SVA

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 4
Lab - 1 : Inline Assertions 6
Lab - 2 : Assertion Binding 7
Lab - 3 : Alarm Clock Case Study 8

Lab Instructions

1. The recommended editor is vi or gvim editor
2. The labs are copied inside the respective user's home directory i.e /home/user_name
3. Here $HOME represents /home/user_name
4. 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/ or env/ or env_lib	- contains verification environment, transactors & interface
test/	- contains testcases & top module
solution	- contains the solution source codes

5. Mentor Graphics Questasim_2019 or Synopsys – VCS tool can be used to run the simulation.
6. The tool can be selected using the command SIMULATOR = VCS/ Questa in the makefile

Questa :

The simulation process for Questa involves different steps such as:

a. Creating the physical library & mapping it with logical library
b. Compilation
c. Optimization
d. Simulation

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 & SystemVerilog files
d. vopt – To optimize the design
e. vsim – To load the design into the simulator
f. run – To run the simulation
g. qverilog – library creation, mapping, compile, and running simulation together

VCS :

The simulation process for VCS involves different steps such as:

a. Compilation / Elaboration
b. Simulation

Following are the VCS commands used for Batch mode simulation:

a. vcs – To compile Verilog & SystemVerilog files and generate an executable file (simv) which simulates the design.

7. We use the makefile to run all the above commands
8. The targets in makefile can be used for Compilation, simulation, deleting certain log files, etc.
9. Use “make help” to understand various targets that can be used in each lab exercise.
10. For any technical support to do the lab exercises, please reach out to us on techsupport_vm@maven-silicon.com

Lab - 1 : Inline Assertions

Objective : *To write the inline assertions to verify FIFO*

Main Working Directory: $HOME/VLSI_RN/SVA_LABS/FIFO_SVA/inline

Working Directory : duv_tb
Source Code : fifo.sv
Instructions :

- ✓ Understand the RTL provided (Note: There are some bugs inserted in DUT)
- ✓ Refer assertion written for RESET condition to write sequences and properties to verify the following conditions
  - FIFO Full
    - If fifo_status is 15, fifo is not full, and write signal is enabled next cycle full will go high
  - FIFO Empty
    - If fifo_status is 0, fifo is not empty, and read signal is enabled next cycle empty will go high
  - Underflow
    - If fifo is empty and only read signal is enabled next cycle underflow will go high
  - Overflow
    - If fifo is full and only write signal is enabled next cycle overflow will go high
  - Write pointer reset operation
    - If write signal is enabled and write pointer is 15 next cycle the write pointer resets back to 0
  - Read pointer reset operation
    - If read signal is enabled and read pointer is 15 next cycle the read pointer resets back to 0
  - Continuous writing
    - If write pointer is zero & if continuous write operation is done for sixteen times without read operation, next cycle the write pointer should go back to zero
  - Continuous reading
    - If read pointer is zero & if continuous read operation is done for sixteen times without write operation, next cycle the write pointer should go back to zero
  - Fifo status increment
    - If fifo status is not equal to 15, full is 0, and only write signal is enabled next cycle fifo status will increment
  - Fifo status decrement
    - If fifo status is not equal to 0, empty is 0, and only read signal is enabled next cycle fifo status will decrement
  - Write pointer increment
    - After every write operation write pointer should increment
    - Note: Take care of full condition

- Read pointer increment
  - After every read operation read pointer should increment
  - Note: Take care of empty condition
- Assert all the above properties

Simulation Process :

- ✓ Go to the directory **SVA_LABS/FIFO_SVA/inline/sim**
- ✓ Call the target run_test to run the simulation: **make run_test**
- ✓ Run the simulation in gui mode and observe the waveforms

Learning outcomes : How to write simple assertions to verify the FIFO

Lab - 2 : Assertion Binding

Objective : *How to bind the assertion module with the DUT module*

Main Working Directory: $HOME/VLSI_RN/SVA_LABS/FIFO_SVA/binding

Working Directory : duv_tb
Source Code : fifo_assertions.sv
Instructions :

- ✓ Understand the RTL provided (Note : There are some bugs inserted in DUT)
- ✓ Write sequences and properties to verify the following conditions
  - Reset
    - On reset overflow and underflow should be zero
  - Underflow
    - After fifo overflow if only is read is enabled continuously 17 times underflow should go high
  - Overflow
    - After reset if only write is enabled continuously for 17 times overflow should go high
  - Assert all the properties

Source Code : tb_fifo.sv
Instructions :

- ✓ Instantiate the assertion module and connect to DUT module using bind keyword

Simulation Process :

- ✓ Go to the directory **SVA_LABS/FIFO_SVA/binding/sim**
- ✓ Call the target run_test to run the simulation: **make run_test**
- ✓ Run the simulation in gui mode and observe the waveforms

Learning outcomes : How to connect assertion module with DUT using **BIND** keyword

Lab - 3 : Alarm Clock Case Study

Objective : *To write inline assertion and also bind the assertion module with the DUT module*

Main Working Directory : $HOME/VLSI_RN/SVA_LABS/Alarm_clock/Alarm_clock_Template

Working Directory : alarm_clock_assertions
Source Code : fsm_assertions.sv
Instructions :

- ✓ Understand the RTL provided (Note : There are some bugs inserted in DUT)
- ✓ Write sequences and properties to verify the following conditions
  - RESET
    - On reset the present state should be in SHOW_TIME
  - SHOW_TIME STATE
    - If Pre_state is SHOW_TIME and if alarm_button is 1 then next_state should be SHOW_ALARM or if key not equal to NOKEY then next_state should be KEY_STORED or next_state should be SHOW_TIME itself
  - KEY_STORED STATE
    - If Pre_state is KEY_STORED then next_state should be KEY_WAITED
  - KEY_WAITED STATE
    - If Pre_state is KEY_WAITED and if key equal to NOKEY, then next_state should be KEY_ENTRY or if time_button is 0, then next_state should be SHOW_TIME or next_state should be KEY_WAITED itself
  - KEY ENTRY STATE
    - If Pre_state is KEY_ENTRY and if alarm_button is 1, then next_state should be SET_ALARM_TIME or if time_out is 1, then next_state should be SET_CURRENT_TIME or if key not equal to NOKEY, then next_state should be KEY_STORED or next_state should be KEY_ENTRY itself
  - SHOW ALARM STATE
    - If Pre_state is SHOW_ALARM and if alarm_button is 0, then next_state should be SHOW_TIME or next_state should be SHOW_ALARM itself
  - SET_ALARM TIME STATE
    - If Pre_state is SET_ALARM_TIME then next_state should be SHOW_TIME
  - SET_CURRENT TIME STATE
    - If Pre_state is SET_CURRENT_TIME then next_state should be SHOW_TIME
  - Show_new_time
    - If Pre_state is KEY_STORED or KEY_ENTRY or KEY_WAITED then show_new_time should be 1
  - Show_a
    - If Pre_state is SHOW_ALARM then show_a should be 1
  - Load_new_a
    - If Pre_state is SET_ALARM_TIME then load_new_a should be 1
  - Load_new_c
    - If Pre_state is SET_CURRENT_TIME then load_new_c and reset_count should be 1

- Shift
  - If Pre_state is KEY_STORED then shift should be 1
- Time_out
  - If pre_state is KEY_WAITED and if key not equal to 10 for continuous 2560 clock cycles, then time_out should be 1

Working Directory : rtl
Source Code : counter.sv

Instructions :

- ✓ Understand the RTL provided (Note : There are some bugs inserted in DUT)
- ✓ Write sequences and properties to verify the following conditions
  - RESET
    - On reset, the signals current_time_ms_hr,current_time_ls_hr,current_time_ms_min,current_time_ls_min should be zero
  - TIMER_WRAP_BACK
    - If one_minute is high and current time is 23:59 then current time should be immediately zero
  - LOAD_NEW_CURRENT_TIME
    - If load_new_c is high, then new_current_time and current_time should be same
  - LS_HR_WRAP_BACK
    - If one_minute is high and current time is 9:59, then current_time_ms_hr should be incremented by 1 and ls_hr, ms_min and ls_min as zero
  - MIN_WRAP_BACK
    - If one_minute is high and if current time is 00:59 then, both current time ms_min and ls_min should be zero
  - MIN_WRAP_BACK
    - If one_minute is high and if current_time_ls_min is less than 9, then current_time_ls_min should increment by 1

Source Code : timegen.sv

Instructions :

- ✓ Understand the RTL provided (Note : There are some bugs inserted in DUT)
- ✓ Write sequences and properties to verify the following conditions
  - RESET
    - On reset, the signals count, one_second and one_minute should be zero
  - ONE_SECOND
    - If reset_count is 0 and count is 255 then one_second pulse should be high
  - ONE_MINUTE
    - If reset_count is 0 and count is 15359 then one_minute pulse should be high
  - MINUTE_COUNT
    - If reset_count is 0 and if count is not equal to 15359 then count should increment by 1
  - FAST_WATCH
    - If fastwatch is high, one_minute should be equal to one_second

Source Code : alarm_clock_top.sv

Instructions :

- ✓ Understand the RTL provided (Note : There are some bugs inserted in DUT)
- ✓ Write sequences and properties to verify the following conditions
  - TOP_RESET
    - On reset ms_hour, ls_hr, ms_minute, ls_minute should be 8'h30 (ASCII representation of 0) and alarm_sound should be high

Working Directory : alarm_clock_env

Source Code : top.sv

Instructions :

- ✓ Instantiate the assertion module and connect to DUT module using bind keyword

Simulation Process :

- ✓ Go to the directory **SVA_LABS/Alarm_clock/Alarm_clock_template/sim**
- ✓ Call the target TC1 to run the first test case: **make TC1**
- ✓ Observe the coverage report
- ✓ Call the target regress_12 which will run the first two test cases & merge the coverage report: **make regress_12**
- ✓ Observe the coverage report
- ✓ Finally, run the regression with all the three testcases by calling the appropriate target: **make regress_123.**
- ✓ Observe the output and also cross-check with the solution source code.
- ✓ Run the simulation in GUI mode and observe the waveforms along with assertions

Learning outcomes : How to write simple assertions to verify the Alarm clock
How to connect assertion module with DUT using **BIND** keyword