1 Introduction

The work of an engineer doesn't stop after having found a *solution* to a given problem; he/she needs to be capable to show on different levels that the solution is *correct*. This holds for software development, but it is particularly important in hardware development. An error in a hardware component cannot be easily fixed when the product is deployed to the customer! When using hardware designed components normally a *testbench* is realized to validate the correct functionality of the hardware component. The functionality of a *testbench* is to stimulate the hardware component at its inputs and compares if the reaction on the outputs is the correct one. If one or more errors are found, they can be corrected in the simulation phase before starting the time-consuming implementation phase. This test is part of the *design flow* of a digital system. A simplified design flow for FPGA is shown in Figure 1. We will enter this design flow at HDL(=Hardware Description Language) level.

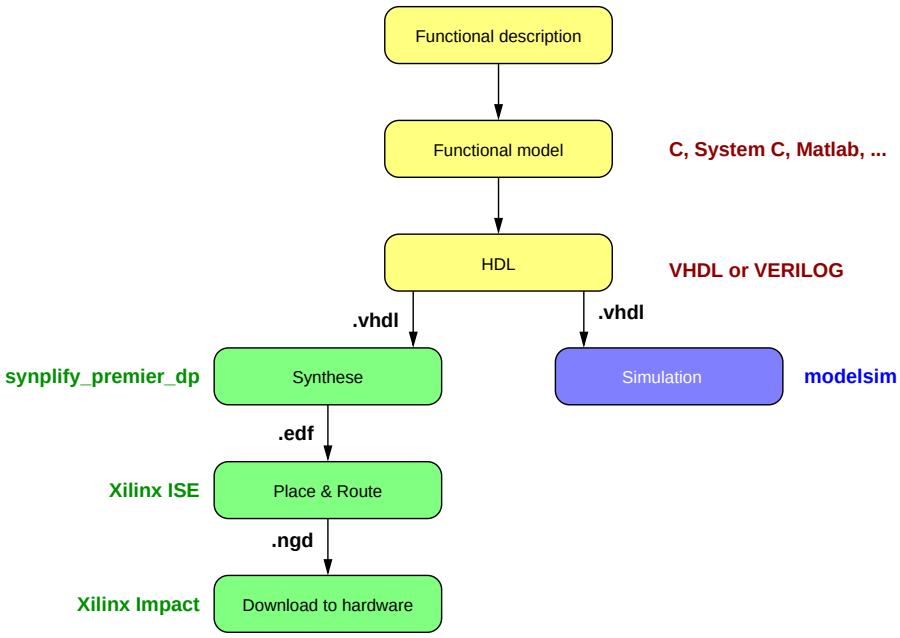


Figure 1: The design flow to get a digital system on an FPGA.

For simple circuits the *testbench* can be replaced by *manual simulation*. In this lab we are going to simulate a simple circuit in *modelsim*.

2 The simple circuit

The circuit below is a simple Finite State Machine (FSM).

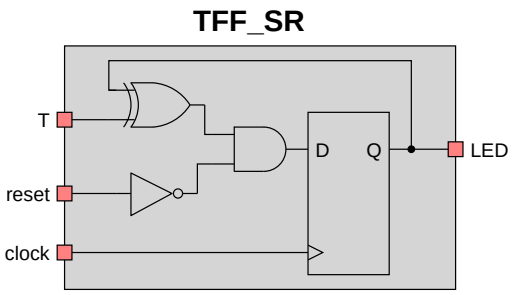


Figure 2: A simple Finite State Machine.

Exercise 1:

Draw the state diagram and the transition table of this FSM.

On moodle you will find the file `template.zip`. Unzip this file in a directory where the name doesn't contain any spaces (for example `My Documents` **is not** a good choice as the directory name contains a space). After unzipping the file you will find three directories, namely (1) `modelsim`, which is empty, (2) `scripts`, which is also empty, and (3) `vhdl`. In this last directory you will find two files, being (a) `tff_sr-entity.vhdl`, which is a *black box* view of the FSM, and (b) `tff_sr-behaviour-generic.vhdl` describing the functionality of the FSM in VHDL.

Exercise 2:

Open the two files and try to determine how the FSM of figure 2 is realized in VHDL. Try to mark all names used in the VHDL files into figure 2.

Now we are going to simulate all possible states of this FSM. To do this simulation we have to determine the sequence of the inputs of the FSM that forces the FSM in all possible states and take all transitions.

Exercise 3:

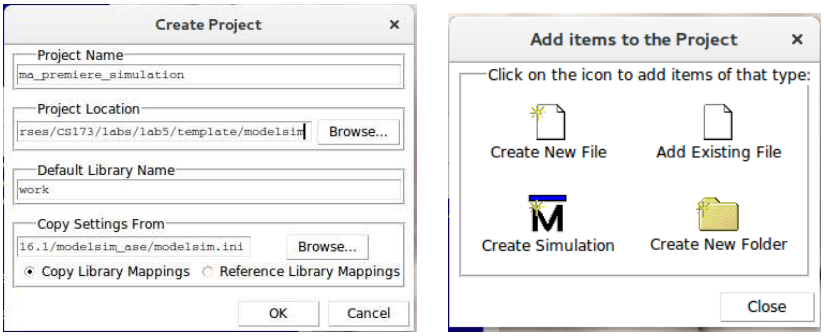
Determine the sequence of the inputs that forces the FSM in all possible states and takes all transitions; in other words: determine your test strategy.

IMPORTANT:

For this, and the next practical works you require *Intel Quartus Prime Lite edition* to be installed on your machine.

For the simulation we are going to use *modelsim*. Start *modelsim* by (1) clicking in Windows on the icon, or (2) in Linux to start a terminal and start the command `vsim`. As soon as modelsim starts, create a new project by:

1. Create a new project through `File > New > Project...`
2. Name your project `ma_premiere_simulation` under `Project Name`.
3. Press the button `Browse...` at `Project Location` and select the directory `modelsim` in the directory where you unzipped `template.zip`.
4. The window should now look like the one shown in figure 3a. Press the button `Ok` and the window shown in figure 3b should appear.



(a) Create a project (b) Add files to the project

Figure 3: Creating a project in modelsim.

5. Click on `Add Existing File` and add the files in the directory `vhdl` (first the file `tff_sr-entity.vhdl` and then `tff_sr-behavior-generic.vhdl`, The order is important as we first need the list of entities and that the behaviors).
6. Close the window shown in figure 3b by clicking on `close`. The main window should now look like figure 4.
7. Now, click first on `tff_sr-entity.vhdl` and than on the shown symbol in the sequence as indicated in figure 4. After this sequence the command window should not list any errors.

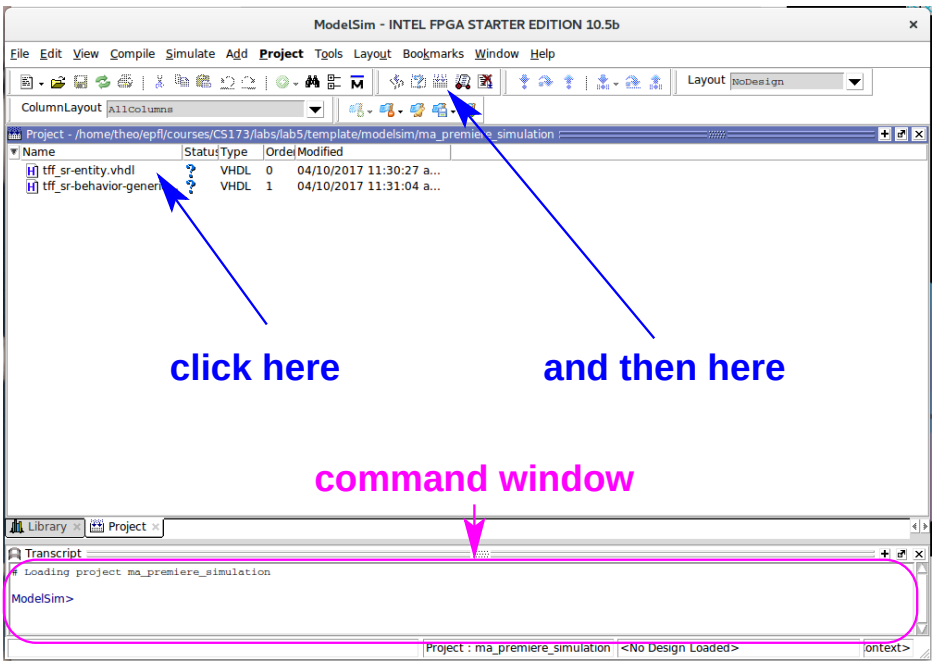


Figure 4: The main window in modelsim.

From now on we are going to use the command line of modelsim in it's **command window** (`Transcript` window in figure 4).
Start the simulation with the command: `vsim -t ps tff_sr`
In this command the command line argument `-t ps` indicates with which time resolution we want to work (in this case pico seconds). After this command line argument we indicate which entity we want to simulate (the entity which is named `tff_sr`).
We open the timing diagram with the command: `add wave *`
Before starting the simulation we have to define the inputs of the system. To define the levels, or the sequences in time of the inputs of a system we use the command `force`. The syntactical definition of this command is:
`force <SignalName> <Sequence>`
Some examples for the `force` command are shown in figure 5. In the first example an input is forced constantly to 1. In the second example a small sequence is realized. Finally in the third example a repeating sequence is realized.

As soon as we have defined all sequences for the inputs of the system with the `force` command we can start the simulation with the `run` command. The `run` command takes one command line argument being the time to simulate. Hence `run 500` will simulate for 500 units of time (remember: one

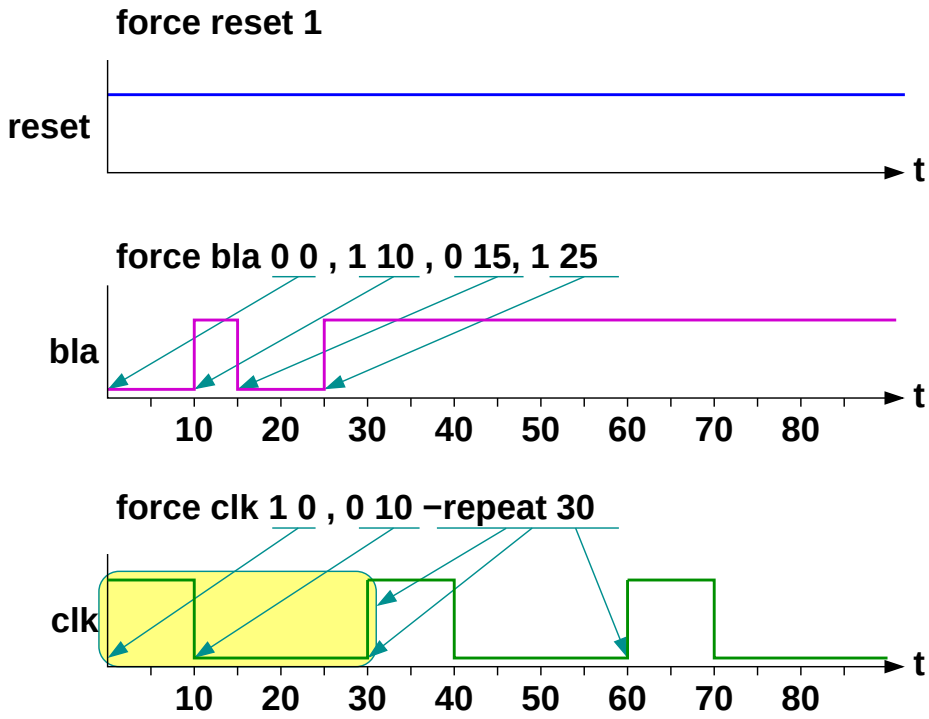


Figure 5: Examples for the `force` command.

unit of time is a pico second in our case, as we started the simulation with the command line argument `-t ps`).
Note: After the execution of the `run` command, the simulator supposes the current point in the timing diagram as being `t=0s`!

Exercise 4:

Execute a simulation of the system such that all states and transitions of your FSM are excited.

IMPORTANT:

Each time you want to restart your simulation you have to restart your simulation with the `vsim` command and redo all the following steps (`add wave ...`, `force ...`, `... `, `run ...`).

3 The automated simulation

Each time you make a change to your system you have to redo all the commands in modelsim to perform a simulation. This is very time consuming and very prone to errors. It's more intelligent to automate the simulation by using scripting. This script contains all the commands you typed before in the command window.

Exercise 5:

After having done one successful simulation of your finite state machine, click on `File > Save transcript as...` and store the transcripts in the directory `script` as `tff_sr.do`. Open the file `tff_sr.do` with a text editor and remove all lines starting with `#`.

Modelsim stores all executed commands in a transcript file. As we store this transcript we have exactly the same command as we typed before.

Exercise 6:

Do a new simulation by executing the command:
`do ../script/tff_sr.do`

Close modelsim.

Exercise 7:

Change in the file `tff_sf-behaviour-generic.vhdl` the line where the `XOR` is described. Replace the `XOR`-gate by an `OR`-gate. Save the file. Start modelsim. Perform the sequence shown in figure 4. Start an automated simulation with the command: `do ../scripts/tff_sr.do` What can you observe?