Tutorial Modelsim

Simulations of logic circuits using Modelsim

Description: Use the Modelsim tool to perform several simulations of logic circuits described in Verilog HDL at different levels of abstraction.

Purpose: To familiarize with the Modelsim development tool and the Verilog HDL hardware description language.

Expected learning:

* Describing a Simple Logic Circuit Using Verilog HDL
* Circuit simulation in an interactive way (step by step)
* Creation of a stimulus file and description of the same through the HDL language
* Using the stimulus file for a batch simulation
* Understanding the difference between behavioral, RTL, and structural descriptions.
* Realization of simulations both at RTL and Gate level by recalling Modelsim from within the Quartus program

# Introduction to ModelSim

Modelsim [1] is a simulation tool developed by Mentor-Graphics [2] that has been adopted as a "state of the art" within several EDA (Electronic Design Automation) software dedicated to the design of electronic circuits. So it is quite common that the producers of a certain development system (in our case Altera) provide interfaces to software developed by third parties such as ModelSim for Logic Simulation, Synopsys for HSpice synthesis for analog simulation and signal analysis, ...

In this tutorial we will analyze the basics for a simulation of a logic circuit described in VerilogHDL using ModelSim and how it can interface with the Quartus II environment [3-4-5].

# Introduction to the problem

Want to simulate the circuit shown below

.

This is an asynchronous sequential circuit that can be affected by static hazard [6]. This involves an interesting criticality: in practice, the operation is strongly related to the delays in the propagation of the signals through the various paths and it will be possible to notice that by altering these, the function of the circuit itself is modified.

The operation of the circuit is described by this flow table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| st\x1x2 | 00 | 01 | 11 | 10 |
| 0 | **0** | **0** | 1 | **0** |
| 1 | **1** | 0 | **1** | **1** |

Suppose that at power-up both input signals x1 and x2 are at 0. In this case, the output could assume both the value 0 and the value 1 indifferently and self-sustain it through the reaction loop (in other words, at the time of switching on the circuit there is no way to determine whether the output is at 0 or 1). However, if the signal x2 is raised, the output takes on the value 0 and this is maintained even if later x2 lowers to return to the initial conditions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| st\x1x2 | 00 | 01 | 11 | 10 |
| 0 | **0** | **0** | 1 | **0** |
| 1 | **1** | 0 | **1** | **1** |

Conversely, if after powering up with both inputs at 0 the signal x1 is raised, the output takes on the value 1 and maintains it even when x1 should go down again.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| st\x1x2 | 00 | 01 | 11 | 10 |
| 0 | **0** | **0** | 1 | **0** |
| 1 | **1** | 0 | **1** | **1** |

If both signals are at high level, the output takes on a value of 1, a value that should be maintained even when x2 goes down. However, it can be noted that depending on the propagation times, a wrong transition could also take place and the output could go to the low state. In practice if $∆i+∆2-∆1>∆3$ , if the output will change to the value 0, otherwise the transition will be correct and the output will remain in the high state.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| st\x1x2 | 00 | 01 | 11 | 10 |
| 0 | **0** | **0** | 1 | **0** |
| 1 | **1** | 0 | **1** | **1** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| st\x1x2 | 00 | 01 | 11 | 10 |
| 0 | **0** | **0** | 1 | **0** |
| 1 | **1** | 0 | **1** | **1** |

**It is therefore essential to be able to simulate the circuit by analyzing in detail the propagation times through the various elements of the circuit.**

## Procedure

Through a common text editor, create a file written in Verilog HDL that describes (at a behavioral level) the circuit reported above with the (manual) introduction of any delay times in the various elements and save it in an appropriate directory.

The file in question could be the following:

'Timescale 1ps/1ps

module alea (x1,x2,y);

input x1,x2;

output y;

wire i1,i2,i3;

assign #20 i1=~(x1 & x2);

assign #20 i2=~(i3 & y);

assign #30 i3=~x2;

assign #20 y=~(i1 & i2);

endmodule

1. Compiling the source

Open the Modelsim tool either via the icon on the desktop or from the list of installed programs

Create a new project by specifying its name and the location of the directory in which it will reside. Also keep the default name "work" of the library to rely on to reside the compiled files. Copy the default configuration file either from the installation directory or from the last project you did.



Add the developed file "add existing file" to the project



> OK

> Close

Compile the file so that it becomes part of one of the various libraries available. By default, it is good practice to use the "work" library (which will contain all the modules of the project you are working on).

*Fill > Fill in All*

Any syntax errors will be highlighted at this stage. If there are no errors in the "Work" library, the "alea" module will appear.

If necessary, to make the relative card appear, check that the card relating to library management is active

View > Library (alt-vu)

1. Step-by-step simulation

In the Library window, right-click on the random module (inside the work library) and select **Simulate.**

A new window configuration appears oriented to the management of the signals in the simulation phase.

Make sure that the Wave Signal Graph Display screen is visible

View > Wave (alt-vv)



Drag the instance named "alea" from the sim window to the wave window. Note that all signals related to this block, including internal signals, appear.

In the Console window, type

> run 100

The system performs the first 100 simulation steps (equivalent in our case to 100 ps)



Note that initially the input signals are not defined (highZ) and therefore the output signals as well as the internal signals are interdetermined (StX).

Now force the input signals to logic level 0.

* Right click on the /alea/x1 signal in the Wave window
* Force...
* In the window that appears, write the value 0 in the Value field
* OK



Repeat the procedure for signal x2 as well

In the Console window, type

> run 100

Now some signals assume the correct logical value after a certain time interval compatible with the delay times introduced during the circuit description phase, but others (y and i2) remain in the "StX" (indefinite) state. This is due to the very nature of the circuit, in fact both logical values (0 and 1) would be compatible with the operation and there is no way, as mentioned in the introduction, to discriminate in which of the two states the system is located.



Repeating the procedure illustrated above, try to raise and lower the signal x2, following each change with an operating period of 100ps.



Note that with this procedure the system has brought the output into a well-defined state (state 0) and that it remains in this state even when the input signals have both returned to state 0.

Now try to assign the sequence of stimuli (relative to the signals x1 and x2) 00 – 01 – 11 – 10 following the procedure suggested above, following each stimulus with a sufficient simulation period (e.g. 100ps) to ensure that all delays have been extinguished.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| st\x1x2 | 00 | 01 | 11 | 10 |
| 0 | **0** | **0** | 1 | **0** |
| 1 | **1** | 0 | **1** | **1** |



It can be seen that the output correctly moves to the high value corresponding to the value 11 for the inputs, but at the passage to 10 instead of remaining at the high value as provided for in the flow table, due to the presence of a hazard it returns to the value 0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| st\x1x2 | 00 | 01 | 11 | 10 |
| 0 | **0** | **0** | 1 | **0** |
| 1 | **1** | 0 | **1** | **1** |

In fact, it can be seen that if the inverter delay is reduced from the current 30ps to 10ps, the risk does not occur and the operation will be different. To carry out this simulation we will adopt a different method and it is more intuitive.

1. Batch simulation through definition of the f.d.o.
* In the Library window, right-click on the "random" instance and choose Edit.
* In the text box that appears, change the delay of the reverser from #30 to #10

. . .

assign #20 i1=~(x1 & x2);

assign #20 i2=~(i3 & y);

assign #**10** i3=~x2;

assign #20 y=~(i1 & i2);

. . .

* Save
* Again by right clicking on "Recompile"
* and then "Simulate"
* Still in the Library window (possibly present under the "sim" window) with a right click on the "alea" instance choose "Create Wave".

In the Wave window now appear 3 signals (x1,x2 and y)

* Delete the y signal (right click > Edit > Delete)
* Edit the x1 and x2 signals so that they take the constant value 0 between 0ps and 1000ps:
	+ Right click > Edit > Create/Modify Waveform
	+ In the window that appears, choose
		- Constant,
		- Start Time=0
		- End Time=1000
		- Time Unit ps
		- Next



* + In the next window
		- set the value to 0
		- Finish
* Repeat the procedure for the signal x2 as well
* Change how the mouse works
	+ Wave > Mouse Mode > Edit Mouse
* Use the mouse to highlight a portion of the signal x2



* Change the value of this slice

Wave > Wave Editor > Invert

Optionally, you can use the "Edit Toolbar" to have buttons available to edit signals more quickly

Window > Toolbars > Wave Edit

* Use this technique to fully define the f.d.o. with which to stimulate the circuit
* Drag and drop all the signals you want to monitor from the "Sim" window to the "Wave" window



* In the console window, type
	+ Run -all



It can be seen that with a different delay of the inverter the phenomenon of the hazard no longer occurs and the circuit works correctly.

If you want to simulate multiple circuit solutions through the same stimuli, you can save both the stimuli and the unit to be tested in a Verilog HDL file.

* Make sure that the "Wave" window is highlighted (if necessary by clicking on it)
* File > Export > Waveform
	+ Choose the Verilog Testbench format
	+ The duration of the simulation
	+ The name of the file and the directory in which to save it



* + OK

It is advisable to open the newly generated file with a text editor and analyze its syntax that describes its behavior and how the stimuli are provided to the UUT (Unit Under Test). This is a file that uses the Verilog HDL language to create a "behavioral" description of the complete system (UUT + stimulus generator) and that evidently does not have to be synthesized nor would it be possible to synthesize.

1. Simulation of a complete testbench described in Verilog HDL

The file generated in the previous step contains both the definition of the stimuli and how they are brought to the unit to be tested (UUT). Therefore, it, together with the Verilog HDL description of the unit to be tested, can be used to regenerate the simulation or possibly can be modified to generate different tests.

Open Modelsim or if the latter is already open, last the simulation in progress:

* Simulate > End Simulation

Import the file in question into the current project: by referring to the Project window (possibly opening it if it is closed)

View > Project (Alt - vx)

Project > Add To Project > Existing File

At the moment the file (in its text version) is part of the project, but in order to use it, it must be compiled. Therefore, highlight the file by clicking on it

 Compile > Compile Selected (Alt – ce)

Now within the work library there are two modules: the file that creates the testbench and the file that describes the circuit under test.

* Right-click on the test bench module and choose Simulate
* In the "Wave" Window drag the signals you want to analyze (for example, you can drag the entire DUT instance contained within the testbench to view all the signals that refer to it)
* In the Type Console window
	+ Run –all

Now it is possible to modify the Verilog HDL source that describes the circuit to be analyzed by modifying delays and/or connections and after a recompilation the simulation can be relaunched with the same stimuli and the results can be verified.

For example, the circuit could be modified by introducing a redundant path to cancel the effects of the hazard:



This circuit could be described in Verilog HDL with the following source:

'Timescale 1ps/1ps

module alea (x1,x2,y);

input x1,x2;

output y;

wire i1,i2,i3,i4;

assign #20 i1=~(x1 & x2);

assign #20 i2=~(i3 & y);

assign #30 i3=~x2;

**assign #20 i4=~(x1 & y);**

assign #20 y=~(i1 & i2 **& i4**);

endmodule

It is therefore possible to

* Stop the current simulation
* Re-edit the original "alea.v" module
* Recompile the new “alea” module
* Relaunch the simulation
* Define the signals to be displayed
* Let the simulation run for a while.



The results show that although the delay time of the inverter can be critical, the introduction of the new path guarantees the absence of randomness and a correct functioning of the reaction.

1. Simulation within Quartus

#### Introduction:

Modelsim can also be called directly within the Quartus development environment, but to ensure the interaction between the two environments you need to make a priori settings:

Following the path

Tools > Option > General > EDA Tool Option

Define the directory where the modelsim executable resides. By default it is located in:

C:\altera\13.0sp1\modelsim\_ase\win32aloem



In addition, following the

Assignment > Settings > EDA Tool Settings

Define how

* simulation tool: ModelSim – Altera
* Format: Verilog HDL
* Run Gate Level simulation automatically – Off

In addition, following the

* Assignment > Settings > EDA Tool Settings > Simulation



* In the **Tool name**, select **ModelSim-Altera.**
* Make sure **that Run gate-level simulation automatically after compilation** is turned off.
* In the **EDA Netlist Writer settings** section, for the **Format for output netlist**  field, select **Verilog HDL.**
* Make sure that the **Map illegal HDL characters, Enable glitch filtering**, and **Generate Value Change Dump (VCD) file script**  fields are disabled
* In the **NativeLink settings** selection, select **None**.

#### RTL-level simulation

Let's suppose that within Quartus you want to carry out the simulation of a project carried out in the form of a schematic such as the one shown below.



Since Modelsim is a simulator for hard description languages (such as Verilog HDL or VHDL) the first step must be to convert our schematic into an equivalent file described in Verilog HDL.

By verifying that you have open the windows with the schematic view

File > Create/Update > Create HDL design File From Current File

Choose language (Verilog HDL) and the target file and click OK



The corresponding file is generated and can be imported into the project instead of the schematic.

Project > Add/Remove Files in project

And use the interface to remove the schematic from the project (Remove) and include the newly generated Verilog HDL file (Add).



In the Project Navigator section, in the Files tab, right-click on the file you just imported and choose "Set as Top Level Entity" (Crt-Shift-J).

Processing > Start > Start Analysis and Synthesis (ctl-k)

Therefore

Tools > Run Simulation Tool > RTL simulation

The Modelsim simulator opens and within it you can proceed as described in paragraph 3:

* Start the simulation of the instance to be analyzed (right click > Simulate)
* Generate the test waveforms (right click > Create Wave)
* Eliminate unused signals
* Through the graphic tool, draw the appropriate f.d.o.
* Drag the signals you want to display into the wave window
* In the Console window, launch a "run"

A few notes:

The internal signals take on names that are difficult to interpret because the Schematic to Verilog conversion system has assigned them names by default.

The simulation stops with an error right in the presence of the transition from 11 to 10. In fact, as seen above, the behavior of the circuit depends essentially on the delays of the circuit, but in this first phase of the design in which the delays are not yet defined (i.e. they are all idealized at 0), the system has no way of determining whether or not there will be a hazard.



It is therefore proposed to save the f.d.o. and carry out a second simulation at a lower level of abstraction, i.e. at the gate level.

File > Export > WaveForm

And choose the name and location where to save the file (in Verilog HDL format).

Close the simulation tool momentarily.

By exploiting the stimulus file just saved, automatic simulations can be performed without having to redefine the trend of the inputs from time to time. To do this, you need to change the settings within Quartus:

Assignments > Settings

And within the tab

Eda Tool Settings > Simulation

Activate **Compile Test Bench** and click on the **Test Benches button**

In the Test Benches window, click New

In the new window define

* A mnemonic name for the bech test that is going to be defined
* IMPORTANT: The name of the Top Entity as it appears in the stimulus file (usually coincides with the name of the stimulus file itself)
* Add the stimulus file you saved in the previous step to the list of files you want to use

Note: The three default names are usually the same



> OK

> OK

> OK

Now the same stimulus file can be adopted to quickly and automatically perform simulations at different levels of abstraction.

#### GATE-level simulation

Fill in the circuit up to the fitting level:

Processing > Start Compilation (ctr-L)

Relaunch the simulation at the gate level

Tools > Run Simulation Tool > Gate Level Simulation

Choose a behavioral model (Fast/Slow) and complete the simulation.



From the results it can be seen that the behavior is not exactly what could be expected and the exit always remains indefinite. This behavior stems from the fact that the simulation now takes into account all propagation delays within the FPGA, and probably the pulse duration on the inputs is not enough to switch the circuit.

Therefore, try to redefine a different stimulus file where the signals are stable for a few tens of ns.

To do this, you can either start again from the RTL simulation or you can re-launch Modelsim but without combining it with the stimulus file: Close ModelSim, then return to the settings tab and deactivate the use of the Test Bench.

Assignments > Settings

Eda Tool Settings > Simulation and in the "Native Link Settings" section choose "None"

Then relaunch the simulation at the gate level.

Tools > Run Simulation Tool > Gate Level Simulation

The procedure for creating the f.d.o. and saving them has already been analyzed previously, but in the case in question there is a further procedure to be followed during the simulation phase.

In fact, in order to perform the circuit simulation at the gate level, the simulator must have information on the gates used, therefore links must be created with some libraries where the parameters and delays associated with the specific FPGA that is used are described in detail. To do this, the simulation must be launched from the toolbar:

Simulate > Start Simulation

* In the "Design" tab, define the unit to be simulated (which is typically inside the work library)



* In the Libraries tab, add the library for the device you are using (if you are using a Cyclone II, opt for the "cicloneii\_ver")



* In the timing sheet (SDF) integrate with the .sdo file that was generated during compilation



Now you can complete the simulation.

> OK

At this point, following the procedure already illustrated above, you can

* define the f.d.o. of the stimuli paying attention that they have a duration of the order of "ns" instead of "ps" as done so far (for which a simulation duration of about 100 ns is expected)
* add in the wave window the signals you want to display (taking into account that these will be signals internal to the FPGA born from the synthesis process and that are different from the signals present in the schematic from which we started.
* Run the simulation for enough time.



The response of the circuit is now consistent and the delay times related to the propagation of signals in the FPGA are highlighted. The stimulus file can now be saved and reused through the procedure described above to carry out other simulations.

**NOTE:** since the "timescales" of the automatically generated Verilog files are expressed in ps, it is always good to refer to this unit to avoid confusion between the various files. For example, when the duration information has to be passed in during the saving phase of the stimulus file, it is good that this is expressed as 100,000 ps rather than as 100 ns. Otherwise the generated stimulus file will stop after 100 units of time, which from its point of view is equivalent to 100 ps.

# Bibliography

[1] Modelsim Home page
[*http://www.mentor.com/products/fv/modelsim/*](http://www.mentor.com/products/fv/modelsim/)

[2] Mentor-Graphics Home page
[*http://www.mentor.com/*](http://www.mentor.com/)

[3] ENEE 359a: Digital VLSI Circuits by B. Jacob – "**Project 1: ModelSim Tutorial and Verilog Basics**"
[*http://www.ece.umd.edu/class/enee359a.S2008/p1.pdf*](http://www.ece.umd.edu/class/enee359a.S2008/p1.pdf)

[4] Altera Corporation: "**Introduction to Simulation of Verilog Designs Using ModelSim Graphical Waveform Editor**" *ftp://ftp.altera.com/up/pub/Altera\_Material/9.1/Tutorials/Verilog/ModelSim\_GUI\_Introduction.pdf*

[5] Altera Corporation: "**Using ModelSim to Simulate Logic Circuits in Verilog Designs"***ftp://ftp.altera.com/up/pub/Altera\_Material/13.0/Tutorials/Verilog/Using\_ModelSim.pdf*

[6] Antonio D'Amore: "**Lecture notes of the course of Logical Networks**"
<http://www2.units.it/marsi/reti/dispense/capitolo%2006.pdf>

 [7] Altera Corporation: "**Introduction to Simulation of Verilog Designs Using ModelSim Graphical Waveform Editor**"
ftp://ftp.altera.com/up/pub/Altera\_Material/13.0/Tutorials/Verilog/ModelSim\_GUI\_Introduction.pdf

[8] Mentor Graphics: **"ModelSim Reference Manual - Software Version 10.1a**"
<http://moodle.units.it/moodle/file.php/670/Manuals/Modelsim/ModelSim_Reference_Manual_v10.1c.pdf>

[9] Mentor Graphics: **"ModelSim User Manual - Software Version 10.1a**"
<http://moodle.units.it/moodle/file.php/670/Manuals/Modelsim/ModelSim_Users_Manual_v10.1c.pdf>

[10] Mentor Graphics "**ModelSim Tutorial -Software Version 10.1a**"
<http://www.fatih.edu.tr/~onur/download/EEE546/modelsimTutorial.PDF>

[11] Larbi Boughaleb "**ModelSim Tutorial**"
<http://www.ece.northwestern.edu/~ismail/courses/c92/ModelSim_Tutorial.pdf>

[12] "**ModelSim Tutorial**"
<http://ee.hawaii.edu/~sasaki/EE361/Fall06/Lab/Lab4.1/ModelSim.pdf>