Management and analysis of physics datasets, Part. 1

First Laboratory

Antonio Bergnoli bergnoli@pd.infn.it - Filippo Marini filippo.marini@pd.infn.it

18/11/2020

Laboratory Introduction

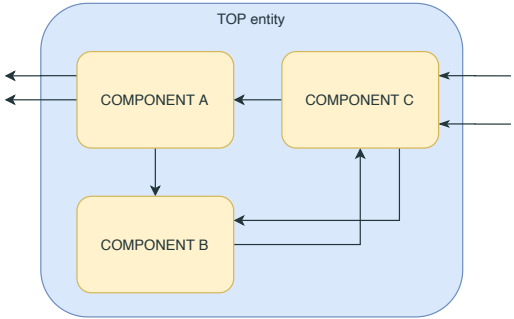
Goals

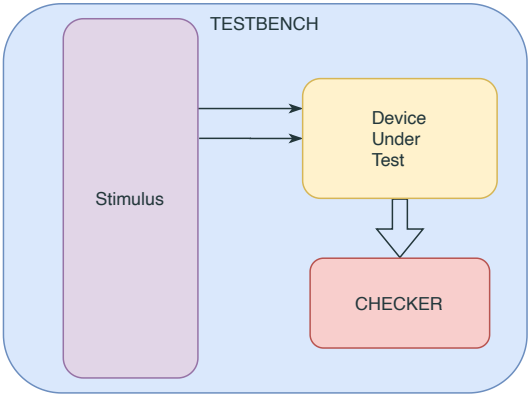
- Introduce the VHDL language
- Exploit the simulation tools.
- Become familiar with the basic VHDL constructs

VHDL constructs used in this lecture

1. **entity** - **component** - **instance**
2. **signal**
3. **process**
4. **assert** - **report** string
5. **wait**

Structured design





Tool used in these lectures

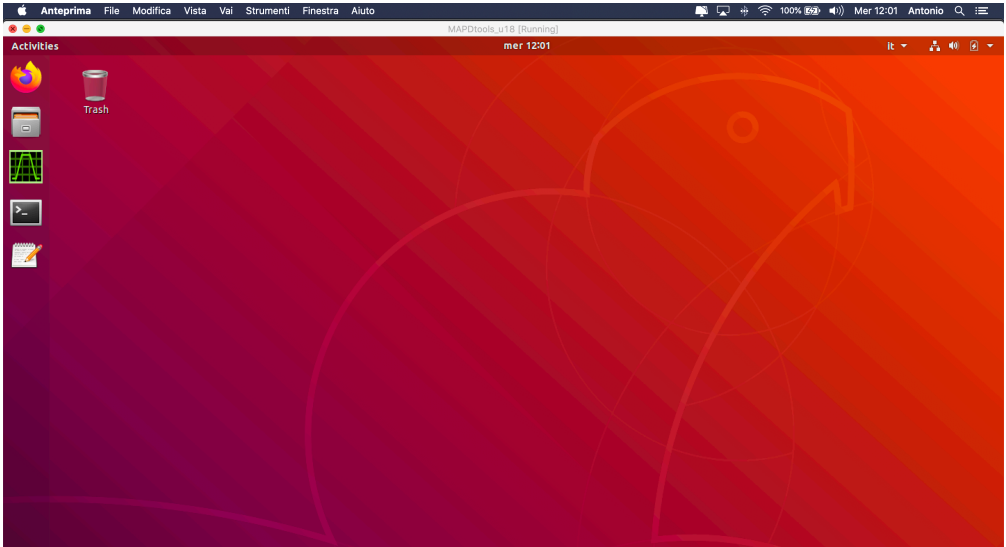
- Simulator
  - ghdl
  - Xilinx vivado simulator
- waveform viewer
  - gtkwave
  - Xilinx Vivado simulator
- Text editor (whatever you prefer)
  - some suggestion:
    - Atom
    - gedit
    - Notepad++
    - Emacs, vim (more advanced)

Selection of Internet resources

1. https://www.edaplayground.com/
2. https://vhdlwhiz.com/
3. https://surf-vhdl.com/

The virtual machine - Hands On

Password: STUDENT01



The “Hello World” step by step - Hands On

Analize - elaborate - run

- edit the source file
- save it with .vhd extension

```
ghdl -a hello_world.vhd
ghdl -d # check the working directory
ghdl -e <entity-name>
```

an executable file it should have appeared, run it:

```
./hello_world
```

The “Heartbeat” step by step - Hands On

Analize - elaborate - run

- edit the source file
- save it with .vhd extension

```
ghdl -a heartbeat.vhd
ghdl -d # check the working directory
ghdl -e heartbeat
```

an executable file it should be appeared , run it (adding some parameters):

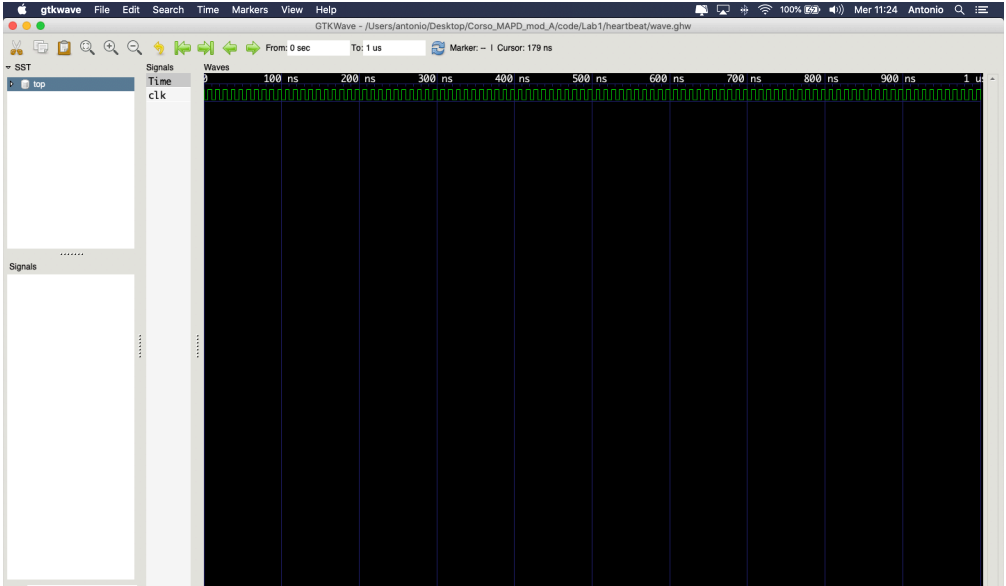
```
./heartbeat --wave=wave.ghw --stop-time=1us
```

a wave file (.ghw extension) shoud have appeared

open it with `gtkwave`

```
gtkwave wave.ghw
```

Gtkwave screenshot



Simulation - Testbenches

What VHDL is designed for :

1. Documentation : Formal description of a digital design;
2. Synthesis : **Infer** digital logic and digital sequential structures specifying a formal description;
3. Verification : assert that a design will behave correctly **before** implement it.
   - Only a **subset** of the language constructs are synthesizables : (IEEE 1076.6)
   - **all** the language contstructs are allowed in simulation.

Simulation - reasons

- Once written the code describing the component, it is necessary check its working.
- Regarding on not trivial projects, simulation is a crucial and mandatory step in the development of the project : testing any firmware on hardware without simulation has to be avoided because it is unsafe and very time consuming.
- So with the simulation we are going to stimulate the inputs and check the accuracy of the outputs.

Check the examples in the final slides .

Testbench is the main verifcation strategy

- to simulate the behavior of the code written, we have to make a VHDL file, in order to define the input stimuluses and so check the output evolution.
- This file is a **testbench**.
- The component to be tested (the "Hello World" top) has to be instantiated and the processes where the input signals values are described have to be written.
- The testbench dœs not have input and output ports in its entity declaration.
- Declaration of the internal signals. They are used to connect the component under test (**U**nit **U**nder **T**est) to the stimulus.
- Instantiation of the component. Each port is connected through the internal signals.
- Definition of the input stimuluses. It is very very very good practice check each input combination.

Behavioural Simulation

- with the **Behavioral Simulation** we check that the *component* works as expected.
- The output changes happen simultaneously to the input changes. This kind of simulation check the behavior of the code written without considering the delays of a real electronic system.
- if this simulation is successfully, almost always the project works in the hardware too.
- The code in this course is reused in the form of **components**.
- The components are then located in the top code.
- You'll can see a construction of a hierarchical design.

Most important Question

How to declare that a simulation is succesful?

How to use a component?

1. Definition; `entity (...)`
2. Declaration; `component (...)`
3. Instantiation. `DUT: (...)`

Combinational Logic Circuits: examples

Example 0 : Hello World

```
use std.textio.all;
entity hello_world is
end hello_world;
architecture behaviour of hello_world is
begin
  process
    variable l : line;
  begin
    write (l, string'("Hello world!"));
    writeline (output, l);
    wait;
  end process;
end behaviour;
```

Example 1 : Heartbeat

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

entity heartbeat is
  port (clk : out std_logic);
end heartbeat;

architecture behaviour of heartbeat is
  constant clk_period : time := 10 ns;
begin
  -- Clock process definition
  clk_process : process
  begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
  end process;
end behaviour;
```

Top Entity

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

entity heartbeat_top is
end entity heartbeat_top;

architecture str of heartbeat_top is
  component heartbeat is
    port (
      clk : out std_logic);
  end component heartbeat;
  signal clk : std_logic;
begin  -- architecture str
  DUT: entity work.heartbeat
    port map (
      clk => clk);
end architecture str;
```

Example 2 : or gate

```vhdl
-- Simple OR gate design
library IEEE;
use IEEE.std_logic_1164.all;

entity or_gate is
  port(
    a : in  std_logic;
    b : in  std_logic;
    q : out std_logic);
end or_gate;

architecture rtl of or_gate is
begin
  process(a, b) is
  begin
    q <= a or b;
  end process;
end rtl;
```

TestBench

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity testbench is
end testbench;
architecture tb of testbench is
  component or_gate is
    port(
      a : in  std_logic;
      b : in  std_logic;
      q : out std_logic);
  end component;
  signal a_in, b_in, q_out : std_logic;
begin
  DUT : or_gate port map(a_in, b_in, q_out);
  process
  begin
    a_in <= '0'; b_in <= '0';
    wait for 1 ns; assert(q_out = '0') report "Fail 0/0" severity error;
    a_in <= '0'; b_in <= '1';
    wait for 1 ns; assert(q_out = '1') report "Fail 0/1" severity error;
    a_in <= '1'; b_in <= 'X';
    wait for 1 ns; assert(q_out = '1') report "Fail 1/X" severity error;
    a_in <= '1'; b_in <= '1';
    wait for 1 ns; assert(q_out = '1') report "Fail 1/1" severity error;
    -- Clear inputs
    a_in <= '0'; b_in <= '0';
    assert false report "Test done." severity note;
    wait;
  end process;
end tb;
```

Example 3 : adder

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity adder is
  generic (
    width : integer := 8);

  port (
    a   : in  std_logic_vector(width - 1 downto 0);
    b   : in  std_logic_vector(width - 1 downto 0);
    sum : out std_logic_vector(width - 1 downto 0));
end entity adder;
architecture rtl of adder is

begin  -- architecture rtl

  sum <= std_logic_vector(unsigned(a) + unsigned(b));

end architecture rtl;
```

TestBench

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity adder_tb is
end entity adder_tb;
architecture test of adder_tb is
  constant width : integer := 8;
  signal a   : std_logic_vector(width - 1 downto 0);
  signal b   : std_logic_vector(width -1 downto 0);
  signal sum : std_logic_vector(width - 1 downto 0);
  signal Clk : std_logic := '1';
begin  -- architecture test
  DUT : entity work.adder
    generic map (
      width => width)
    port map (
      a   => a,
      b   => b,
      sum => sum);
  main : process is
  begin  -- process main
    a <= X"AA";
    B <= X"BB";
    wait for 1 ns;
    a <= X"A0";
    B <= X"B0";
    wait for 1 ns;
    wait;
  end process main;
end architecture test;
```

Homework

Suggested exercises

- Redo 1, 2, 3, ..., N times the exercises changing some directives and check the results.
  - change the logical operator(s)
  - change the arithmetic operation
  - try to instance and connect more than one component in the same design