Lab01 – Combinatorial circuits

1 Single output logic function: model & test-bench

1.1 Your first runs

VHDL model of a simple combinatorial circuit is given below. Make sure that you understand all the statements first (which libraries are used & why, module definition & implementation).
Encode the file in say `lc.vhd` text file.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity myLogicFunction is port(
   a    : in std_logic;
   b    : in std_logic;
   c    : in std_logic;
   o    : out std_logic  -- Note that the last one doesn't have a semicolon at the end
);end myLogicFunction;

architecture arch of myLogicFunction is
begin
   o <= a and b and c ; -- Concurrent assignment (here only one)
end arch;
```

A test-bench for the circuit above is given below. Make sure that you understand all the statements first. Encode the file in the `lc_tb.vhd`.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity myLogicFunction_tb is
   -- You should know why here we do not have anything
end entity;
architecture beh of myLogicFunction_tb is
   -- We will use the module myLogicFunction,
   -- so we need to have it's definition
   -- that needs to match the one in the module
   component myLogicFunction is port(
   a,b,c      : in  std_logic;
   o          : out std_logic);
   end component;
   -- These are the internal wires
   signal a,b,c,o : std_logic;
   begin
      -- Here we instantiate the module with instance name uut
      uut : myLogicFunction port map(a => a, b => b, c => c, o => o);
      -- What would be the alternative way to connect this module?
      stim : process
         begin
            a <= '0'; b <= '0'; c <= '0';
            wait for 10 ns;
            assert ((o = '0'))
            report "test failed for input combination 000" severity error;
            a <= '1'; b <= '1'; c <= '1';
            wait for 10 ns;
            assert ((o = '1'))
            report "test failed for input combination 111" severity error;
            a <= '0'; b <= '1'; c <= '0';
            wait for 10 ns;
            assert ((o = '0'))
            report "test failed for input combination 010" severity error;
      end process;
end beh;
```

Do the following:

- Synthesise the circuit and the test-bench (basic syntax check)
- Perform functional simulation and observer the waveforms, that should look something like this:

Do not forget to recompile if you change anything in the model and the test-bench.

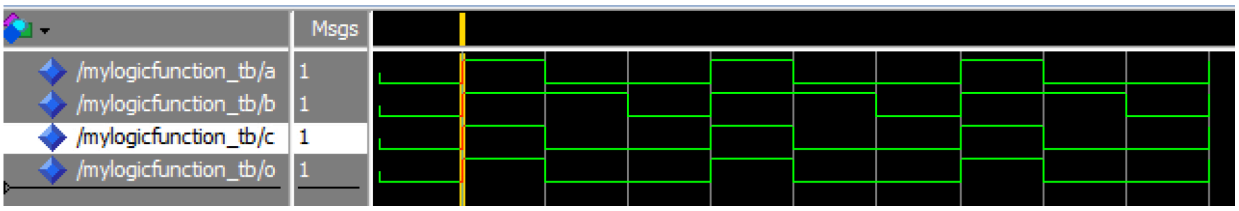

Figure 1: Waforms

1.2 Make your own model

Extend the same model to have a logic function with 5 inputs. Logic function choice is up to you, but make sure that you could eventually check the output by hand.

1.3 Make your own model

Write the test-bench for 5-inputs that covers all possible input combinations to enable full verification. Do this in a smart way, i.e. do not enumerate all the possibilities by yourself, imagine a problem with 25 input variables.
Perform circuit simulation and analyse the waveforms.

2 Logic functions with multiple outputs

Using the previous exercise (Section 1.1) as a starting point, implement the following:

- Add two supplementary outputs (use arbitrary logic functions) to the design (you end up with three logic functions with three inputs)
- Extend the test-bench with multiple assignments
- Use one of the module outputs as input (output on the right of assignment operator), example:

```vhdl
entity myLogicFunction is port(
   a   : in std_logic;
   b   : in std_logic;
   c   : in std_logic;
   o   : out std_logic
);end myLogicFunction;

architecture arch of myLogicFunction is

begin
   o <= o and b and c; -- Output is on the right !!!
end arch;
```

- Assign two different logic functions to the same output, example:

```vhdl
entity myLogicFunction is port(
   a   : in std_logic;
   b   : in std_logic;
   c   : in std_logic;
   o   : out std_logic
);end myLogicFunction;
architecture arch of myLogicFunction is
begin
   o <= a and b and c; -- One concurrent assignment to output
   o <= a or b or c;   -- Targets the same output
end arch;
```

Synthesise, simulate (when possible !), analyse & interpret the waveforms obtained.

3 Design hierarchy (structured designs)

Make a hierarchical design composed out of two identical sub-modules as shown:

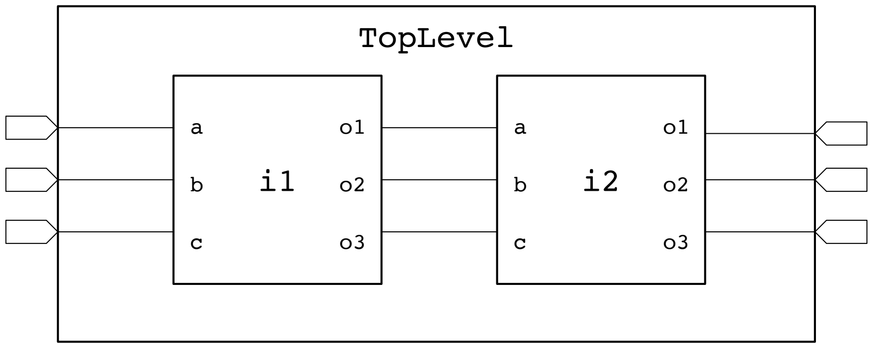

Figure 2: Hierarchical logic circuit

You are free to chose a sub-module functionality, but keep the number of inputs/outputs as in the figure. Write the `TopLevel.vhd`, the test-bench and validate the circuit functionality.

4 Basic Half & Full adder circuits

VHDL models of a half and full adder circuits are given below. Make sure that you understand all the statements first (library used, module definition, as well as the origin of the logic equations).

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity HalfAdder is port(
   a, b        : in std_logic;
   s, c0       : out std_logic
);end HalfAdder;

architecture beh of HalfAdder is begin
   s  <= a xor b ;        -- sum bit
   co <= a and b ;        -- carry bit
end beh;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity FullAdder is port(
   a, b, ci    : in std_logic;
   s, c0       : out std_logic
);end FullAdder;

architecture beh of FullAdder is begin
   s  <= a xor b xor ci;                  -- sum bit
   co <= (a and b) or ((a xor b) and ci); -- carry bit
end beh;
```

Then do the following:

- Write the test-benches for the above modules (imagine exhaustive tests)
- Simulate the design and print out the waveforms
- Analyse the waveforms (you can use decimal output for the waveforms)

5 Ripple carry adders

Using half and full adder circuits from the previous exercise as elementary modules, write a structured VHDL model for 4, 8 and 16-bit adders. Use `generics` to allow model configurability at RTL level, so that you have to write only one VHDL model.
For the models above:

- Write the top-level HDL module & test-benches for different adders. Make an exhaustive test of all input combination.
- Simulate & analyse the waveforms (use decimal output formatting)