Arquiteturas de Alto Desempenho 2024/2025

Practical class AAD_P10 (2024-11-25 and 2024-11-26)

Tomás Oliveira e Silva

1 Objectives

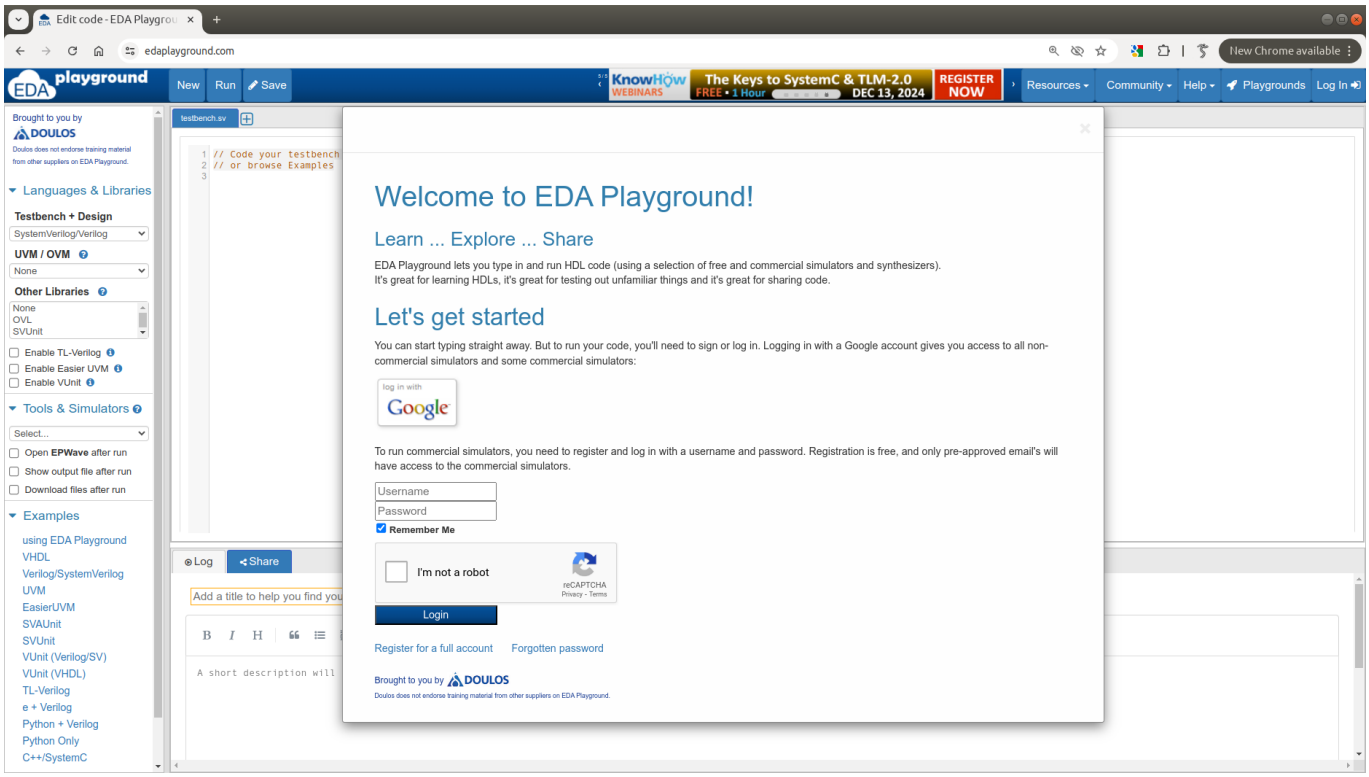
Simulate some simple combinational circuits described using VHDL.

2 First exercise, with an introduction to the VHDL simulators we will use

For convenience, we with use the edaplayground online Electronic Design Automation (EDA) tool. Among other things, it is capable of simulating VHDL designs. As an alternative, it is also possible to use the `ghdl` program (to simulate), coupled with the `gtkwave` program (to view the waveforms).

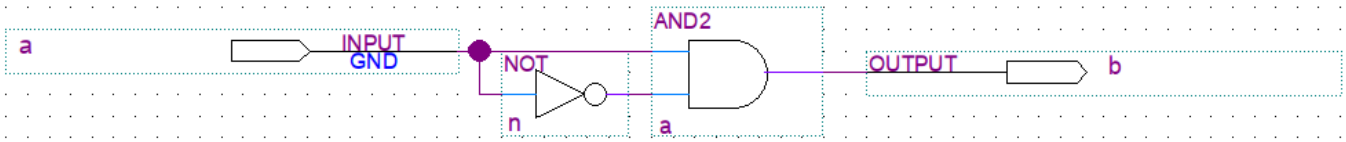
2.1 Using `edaplayground`

First of all, it is necessary to go to the `edaplayground` web site, `http://edaplayground.com/` and register yourself. You should be greeted with the following web page:



Just press the `Register for a full account` link on the lower left, enter your name and email address (use your student email address) and any other information that is mandatory, and finish the registration process. You will receive an email with instructions about how to activate your account.

You are now ready to simulate our first example, which describes in VHDL the digital circuit of the following figure. The description of this digital circuit is split among several files.



not_gate_1.vhd — Description of a simple **not** gate. Its contents are as follows:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity not_gate_1 is
  port(input0  : in  std_logic;
       output0 : out std_logic);
end not_gate_1;

architecture behavioral of not_gate_1 is
begin
  output0 <= transport not input0 after 10 ps;
end behavioral;
```

Recall that in VHDL we describe entities and interconnections between entities. Each entity has input, output, and input/output signals, that are specified using a `port` statement. In this case we have one input signal, of type `std_logic` (the type is declared in `IEEE.std_logic_1164.all`), and one output signal, of the same type. Each entity has one or more architectures, that describe how its output signals are computed from its input signals. Recall also that <= represents a physical connection: the signal on the left-hand side is connected to the result of the expression on its right-hand side. For simulation purposes a transport delay of 10ps was added to the output. (Omitting the keyword `transport` gives rise to an inertial delay; in an inertial delay pulses shorter than the delay are discarded.)

and_gate_2.vhd — Description of a simple **and** gate with two inputs.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity and_gate_2 is
  port(input0  : in  std_logic;
       input1  : in  std_logic;
       output0 : out std_logic);
end and_gate_2;

architecture behavioral of and_gate_2 is
begin
  output0 <= transport input0 and input1 after 10 ps;
end behavioral;
```

ex1.vhd — Description of our circuit.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ex1 is
   port(a : in  std_logic;
        b : out std_logic);
end ex1;

architecture structural of ex1 is
  signal s_not_a : std_logic;
```

```
begin
  my_not : entity work.not_gate_1(behavioral)
           port map(input0  => a,
                    output0 => s_not_a);
  my_and : entity work.and_gate_2(behavioral)
           port map(input0  => a,
                    input1  => s_not_a,
                    output0 => b);
end structural;
```

Notice how the **not** and **and** gates are instantiated: =>, not to be confused with <=, connects a port signal of the entity being instantiated, whose name is on the left-hand side of =>, to a signal of the entity that is doing the instantiation (whose name is on the right-hand size of =>).

`ex1_tb.vhd` — The test bench. The test bench instantiates the circuit we want to simulate and provide stimuli to it. It may also test the outputs of that circuit, but we will not do it here.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ex1_tb is
end ex1_tb;

architecture stimulus of ex1_tb is
  signal s_u,s_v : std_logic;
begin
  uut : entity work.ex1(structural)
              port map(a => s_u,
                       b => s_v);
  process
  begin
    for cycle in 0 to 4 loop
      u <= '0';
      wait for 50 ps;
      u <= '1';
      wait for 50 ps;
    end loop;
    wait;
  end process;
end stimulus;
```

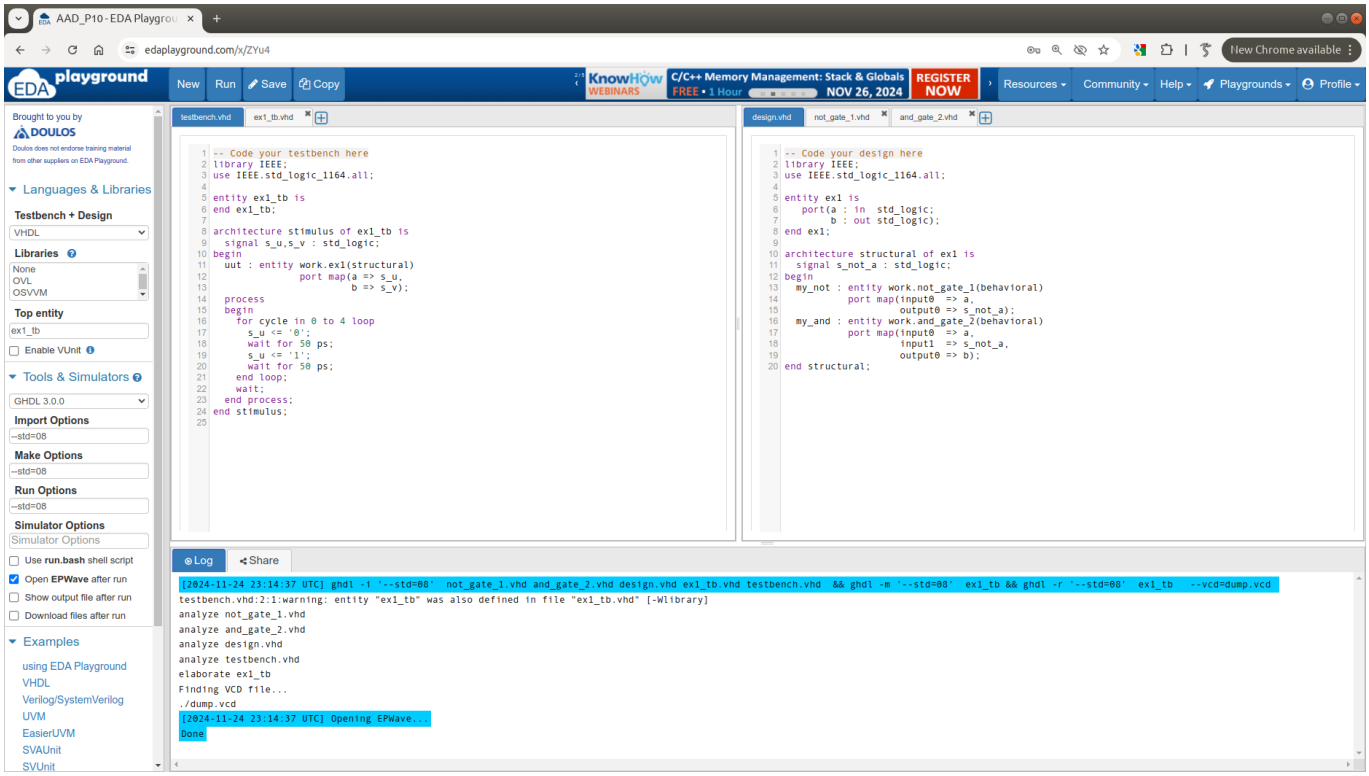
The stimulus is just fives periods of a square wave with a period of 100ps. The `wait` at the end stops the simulation.

To upload the example to the `edaplayground` web page, do the following:

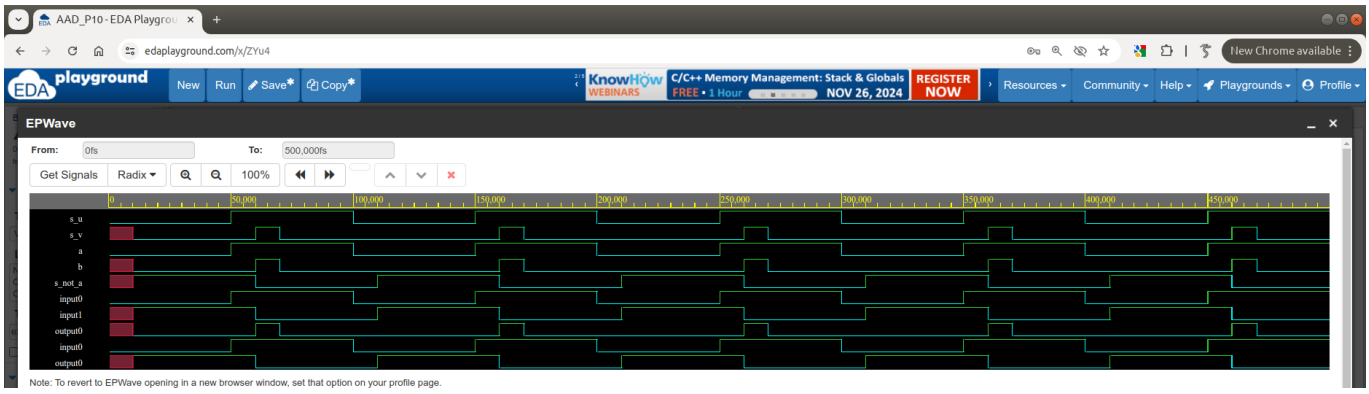
1. Press New.
2. On the left pane, in the `Design` box select VHDL.
3. In the `Libraries` box select None.
4. In the `Top entity` box write `ex1_tb`, which is the name of our test bench entity.
5. In the `Tools and Simulators` box select `GHDL 3.0.0`.
6. Write `--std=08` in the `Import Options`, `Make Options`, and `Run Options` boxes. This tells the `ghdl` program that we will be using the 2008 version of the VHDL hardware description language.
7. Select the `Open EPWave after run` option.

8. Press the + button on the right-hand side window to upload files. Upload `not_gate_1.vhd` and `and_gate_2.vhd`.
9. Copy the contents of `ex1.vhd` to the `design.vhd` file. (It looks like the `edaplayground` web page requires a fixed name for this file).
10. Copy the contents of `ex1_tb.vhd` to the `testbench.vhd` file on the left-hand side window. (It looks like the `edaplayground` web page also requires a fixed name for this file).

After all this you should see something similar to this:



Actually, this image was captured after running the simulation (click on the Run box near the top left side of the window (the bottom window pane already displays what was done to run the simulation). Do it now. After a few seconds the editor windows should be replaced by the following:



It this what you were expecting? To return to the editor window, press the small x on the top right-hand side of the waveform window. (It is possible to open the waveform window in a new window, you have to edit your profile in order to do so, but you must also allow popup windows in your browser).

2.2 Using ghdl and gtkwave

First, you need to install `ghdl` and `gtkwave`. Do it now. You may also download and compile `ghdl` yourself (github page).

After that, just run the following four commands (check the `makefile`).

```
$ ghdl -i --std=08 not_gate_1.vhd and_gate_2.vhd ex1.vhd ex1_tb.vhd
```

This command imports the given files into our project (called `work` by default).

```
$ ghdl -m --std=08 ex1_tb
```

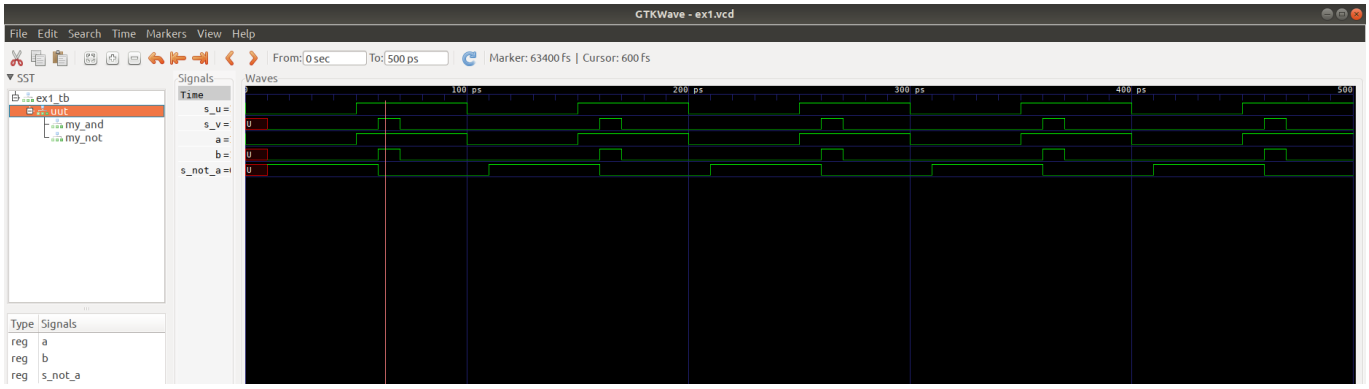
This one "makes" the project for a given top level entity.

```
$ ghdl -r --std=08 ex1_tb --stop-time=1000ps --vcd=ex1.vcd
```

This one runs the simulation, for a given maximum simulation time, and writes the waveforms on a given file.

```
$ gtkwave ex1.vcd
```

This last one displays the waveforms. You will have to select the signals to show and you will have to "zoom out". After some adjustments, you should see something like the following:



At the end, as usual, clean up:

```
$ rm ex1.vcd
$ rm work-*.cf
```

3 Second exercice, n-bit adder

The files

- `and_gate_2.vhd`
- `or_gate_3.vhd`
- `xor_gate_3.vhd`
- `full_adder.vhd`
- `adder_n.vhd`
- `ex2_tb.vhd`

contain the entire description of an n-bit adder (with a testbench top level, used only to generate all waveforms). Simulate it. What is the largest propagation delay of the adder? Try it also with 8, 16, and 32 bits.