MAVEN SILICON

Centre of Excellence in VLSI

Advanced Verilog

Lab manual

www.maven-silicon.com

Maven Silicon Confidential

All the presentations, books, documents [hard copies and soft copies] labs and projects [source code] that you are using and developing as part of the training course are the proprietary work of Maven Silicon and it is fully protected under copyright and trade secret laws. You may not view, use, disclose, copy, or distribute the materials or any information except pursuant to a valid written license from Maven Silicon

Table of Contents

Lab Instructions

1. The recommended editor is vi or gvim editor
2. Mentor Graphics Questasim_2019 tool is used to run the simulation.
3. The following directory structure is followed for all the lab exercises:

sim/	- contains make file to run the simulation
rtl/	- contains DUT RTL code
tb/	- contains self-checking testbench
solution/	- contains solutions for testbench

4. The labs are copied inside the respective user's home directory i.e. **/home/<user_name>**
5. Here, **$HOME** is used to represent **/home/<user_name>**
6. The simulation process involves different steps such as:
   a. Creating the physical library & mapping it with logical library
   b. Compilation
   c. Elaboration
   d. Simulation
7. Following are the Questa commands used for Batch mode simulation:
   a. vlib – To create a physical working library
   b. vmap – To map logical library with physical library
   c. vlog – To compile Verilog files
   d. vopt – To optimize the design
   e. vsim – To load the design into the simulator
8. We use the makefile to run all the above commands
9. The targets in makefile can be used for Compilation, simulation, deleting certain log files, etc.
10. Use "**make help**" to understand various targets that can be used in each lab exercise.
11. For any technical support to do the lab exercises, please reach out to us on tech_support@maven-silicon.com

Lab - 1: Verification of 4x1 multiplexer

Objective : *Learning instantiation and simple stimulus generation*

Working Directory : $HOME/VLSI_RN/Advanced_verilog/lab1/mux4_1/tb

Source Code : mux4_1_tb.v

Instructions : The following instructions have been included in the source code. as comments. Refer to the comments in the source code and edit the source code.

- ✓ Instantiate the Design using name based port mapping.
- ✓ Write initial block for stimulus generation.
- ✓ Within initial begin
  - Initialize inputs to 0.
  - Use nested 'for' loop for generating stimulus for inputs.
  - Use $finish task to finish the simulation at 1000ns.

Simulation Process :

- ✓ Go to the **sim** directory
- ✓ Call the target run_test to run the simulation: **`make run_all`**
- ✓ Observe the output.

Learning outcomes :

How to verify Multiplexer using command mode of simulation?

Lab - 2: Verification of DFF

Objective : *Learning how to write selfchecking testbenches using tasks*

Working Directory : $HOME/VLSI_RN/Advanced_verilog/lab2/dff/tb
Source Code : tb_dff.v
Instructions : The following instructions have been included in the source code as comments. Refer to the comments in the source code and edit the source code.

- ✓ Instantiate the dff design.
- ✓ Understand the constants used for defining setup, hold time & clock period.
- ✓ Write clock generation logic for a clock with period of 100ns.
- ✓ Define the task 'sync_reset' for resetting the dff.
- ✓ Define the task 'load_d0' and 'load_d1' for loading input values.
- ✓ Within the tasks
  - Inputs are driven within Tsetup before the active posedge of the clock.
  - Outputs are sampled after the Thold time after the active posedge of the clock.

Simulation Process :

- ✓ Go to the **sim** directory
- ✓ Call the target run_test to run the simulation: `make run_all`
- ✓ Observe the output.

Learning outcomes :
Understand how to write self-checking TestBench for a DFF RTL design.

Lab - 3: Code-coverage analysis for a coin collector

Objective : *Analysis of Code Coverage for a Coin Collector design*

Working Directory :
$HOME/VLSI_RN/Advanced_verilog/Code_coverage/lab3/vending_machine/tb

Source Code : tb_coincol1.v, tb_coincol2.v, tb_coincol3.v

Instructions : The following instructions have been included in the source code as comments.

- ✓ Understand the self-checking testbench concept used in the tasks.
- ✓ Within the tasks
  - Inputs are driven within Tsetup before the active posedge of the clock.
  - Outputs are sampled after Thold after the active posedge of the clock.
- ✓ Generate the code-coverage report for the RTL design, by simulating the testbenches one at a time.
- ✓ Edit the tb_coincol3.v file to achieve Statement, Branch, Condition,FSM state & transition coverage to be 100%.
- ✓ Compare the coverage reports generated by each TB and check which TB is a good quality TB.

Simulation Process :

- ✓ Go to the **sim** directory
- ✓ Call the target run_test to run the simulation: **`make run1, make run2, make run3`**
- ✓ Observe the output.

Learning outcomes :

Understand how to write self-checking TestBench for a RTL design and compare which one is a good quality Testbench.

Lab - 4: Code-coverage analysis for a ALU

Objective : *Analysis of Code Coverage for a ALU design.*

Working Directory :
$HOME/VLSI_RN/Advanced_verilog/Code_coverage/lab4/alu/tb

Source Code : tb_alu.v

Instructions : The following instructions have been included in the source code as comments.

- ✓ Understand the self-checking testbench concept used in the tasks.
- ✓ Understand the Input & Output file operations.
- ✓ Within the tasks
    - Inputs are driven within Tsetup before the active posedge of the clock.
    - Outputs are sampled after Thold after the active posedge of the clock.
    - File read operations are used to generate inputs for the operands.
    - The actual DUT outputs are compared with the values stored in an array which is also updated through the file read operation.
- ✓ Generate the code-coverage report for the design by simulating the testbench.
- ✓ Statement, Branch, Toggle coverage has to be 100%.

Simulation Process :

- ✓ Go to the **sim** directory
- ✓ Call the target run_test to run the simulation: `make run`
- ✓ Observe the output.

Learning outcomes :

Understand how to write self-checking TestBench for a RTL design and analyze the coverage report.

Lab - 5: Code-coverage analysis for an Arbiter

Objective : *Analysis of Code Coverage for an Arbiter design.*

Working Directory :
$HOME/VLSI_RN/Advanced_verilog/Code_coverage/lab5/arbiter/tb

Source Code : bfm_arbiter.v, tb_userinterface.v

Instructions : The following instructions have been included in the source code as comments.

- ✓ Understand the self-checking testbench concept used in the tasks.
- ✓ In the bfm_arbiter.v file,
  - Understand how the processors are sending requests to the arbiter for accessing the shared RAM slave.
  - Understand how the processors are sending data & address to the arbiter.
- ✓ In the tb_userinterface.v file,
  - Understand how the bfm_arbiter is driven by the tb_userinterface using the bfm_command.
- ✓ Generate the code-coverage report for the design by simulating the testbench.
- ✓ FSM state coverage & transition coverage has to be 100%.

Simulation Process :

- ✓ Go to the **sim** directory
- ✓ Call the target run_test to run the simulation: `make run`
- ✓ Observe the output.

Learning outcomes :

Understand how to write self-checking TestBench for a RTL design and analyze the coverage report.