Based on tutorial in Croatia Summer School given by Associate Professor Dr. Nele Mentens from KU Leuven.

PRE-TUTORIAL INSTALLATION GUIDELINES

1 Install GHDL

GHDL is an open-source command-line simulator for the VHDL language: `http://ghdl.free.fr`.
To install GHDL, follow these steps:

1.1 Linux users

- Step 1 : Make sure gnat and zlib are installed :
  (gnat:`https://gcc.gnu.org/wiki/GNAT`, zlib:`https://www.zlib.net/`)

  ```
  >> sudo apt-get install gnat zlib1g-dev
  ```

- Step 2 : Clone and compile ghdl from the official github repository:

  ```
  >> git clone https://github.com/ghdl/ghdl.git

  >> cd ghdl && ./configure --prefix=/usr/local && make

  >> sudo make install && make clean
  ```

1.2 Windows and Mac users

- Step 1 : Download the installation files from `https://github.com/ghdl/ghdl/releases` for the operating system of your choice. E.g., if you are using Windows, download `"ghdl-0.36-mingw32-mcode.zip"`. The following steps explain the setup for Windows; similar steps can be followed for other operating systems. In case you cannot find the version for your operating system, you can build it by following the mcode instructions at this link: `http://ghdl.readthedocs.io/en/latest/building/Building.html`
- Step 2 : Unzip the downloaded file.
- Step 3 : The folder `<download_location>\GHDL\0.36-mingw32-mcode\bin` contains the executable `"ghdl.exe"`. Add the location of this folder to the `"Path"` system variable such that you can use the `ghdl` command from anywhere. E.g., in Windows 7, you can do this by right-clicking the `Computer` icon (on your desktop, in the start menu, or in an explorer window) and going to `Properties` → `Advanced System Settings` → `Advanced` (tab) → `Environment variables`. Highlight the `Path` variable in the `System variables` section and click the `Edit` button. Add `<download_location>\GHDL\0.36-mingw32-mcode\bin;` to the path.

2 Install GTKWave

GTKWave is an open-source waveform viewer: `http://gtkwave.sourceforge.net`.

2.1 Linux users

As GTKWave is based on GTK+, you need to install it. *libcanberra* implements GTK modules: `http://0pointer.de/lennart/projects/libcanberra/`

```
>> apt-get install gtkwave libcanberra-gtk-module
```

2.2 Windows and Mac users

Download the installation files from `https://sourceforge.net/projects/gtkwave/files` for the operating system of your choice. Follow the same steps as for the setup of GHDL. Make sure to add the path to your global environment (e.g. for Windows, `<download_location>\gtkwave-3.3.100-bin-win64\bin` should be added in `Path`) such that the `gtkwave` command can be executed from anywhere.

3 Run an example program

To verify that GHDL and GTKWave are working correctly, we will run an example program by following these steps:

- Step 1 : Create two VHDL files: "invert.vhd" and "tb_invert.vhd". Copy the code below into these files.

invert.vhd

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

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

architecture arch of invert is

begin

b <= not a;

end arch;
```

tb_invert.vhd

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

entity tb_invert is
end tb_invert;

architecture arch of tb_invert is

signal a, b: std_logic;

component invert is
    port(
        a: in std_logic;
        b: out std_logic
    );
end component;

begin
inst_invert: invert
    port map(a=>a, b=>b);

p: process
begin
    a <= '0';
    wait for 10 ns;
    a <= '1';
    wait for 10 ns;
    wait;
end process;

end arch;
```

"invert.vhd" describes an inverter with input a and output b. "tb_invert.vhd" describes a testbench that applies test stimuli to the input: a is initialized to 0 and toggled to 1 after 10 ns. The simulation ends 10 ns later. We will elaborate more on the structure of these files during the tutorial.

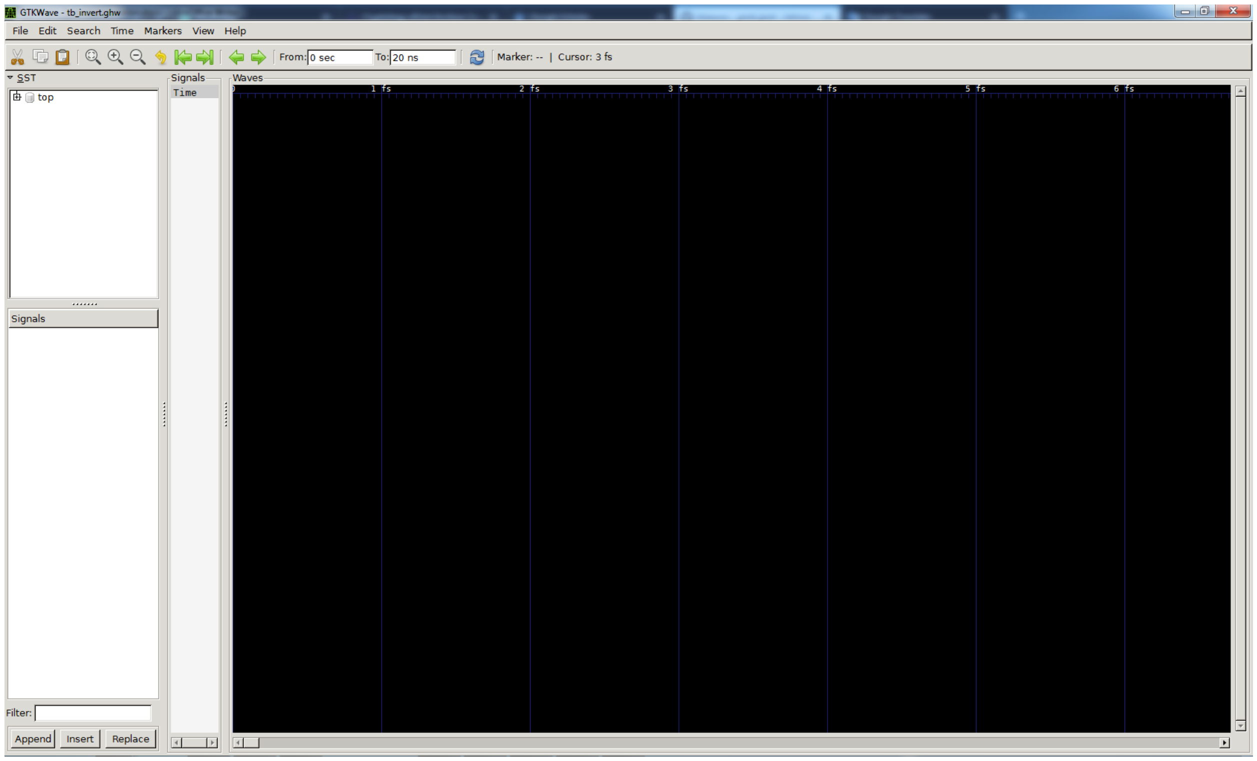
- Step 2 : Execute the following commands in a command/terminal window, after going to the directory in which the VHDL files are stored. First we analyze both files using ghdl -a, then elaborate on the testbench using ghdl -e, then run the testbench and specify the waveform output file using ghdl -r:

```
ghdl -a invert.vhd
ghdl -a tb_invert.vhd
ghdl -e tb_invert
ghdl -r tb_invert --wave=tb_invert.ghw
```

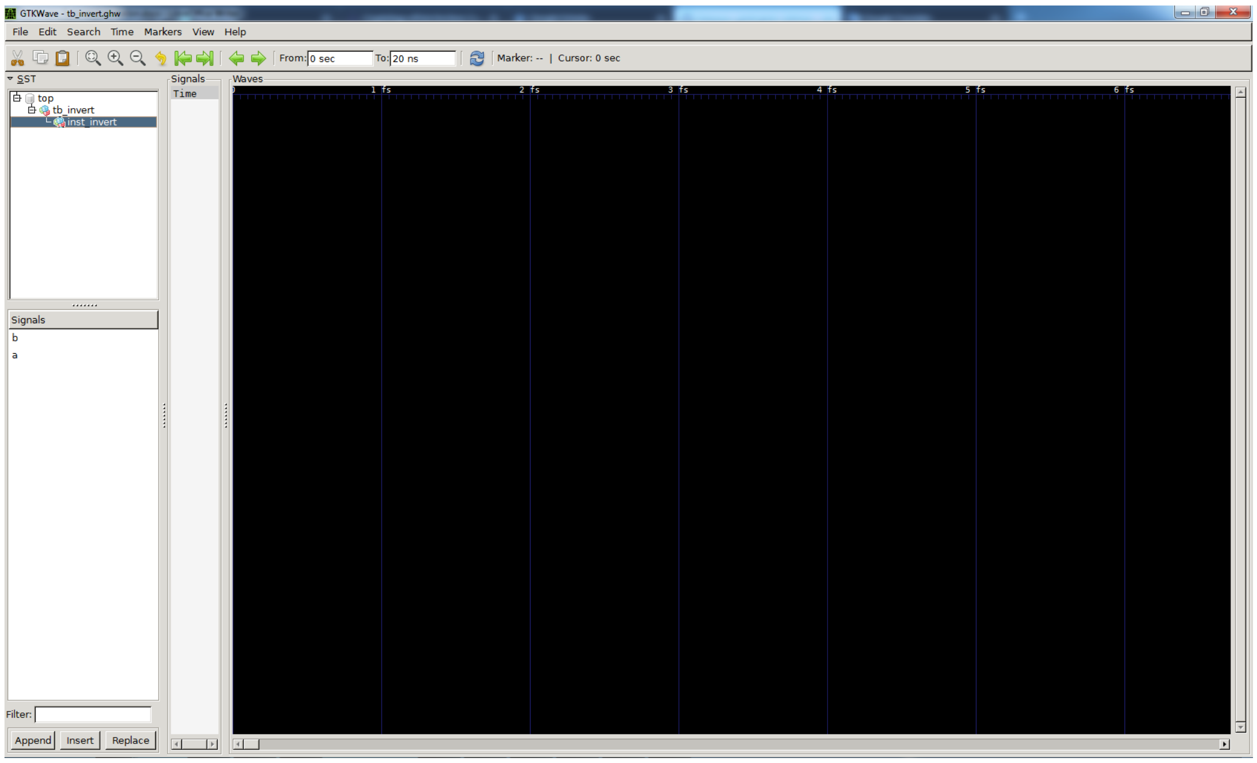
- Step 3 : Open a new command/terminal window in the same VHDL directory you performed the ghdl commands. Keep the previous window that you used with ghdl, it will be necessary on the next steps. In this command line we will open the waveform generated by ghdl with gtkwave:

```
gtkwave tb_invert.ghw
```

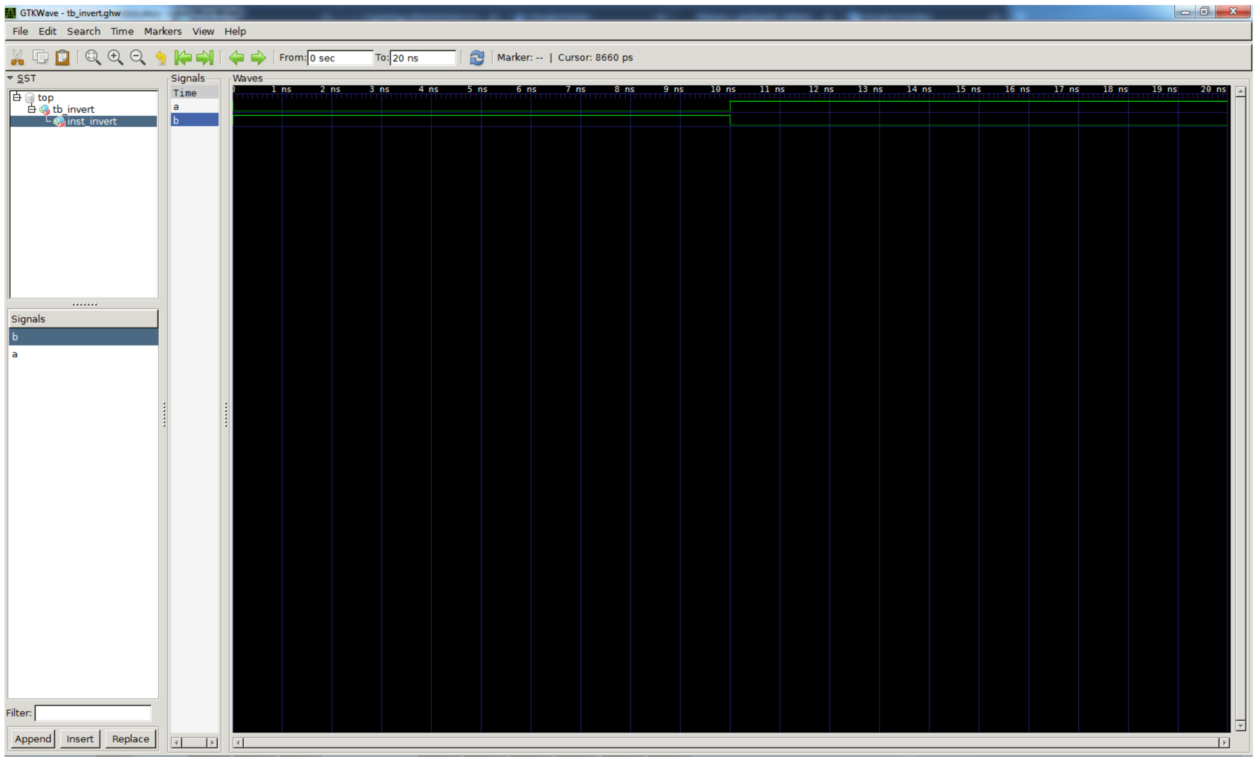
The GTKWave graphical user interface should open now, presenting this view:



- Step 4 : Expand the top module in the SST (Signal Search Tree) Window by clicking the + sign. Expand the tb_invert module by clicking the + sign. Select the tb_invert module such that you get this view:



- Step 5 : Select the signal a and drag it to the Signals Window. Do the same for signal b. Zoom to the full simulation view by using the Zoom Fit button . Finally, this view should be displayed:



Now the waveforms of b and a are shown over the total duration of the simulation, namely 20 ns. We see that a behaves as driven by the testbench, and b is the inverse of a.

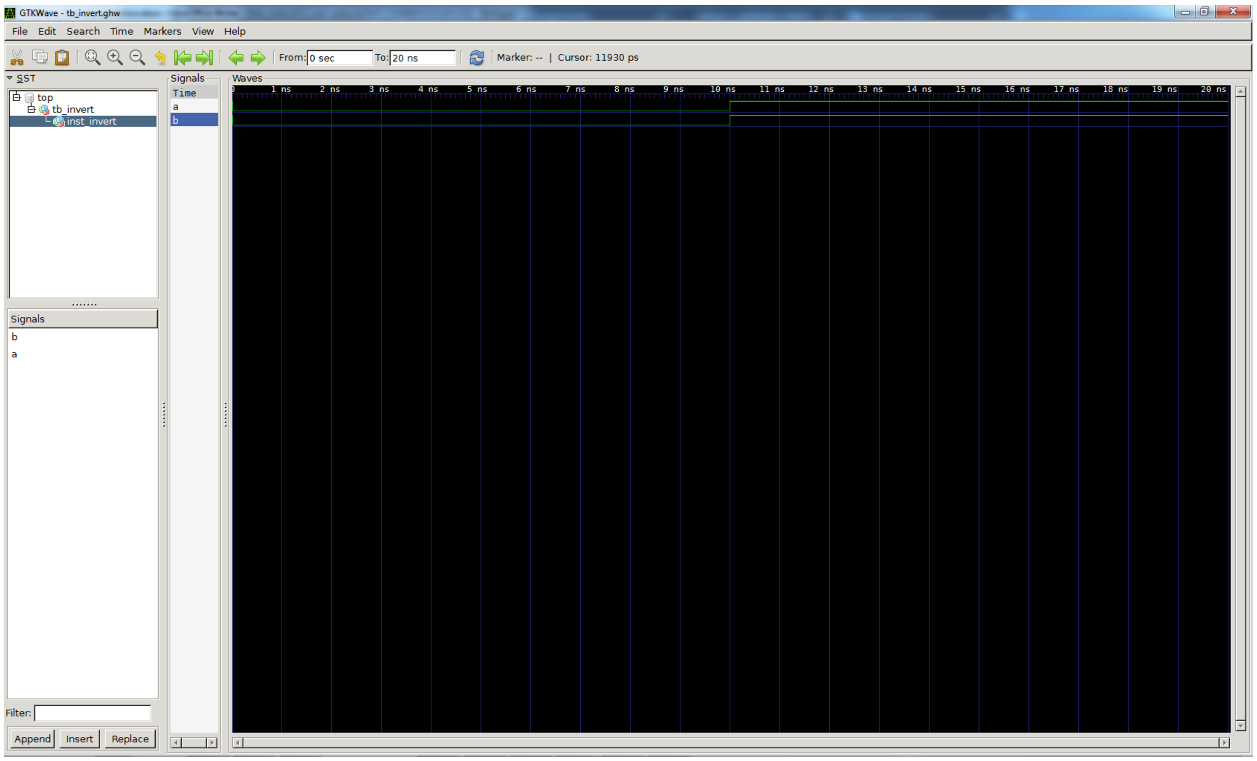
- Step 6 : We will now change the code and update the simulation in the same gtkwave window. Open the file invert.vhd, and remove the inversion. It should become:

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

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

architecture arch of invert is

begin

b <= a;

end arch;
```

Save the file, and run the same simulation commands as before:

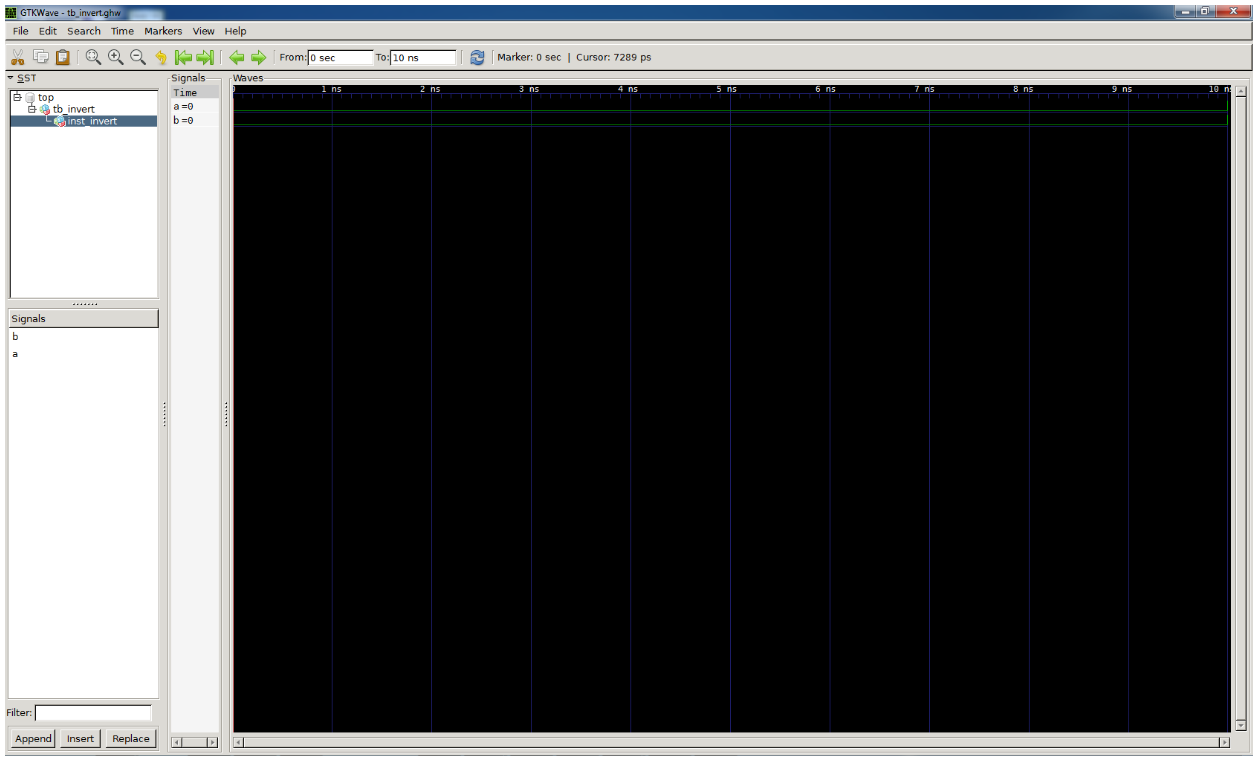
ghdl -a invert.vhd
ghdl -a tb_invert.vhd
ghdl -e tb_invert
ghdl -r tb_invert --wave=tb_invert.ghw

- Step 7 : Press the reload button and the waveform will be automatically updated:



This is very useful when debugging or updating implementations, as you do not need to open gtkwave again and add all signals. In this case the number of signals is only two, but as your design grows, there will be many signals to take into account.

- Step 8 : Redo the simulation with the extra command “stop-time”:
  ghdl -r tb_invert --wave=tb_invert.ghw --stop-time=10ns



The simulation now finished at 10ns instead of 20ns. Finishing a simulation earlier can be used to debug intermediate signals in a long simulation. Also, it can help when some modules might be locked in an infinite loop, therefore forcing the testbench to finish, since the testbench will never arrive at the end of it is execution.