SystemVerilog

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. 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: Data Types

Objective : *Understand the usage of different types of arrays & array methods in Sytemverilog*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab01

Working Directory : tb
Source Code : test_array.sv
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.

- ✓ Declare dynamic array ***data_da***, of int data type
- ✓ Declare queues ***data_q*** & ***addr_q*** of int data type
- ✓ Declare associative array ***data_mem*** of int data type and indexed with bit[7:0]
- ✓ Declare int variable ***result***
- ✓ Declare a variable ***idx*** *as* bit [7:0]
- ✓ Within initial begin
  - Allocate 10 memory locations for dynamic array & initialize all the locations with some random values less than 20
  - Display the array
  - Call the array reduction method sum which returns the sum of all elements of the array and collect the return value to the variable ***result***
  - Display the sum of elements, ***result***
  - Similarly, explore other array reduction methods product, or, and & xor
  - Call the array reduction method sum with "with" clause which returns the total number of elements satisfying the condition within the "with" clause.
  - Display the value of the ***result***
  - Similarly, explore other array reduction methods with "with" clause
  - Call all the sorting methods like reverse, sort, rsort & shuffle & display the array after the execution of each method to understand the behavior of the array methods
  - Call array locator methods like min, max, unique, find_* with, find_*_index with using dynamic array & display the contents of the ***data_q*** after the execution of each method to understand the behavior of the array methods
  - Generate some 10 random address less than 100 within a repeat loop push the address into the ***addr_q***
  - Display the ***addr_q***
  - Within for loop, update the associate array with random data less than 200 based on the address stored in ***addr_q***
    - Hint: To get the address use pop method
  - Display the contents of the associate array using the foreach loop
  - Display the first index of the array by using associative array method first
  - Display the first element of the array
  - Display the last index of the array by using associative array method last
  - Display the last element of the array

Simulation Process :

- ✓ Go to the directory: **cd SV_LABS/Lab01/sim**
- ✓ Call the target run_test to run the simulation: **make run_test**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :

How to declare and use dynamic array, queue & associative array
Usage of array reduction methods, array sorting methods & array locator methods.

Lab - 2: Interface

Objective : *Understand the usage of Systemverilog interface construct*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab02

Working Directory : env
Source Code : ram_if.sv
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.

- ✓ Declare the interface signals:
  - ***data_in, data_out***(logic type , size 64 bits)
  - ***read, write*** (logic type, size 1bit)
  - ***rd_address, wr_address*** (logic type, size 12 bits)
- ✓ Add clocking block wr_drv_cb for write driver which drives the values at the positive edge of the clock
  - Define the input skew as #1 & output skew as #1
  - Define the direction of the signals as:
    - output: ***data_in, wr_address, write***
- ✓ Add clocking block rd_drv_cb for read driver which drives the values at the positive edge of the clock
  - Define the input skew as #1 & output skew as #1
  - Define the direction of the signals as:
    - output: ***rd_address, read***
- ✓ Add monitor clocking block wr_mon_cb for write monitor which monitors at the positive edge of the clock
  - Define the input skew as #1 & output skew as #1
  - Define the direction of the signals as:
    - input: ***data_in, wr_address, write***
- ✓ Add monitor clocking block rd_mon_cb for read monitor which monitors at the positive edge of the clock
  - Define the input skew as #1 & output skew as #1
  - Define the direction of the signals as:
    - input: ***rd_address, read, data_out***
- ✓ Add write driver modport WR_DRV_MP with driver clocking block(wr_drv_cb) as the input argument
- ✓ Add read driver modport RD_DRV_MP with driver clocking block(rd_drv_cb) as the input argument
- ✓ Add write monitor modport WR_MON_MP with monitor clocking block (wr_mon_cb) as the input argument

- ✓ Add read monitor modport RD_MON_MP with monitor clocking block (rd_mon_cb) as the input argument

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab02/sim**
- ✓ Call the target run_test to run the simulation in batch mode: **make run_test**
- ✓ Call the target view_wave to view the waveform: **make view_wave**
- ✓ In the waveform observe the following
  - Clocking block outputs will be delayed by #1 - time unit by the time it reaches DUT
  - Testbench inputs will be delayed by one cycle
- ✓ Cross-check with the solution source code.

Learning outcomes :

How to define interface block, clocking blocks & modports
Understand the input and output skew

Lab - 3: OOP Basics

Objective : *Understand the usage of Class and Basics of OOP such as handle assignment*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab03

Working Directory : tb
Source Code : test_class_defn.sv
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.

- ✓ Declare class account_c
- ✓ Within the class account_c
  - Declare variable ***balance*** as int type
  - Write a function summary that returns ***balance*** from it
  - Write a task deposit with pay (type int) as the input argument and add the pay with the ***balance***
- ✓ Declare a handle for class account_c
- ✓ Within initial block
  - Create an instance of the class account_c
  - Call method deposit and pass the pay value as the input argument and call method summary to display ***balance***

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab03/sim**
- ✓ Call the target run_class to run the simulation: **make run_class**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes:

How to define a class

How to access class properties & methods.

Working Directory : tb

Source Code : test_obj_assignment.sv

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.

- ✓ Declare class packet
- ✓ Within the class packet, declare the below class properties
  - ***data*** (bit type, size 4 bits)
  - ***addr*** (bit type, size 16 bits)
  - ***mem*** (bit type, size 16 bits)
- ✓ Declare two handles for the packet class 'pkt_h1' and 'pkt_h2'
- ✓ Within initial block
  - Construct pkt_h1 object
  - Assign random values to the ***addr***, ***data***, and ***mem*** of pkt_h1 object
  - Display the object pkt_h1
  - Assign pkt_h1 to pkt_h2
  - Display the objects pkt_h1 & pkt_h2
  - Make changes to ***address*** and ***data*** using handle pkt_h2
  - Display the object pkt_h1 & pkt_h2

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab03/sim**
- ✓ Call the target run_obj_ass to run the simulation: **make run_obj_ass**
- ✓ Observe that pkth1 and pkth2 will display the same contents because, both the handles point to the same object
- ✓ Cross-check with the solution source code.

Learning outcomes :

Handle assignment does not create a new object

Both handles point the same object

Lab - 4: Advanced OOP

Objective : *Understand deep copy, inheritance and polymorphism*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab04

Working Directory : tb
Source Code : test_deep_copy.sv
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.

- ✓ In class parity_calc_c
  - Declare ***parity*** (bit type, size 8 bits), initialize it with some random value
  - Write copy method that returns parity_calc_c class type
    - Create the copy instance
    - Copy all the current properties into copy object
- ✓ In class packet_c
  - Declare ***header*** (bit type, size 8 bits), initialize it with some random value
  - Declare ***data*** (bit type, size 8 bits), initialize it with some random value
  - Declare and create an instance of parity_calc_c
  - Define copy method that returns packet_c class type
    - Create the copy instance
    - Copy all the current class properties into copy object
- ✓ Declare 2 handles pkt_h1 & pkt_h2 for packet_c class
- ✓ Within initial
  - Create pkt_h1 object
  - Use the shallow copy method to copy pkt_h1 to pkt_h2
  - Display the properties of parent class and subclass properties of pkt_h1 and pkt_h2
  - Assign a random value to the ***header*** of pkt_h2
  - Display the properties of parent class and sub-class properties of pkt_h1 and pkt_h2
    - observe pkt_h1.***header*** does not change
  - Change parity of pkt_h2 using subclass handle from the parent class packet_c
    - Ex: pkt_h2.par.***parity*** = 19;
  - Display the properties of parent class and sub-class properties of pkt_h1 and pkt_h2
    - Observe that change reflected in pkt_h1 as the subclass handle in pkt_h1 and pkt_h2 are pointing to the same subclass object
  - Perform deep copy by calling parent class copy method
    - Ex: pkt_h2 = pkt_h1.copy;
  - Display the properties of parent class and sub-class properties of pkt_h1 and pkt_h2
    - observe the parent and subclass properties
  - Change parity of pkt_h2
    - Ex: pkt_h2.par.***parity*** = 210;
  - Display the properties of parent class and sub-class properties of pkt_h1 and pkt_h2
    - Observe that ***parity*** does not change for pkt_h1 as they are two different subclass objects

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab04/sim**
- ✓ Call the target run_copy to run the simulation: **make run_copy**
- ✓ Observe that after doing the shallow copy, a separate copy is created for class properties but for subclass objects separate copy is not created.
- ✓ Deep copy creates a separate copy for both properties and the subclass object
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :

Difference between Deep copy & Shallow copy

Working Directory : tb

Source Code : test_inheritance.sv

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.

- ✓ Within class account_base
  - Declare property ***balance*** of int type and initialize with value 2000
  - In the function summary_base, return ***balance***
  - In the task deposit
    - Pass an input argument ***pay*** of int type
    - Add pay with previous ***balance***
- ✓ Extend class account_extd from account_base
- ✓ Declare property ***balance*** of int type and initialize with value 3000
- ✓ In function summary_extd, return ***balance*** from extended class
  - The new value returned should be the summation of base class ***balance*** and extended class ***balance***
    - Hint: use super to access the base class ***balance***
- ✓ Declare a handle for account_extd class as acc_h
- ✓ Within initial block
  - Create an object for acc_h
  - Pass the amount for the method deposit
  - Call the method summary_base to display the base class ***balance***
  - Call method summary_extd to display the ***balance***
  - Observe that the super.***balance*** returns the base class ***balance***

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab04/sim**
- ✓ Call the target run_inherit to run the simulation: **make run_inherit**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :

How to extend the class.

How to access parent class properties in child class while overriding the properties

Working Directory : tb

Source Code : test_polymorphism.sv

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.

- ✓ In class packet_c
  - In task send
    - Display message "Sending base class packet"
- ✓ Extend badpacket_c from packet_c
  - Override task send
    - Display message "Sending derived class packet"
- ✓ Within initial
  - Create instances for badpacket_c and packet_c
  - Call send tasks using the base and extended handles
  - Assign extended class handle to base class handle
  - Call send task using the base class object

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab04/sim**
- ✓ Call the target run_poly to run the simulation: **make run_poly**
- ✓ Observe that the base class send task is called instead of the extended class send task, even though the base handle is pointing to the extended class object
- ✓ Now make task send as virtual in the parent class
- ✓ Run the simulation again by calling the target run_poly
- ✓ Now observe that the child class send task overrides the parent send method as the method is declared as virtual in the base class
- ✓ Cross-check with the solution source code.

Learning outcomes :

Overriding the base class virtual methods with child class methods

Lab - 5: Randomization

Objective : *Understand the usage of Random variables and Constraints*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab05

Working Directory : tb
Source Code : test_random.sv
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.

- ✓ In class alu_trans_c
  - Declare a typedef variable OPERATION of type enum with ***LOGIC***, ***ARITH***, and ***SHIFT*** as the values
  - Add the following rand fields
    - ***alu_sel_in*** (logic type, size 4 bits)
    - ***rand_oper*** (OPERATION type)
  - Apply the following constraints
    - If ***rand_oper*** = ***LOGIC*** then ***alu_sel_in*** should be inside [0:5]
    - If ***rand_oper*** = ***ARITH***, then ***alu_sel_in*** should be inside [6:9]
    - If ***rand_oper*** = ***SHIFT***, then ***alu_sel_in*** should be inside [10:15]
    - Give weightage of 2 to ***ARITH*** operation
  - In post_randomize method, display all the randomized values
- ✓ Within initial
  - Create an instance of alu_trans_c
  - Generate 10 sets of random values for the instance alu_trans_c
  - Randomize using 'assert' or 'if ' construct

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab05/sim**
- ✓ Call the target run_test to run the simulation: **make run_test**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes:

How to randomize the class properties & define the constraints

Lab - 6: Threads, Mailbox and Semaphore

Objective : *Understand the usage of threads, mailbox, and semaphore*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab01

Working Directory : tb
Source Code : test_threads.sv
Instructions : Understand the code provided

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab06/sim**
- ✓ Call the target run_thread to run the simulation: **make run_thread**
- ✓ Observe the output
- ✓ Now replace join with join_any & join_none and run the simulation. Observe the difference in the output
- ✓ Observe the difference in output by commenting and uncommenting the disable thread
- ✓ Cross-check with the solution source code.

Learning outcomes :

The difference between the execution of threads: fork-join, fork-join_any & fork-join_none and how to disable the threads.

Working Directory : tb
Source Code : test_mailbox.sv
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.

- ✓ In class packet
  - Add the following rand fields
    - ***data*** (bit type, size 4 bits)
    - ***addr*** (bit type, size 4 bits)
  - In the display function, pass a string as an input argument
    - Display the input string message
    - Display the ***data*** and ***address***
  - In post_randomize methods call display method and pass the string argument as "randomized data"
- ✓ In class generator
  - Declare a handle of packet class
  - Declare the mailbox parameterized by type class packet
  - In constructor
    - Pass the mailbox parameterized by the packet as an argument of the constructor
    - Assign the mailbox handle argument to local mailbox handle of generator
  - In task start, within fork - join_none

- Within fork join_none
- Create 10 random packets
- Randomize each packet using assert & randomize method
- Put the generated random packets into the mailbox

✓ In class driver
- Declare a handle of packet class
- Declare a mailbox parameterized by type class packet
- In constructor
  - Pass the mailbox parameterized by the packet as an argument
  - Assign the mailbox handle argument to local mailbox handle of the driver
- In task start, within fork - join_none
  - Get the 10 generated random packets from the mailbox
  - Use the display method in the packet class to display the received data

✓ In class env
- Create the mailbox instance parameterized by packet
- Declare handles of generator and driver
- In build function
  - Create an instance of generator and driver bypassing mailbox as an input argument
- In task start
  - Call start task of generator and driver

✓ Within initial block
- Create an instance of env
- Call build and start the task of env

Simulation Process :

✓ Go to the directory **cd SV_LABS/Lab06/sim**

✓ Call the target run_mailbox to run the simulation: **make run_mailbox**

✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :

How to connect two transactors using mailbox

Working Directory : tb

Source Code : test_semaphore.sv

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.

✓ In class driver
- write task send with input argument of string type
  - Get the key using sem handle
  - Display the string which indicates the respective driver information
  - Put the key using sem handle
  - Display the string which indicates the respective driver information

✓ Declare an array of two drivers

✓ Declare a handle for semaphore

✓ Within initial block

- Create instances of drivers

- Create the instance of semaphore handle and initialize it with 1 key
- Call send task of both drivers 5 times within fork-join, pass any meaning full string message to indicate the driver information.

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab06/sim**
- ✓ Call the target run_sem to run the simulation: **make run_sem**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :

Usage of semaphore

Lab - 7: Transaction

Objective : *Understand how to define the transaction*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab07

Working Directory : env

Source Code : ram_trans.sv

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.

- ✓ In class ram_trans
  - Declare the following rand fields
    - ***data*** (bit/logic type , size 64 bits)
    - ***rd_address, wr_address*** (bit/logic type , size 12 bits)
    - ***read, write*** (bit/logic type , size 1 bit)
  - Declare a variable ***data_out*** (logic type , size 64 bits)
  - Declare a static variable ***trans_id*** (int type), to keep the count of transactions generated
  - Declare 3 static variables ***no_of_read_trans***, ***no_of_write_trans***, ***no_of_RW_trans*** (int type)
  - Add the following constraints
    - ***wr_address*** != ***rd_address***
    - ***read,write*** != 2'b00
    - ***data*** between 1 and 4294
  - In virtual function display
    - Display the string
    - Display all the properties of the transaction class
  - In post_randomize method
    - Increment ***trans_id***
    - If it is only read transaction, increment ***no_of_read_trans***
    - If it is only write transaction, increment ***no_of_write_trans***
    - If it is read-write transaction, increment ***no_of_RW_trans***
    - Call the display method and pass a string

Source Code : ram_pkg.sv

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.

- ✓ In class ram_pkg
  - Declare a variable ***number_of_transactions*** of int data type and initialize it to 1.
  - Include transaction class
  - Include all the transactors
  - Include environment
  - Include test

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab07/sim**
- ✓ Call the target run_test to run the simulation: **make run_test**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :

How to define the transaction class

Lab - 8: Transactors

Objective : *Understand how to create the transactors like generator, driver, monitor and reference model*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab08

Working Directory : env

Source Code : ram_gen.sv

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.

- ✓ In class ram_gen
  - Declare gen_trans handle of class type ram_trans which has to be randomized
  - Declare a handle data2send of class type ram_trans which has to be put into the mailboxes
  - Declare 2 mailboxes gen2rd, gen2wr parameterized by ram_trans
  - In constructor new function
    - Add mailbox argument parameterized by transaction class and make the assignment to the mailbox
    - Create the object for the handle to be randomized
  - In virtual task start
    - Inside fork join_none
    - Generate random transactions equal to ***number_of_transactions***(defined in package)
    - Randomize using transaction handle using 'if ' or 'assert'
    - If randomization fails, display the message "DATA NOT RANDOMIZED" and stop the simulation.
    - Shallow copy gen_trans to data2send
    - Put the handle into both the mailboxes

Source Code : ram_read_drv.sv

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.

- ✓ In class ram_read_drv
  - Instantiate virtual interface instance rd_drv_if of type ram_if with RD_DRV_MP modport
  - Declare a handle for ram_trans as data2duv
  - Declare a mailbox gen2rd parameterized by ram_trans
  - In the constructor, pass the following as the input arguments
    - Virtual interface, mailbox handle gen2rd parameterized with ram_trans
    - Make connections
    - For example this.gen2rd = gen2rd
  - Understand the virtual task drive provided
  - In virtual task start
    - Within fork join_none
    - Within forever, inside begin end get the data from mailbox gen2rd and call drive task

Source Code : ram_write_drv.sv

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.

- ✓ In class ram_write_drv
  - Instantiate virtual interface instance wr_drv_if of type ram_if with WR_DRV_MP modport
  - Declare a handle for ram_trans as data2duv
  - Declare a mailbox gen2wr parameterized by ram_trans
  - In constructor pass the following as the input arguments
    - Virtual interface, mailbox handle parameterized gen2wr ram_trans
    - Make connections
    - For example this.gen2wr=gen2wr
  - Understand the virtual task drive provided
  - In virtual task start
    - Within fork join_none
    - Within forever, inside begin-end get the data from mailbox gen2wr and call drive task

Source Code : ram_read_monitor.sv

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.

- ✓ In class ram_read_monitor
  - Instantiate virtual interface instance rd_mon_if of type ram_if with RD_MON_MP modport
  - Declare 3 handles rddata, data2rm, data2sb of class type ram_trans
  - Declare 2 mailboxes mon2rm and mon2sb parameterized by type ram_trans
  - In constructor pass the following properties as the input arguments
    - Virtual interface and the two mailboxes as arguments
    - Make the connections and allocate memory for 'rddata'
  - Understand the virtual task monitor provided
  - In the virtual task start
    - Within fork-join_none
    - In forever loop call the monitor task
    - Shallow copy rddata to data2rm and data2sb
    - Put the transaction item into two mailboxes mon2rm and mon2sb

Source Code : ram_write_monitor.sv

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.

- ✓ In class ram_write_monitor
  - Instantiate virtual interface instance wr_mon_if of type ram_if with WR_MON_MP modport
  - Declare 2 handles wrdata and data2rm of class type ram_trans
  - Declare a mailbox mon2rm parameterized by type ram_trans
  - In constructor pass the following properties as the input arguments
    - Virtual interface and a mailbox as an argument
    - Make the connections and allocate memory for 'wrdata'
  - Understand the virtual task monitor provided
  - In the virtual task start
    - within fork-join_none

- In forever loop call the monitor task
- Shallow copy wrdata to data2rm
- Put the transaction item into the mailbox mon2rm

Source Code : ram_pkg.sv

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.

- ✓ In class ram_pkg
  - Declare a variable ***number_of_transactions*** of int data type and initialize it to 1.
  - Include transaction class
  - Include all the transactors
  - Include environment
  - Include test

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab08/sim**
- ✓ Call the target run_test to run the simulation: **make run_test**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :

How to define the transactors

Lab - 9: Scoreboard and Reference Model

Objective : *Understand how to define scoreboard and reference model*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab09

Working Directory : env
Source Code : ram_sb.sv
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.

- ✓ In class ram_sb
  - Declare an event DONE
  - Declare 3 variables of int datatype for counting
    - number of read data received from the reference model(***rm_data_count***)
    - number of read data received from the monitor(***mon_data_count***)
    - number of read data verified(***data_verified***)
  - Declare ram_trans handles as rm_data, rcvd_data and cov_data
  - Declare 2 mailboxes as rm2sb, rdmon2sb parameterized by ram_trans
  - Write the functional coverage model
    - Define a covergroup as mem_coverage
    - Define coverpoint and bins for read, data_out and rd_address
    - Define cross for read, rd_address
  - In constructor pass mailboxes as an argument, connect the mailboxes and create an instance for the covergroup
  - In the virtual task start
    - Within fork, join_none begin end
    - Get the data from mailbox rm2sb and increment ***rm_data_count***
    - Get the data from mailbox rdmon2sb and increment ***mon_data_count***
    - Call check task and pass rcvd_data handle as the input argument
  - In virtual task check
    - After the comparison logic which is provided shallow copy the rm_data to cov_data
    - Call the sample function on the covergroup
    - Increment ***data_verified***
    - Trigger the event if the verified data count is equal to the number of read and read- write transactions
  - In virtual function report
    - Display ***rm_data_count***, ***mon_data_count***, ***data_verified***

Source Code : ram_model.sv
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.

- ✓ In class ram_model
  - Declare 2 handles wrmon_data and rdmon_data for ram_trans to store the data from the read and write monitor
  - Declare an associative array of 64 bits(logic type) and indexed int type
  - Declare 3 mailboxes wr2rm, rd2rm and rm2sb parameterized by ram_trans

- In constructor pass mailboxes as the arguments and connect the mailboxes
- Understand and include the virtual tasks dual_mem_fun_write and dual_mem_fun_read
- These tasks perform the following
  - During the ***write***, the associative array is updated with ***data*** based on the write address
  - During ***read***, the expected data is generated from the associative array based on the read address
- In virtual task start
  - Within fork join_none
  - Within fork join
  - Within forever
    - Get the data from wr2rm mailbox to wrmon_data
    - Call dual_mem_fun_write task and pass the data wrmon_data as an argument
  - Within forever
    - Get the data from rd2rm mailbox to rdmon_data
    - Call dual_mem_fun_read task and pass the data rdmon_data as an argument
    - Put rdmon_data into rm2sb mailbox

Source Code : ram_pkg.sv

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.

- ✓ In class ram_pkg
  - Declare a variable ***number_of_transactions*** of int data type and initialize it to 1.
  - Include transaction class
  - Include all the transactors
  - Include environment
  - Include test

Simulation Process :

- ✓ Go to the directory **cd SV_LABS/Lab09/sim**
- ✓ Call the target run_test to run the simulation: **make run_test**
- ✓ Observe the functional coverage report

Learning outcomes :

How to define the scoreboard and reference model class

Lab - 10: Environment and Testcases

Objective : *Understand how to create the verification environment*

Main Working Directory: $HOME/VLSI_RN/SV_LABS/Lab10

Working Directory : env
Source Code : ram_env.sv
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.

- ✓ In class ram_env
  - Instantiate virtual interface with Write Driver modport, Read Driver modport, Write monitor modport, Read monitor modport
  - Declare 6 mailboxes parameterized by ram_trans and construct it
  - Create handle for ram_gen, ram_read_drv, ram_write_drv, ram_read_mon, ram_write_mon, ram_model, ram_sb
  - In constructor pass the driver and monitor interfaces as the argument and connect with the virtual interfaces of ram _env
  - In the virtual task build, create instances for generator, Read Driver, Write Driver, Write monitor, Read monitor, Reference model, Scoreboard
  - Understand and include the virtual task reset_dut
  - In the virtual task start, call all the start methods of the generator, Read Driver, Write Driver, Read monitor, Write Monitor, reference model, scoreboard
  - In the virtual task run, call reset_dut, start, stop methods and report function from the scoreboard

Source Code : ram_pkg.sv
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.

- ✓ In class ram_pkg
  - Declare a variable ***number_of_transactions*** of int data type and initialize it to 1.
  - Include transaction class
  - Include all the transactors
  - Include environment
  - Include test

Working Directory : test
Source Code : ram_test.sv
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.

- ✓ In class ram_base_test
  - Instantiate virtual interface with Write Driver modport, Read Driver modport, Write monitor modport, Read monitor modport
  - Declare a handle for ram_env as env_h
  - In constructor pass the driver and monitor interface as the arguments

- Create the object for env and pass the arguments for the virtual interfaces in new() function
- Understand the virtual task build which builds the TB environment
- Understand the virtual task run which runs the simulation for base test

✓ In class ram_test_extnd1

- Declare a handle for extended ram_trans
- In constructor pass the driver and monitor interface as the arguments
  - Call the parent constructor using super.new and pass the arguments of the virtual interfaces
- Understand the virtual task build which builds the TB environment
- Understand the virtual task run which runs the simulation for extended tests
  - Create the handle for extended ram_trans
  - Assign the ram_trans extended handle to the base class handle of ram_trans

Source Code : top.sv

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.

✓ In module top

- Import the package
- Declare the required handles for all the testcases
- Instantiate the interface
- Instantiate DUT and portmap the signals with interface signals
- Generate the clock
- Create the object for base test and pass the interface instances
  - Call the task build and task run with base test handle
- Create the object for extended test and pass the interface instances
  - Call the task build and task run with extended test handle

Simulation Process :

✓ Go to the directory **cd SV_LABS/Lab10/sim**

✓ Call the target TC1 to run the first test case: **make TC1**

✓ Observe the coverage report and based on the bins that are not covered define additional constraints in the provided ram_trans_extnd class

✓ Call the target TC2 to run the second test case: **make TC2**

✓ Or call the target regress_12 which will run the first two test cases & merge the coverage report: **make regress_12**

✓ Based on the coverage report, if required add additional test cases by adding constraints to the extending ram_trans class & by extending ram_base_test class

✓ Finally, run the regression with all the three testcases by calling the appropriate target: **make regress_123.**

✓ Based on the coverage report, if required add additional test cases by extending ram_trans class & by adding constraints.

✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :

How to build a TB environment and how to write different testcase for the coverage closure.