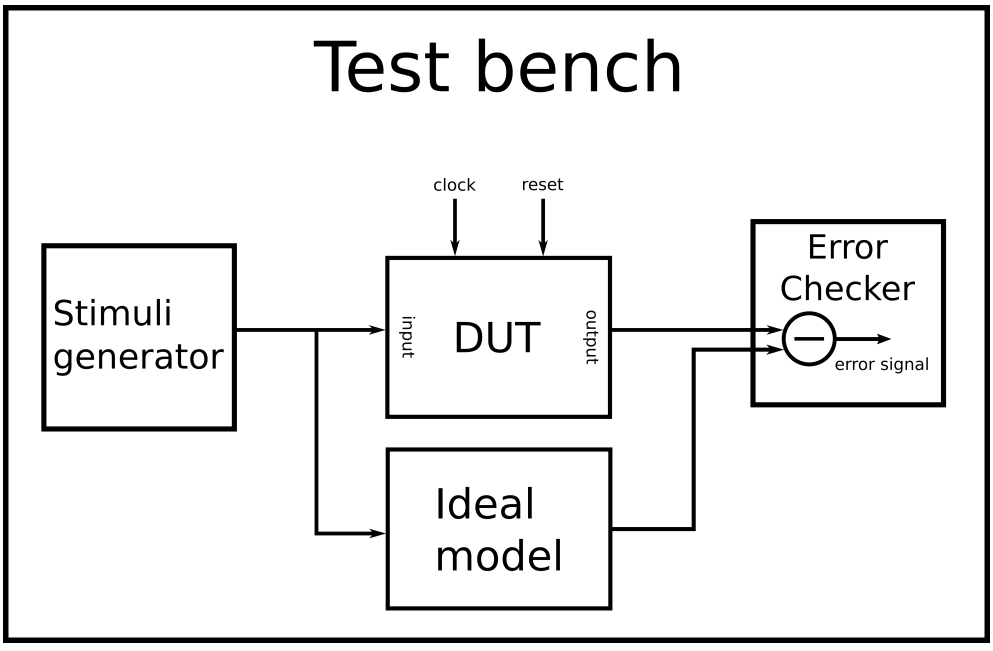

Figura 1: Schematic block of a test bench.

This VHDL operation is possible with an HDL simulator, in our case Modelsim or Questasim. An appropriate entity shall be provided to describe the test to be performed: the **unit of test bench**.
The unit of test bench shall:

- incorporate the Unit Under Test (UUT), or Device Under Test (DUT);
- contain a description of the stimuli to be applied;

HDL test bench characteristics:

- No input or output ports;
- Input signals can be varied in time, so as to cover multiple configurations;
- If combinations of input signal values are too numerous, the test is limited to more meaningful configurations;
- Signals can be described using concurrent instructions (assignment with **after** keyword) or sequential (**wait** and **wait for** in **process** instructions).

Test benches are used for test purpose only, **not for synthesis**, therefore is possible to use several VHDL constructs such as **assert**, **report**, **for loops** etc. Remember to remove all test bench files when you perform the synthesis of your circuit, otherwise, Design Compiler will report many errors, and more important to synthesize the test bench has no meaning.
In the following, a simple example of a test bench for a combinatorial component will be presented.

```
-- half_adder.vhd

library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
port (a, b : in std_logic;
sum, carry : out std_logic
);
end half_adder;

architecture arch of half_adder is
begin
sum <= a xor b;
carry <= a and b;
end arch;
```

As you can easily understand the DUT is a half adder. Now let's analyze a possible implementation of the test bench usually abbreviated as "tb".

```
-- example of a simple test bench

library ieee;
use ieee.std_logic_1164.all;


entity half_adder_tb is
end half_adder_tb;

architecture tb of half_adder_tb is
signal a, b : std_logic;  -- inputs
signal sum, carry : std_logic;  -- outputs
begin
-- connecting test bench signals with half_adder.vhd
UUT : entity work.half_adder port map (a => a, b => b, sum => sum, carry => carry);

-- inputs change in time (not synthesizable)
-- 00 at 0 ns
-- 01 at 20 ns, as b is 0 at 20 ns and a is changed to 1 at 20 ns
-- 10 at 40 ns
-- 11 at 60 ns
```

```
a <= '0', '1' after 20 ns, '0' after 40 ns, '1' after 60 ns;
b <= '0', '1' after 40 ns;
end tb ;
```

As you can see the entity of the *tb* has no inputs or outputs. The reason is that this component does not communicate with other external elements. The *DUT* is then connected to local signals that vary in time, going through all the possible combinations. Analyzing how the outputs of the DUT vary one can understand if it is performing correctly.

What happens when we have a very complex system? Do we have to check the behavior of every single signal?

No, we do not. When we test very complex device one trick is to generate an **error signal** that is triggered when something goes wrong.

How?

When you design a hardware device, it typically derives from a mathematical model or an algorithm. To test means to check that the model and the behavior of the HDL description are equal. So we have results coming from the HDL simulation and the ideal model. The error signal can be easily generated by a simple difference between such results.
You can exploit three different approaches:

- Implement the ideal model directly in the test bench by describing it in a behavioral manner. The error signal would be part of the tb as a difference of the data produced by the DUT and the model;
- Implement the ideal model exploiting another language such as Python, Matlab, etc. and save the results in a text file (or similar). In the tb is possible to read the text file and generate an error signal as the difference of the data produced by the DUT and the model;
- Save the data produced by the DUT on a txt file (or similar) and exploit external software to check for errors.

N.B. The previous example implement a tb for a combinatorial circuit. For sequential circuits you need also to create a **reset** and a **clock** signals. Here an example:

```
-- Process for generating the clock
Clk <= not Clk after ClockPeriod / 2;
```

Clk is a signal that every ClockPeriod / 2 changes its state.

To deepen you knowledge you can visit:
`https://vhdlguide.readthedocs.io/en/latest/vhdl/testbench.html`
`https://moodle.epfl.ch/pluginfile.php/1833321/mod_resource/content/1/vhdl_testbench_tutorial.pdf`
`https://www.unirc.it/documentazione/materiale_didattico/599_2008_94_2798.pdf`

2 Scripting

Almost all the EDA tools used in this course supporte scripting. The use of the GUI is helpful as a first step to understand the flow, but then to repeat the flow (or part of the flow) scripts help in speeding up. The main language used in EDA tools scripting is TCL. However, for our purpose it becomes almost a sequence of commands. You can find the commands with all

the details about paramenters in the documentation of each tool. However, it is possible to derive the commands by checking the console or the log file of the command-line shell, which is available in the GUI of the tool.

Example:

```
analyze -f vhdl -lib WORK ../src/file1.vhd
analyze -f vhdl -lib WORK ../src/file2.vhd
analyze -f vhdl -lib WORK ../src/top.vhd
elaborate top -lib WORK
create_clock -name My_CLK -period 5.0 CLK
compile
report_area >> ./report_area.txt
report_timing >> ./report_timing.txt
```

3 Homework report preparation

On "Portale della didattica" you have a template for the homework report. Please remeber that the report should be self-contained and should show tables to summarize the achieved results comparing different implementations. Diagrams, graphs and figures which could be useful to better understand your design have to be included in the report. Code, scripts and tool's reports are not required in the final report.

Note: Please be ready to present your work during the oral discussion. You have to explain your work, the results you obtained and compare the different solutions.