MAVEN SILICON

Centre of Excellence in VLSI

UVM
Lab Manual

www.maven-silicon.com

Maven Silicon Confidential

Table of Contents

Lab Instructions....4
Lab - 1 : Stimulus Modelling 6
Lab - 2 : Factory Overriding 8
Lab - 3 : Phases 9
Lab - 4 : Creating Agent....11
Lab - 5 : UVM Sequences....15
Lab - 6 : Virtual Interface....18
Lab - 7 : Agent Integration....21
Lab - 8 : Scoreboard....27
Lab - 9 : SOC Implementation....30
Lab - 10 : Coverage and Regression 33

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 & interface.
*_agt_top/	- contains driver, monitor, sequencer, agent, agt_top, agent configuration and transaction class
tb/ or env/	- contains verification environment, scoreboard, virtual sequence, virtual sequencer, environment configuration & top module
test/	- contains testcases & package
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 : Stimulus Modelling

Objective : *Creating Sequence item and exploring UVM built-in methods copy, compare, print, clone, etc*

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

Working Directory : wr_agt_top
Source Code : write_xtn.sv
Instructions : The following instructions have been included in the source code as comments. Refer the comments in the source code and edit the source code.

- ✓ Extend write_xtn from uvm_sequence_item
- ✓ Add UVM Factory Registration Macro
- ✓ Add the following rand fields
    - ***data***(`RAM_WIDTH-1:0), ***address***(`ADDR_SIZE-1:0) and ***write*** (type bit)
- ✓ Add the rand control knobs declared in tb_defs.sv
    - ***xtn_type*** (enumerated type addr_t) - for controlling the type of transaction
    - ***xtn_delay*** (integer type) - for inserting delay between transactions
- ✓ Add the following constraints:
    - Data between 20 through 90
    - Address between 0 through 200
    - Distribute weights for xtn_type : BAD_XTN=2 and GOOD_XTN=30
- ✓ Add following standard UVM Methods
    - Add code for new() constructor
    - Add code for do_copy() to copy address, data, write, xtn_type and xtn_delay as per the instructions given in lab exercise
    - Add code for do_compare() to compare address, data, write, xtn_type and xtn_delay as per the instructions given in lab exercise
    - understand the do_print method implemented in the lab exercise
- ✓ In post_andomize method assign address to 6000 if xtn_type is BAD_XTN

Working Directory : wr_agt_top
Source Code : write_xtn_macros.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.

- ✓ Extend write_xtn from uvm_sequence_item
- ✓ Add the following rand fields
    - ***data***(`RAM_WIDTH-1:0), ***address***(`ADDR_SIZE-1:0) and ***write*** (type bit)
- ✓ Add the rand control knobs declared in tb_defs.sv
    - ***xtn_type*** (enumerated type addr_t) - for controlling the type of transaction
    - ***xtn_delay*** (integer type) - for inserting delay between transactions
- ✓ Add factory registration and use macros for all the fields
- ✓ Add the following constraints:
    - Data between 20 through 90
    - Address between 0 through 200
    - Distribute weights for xtn_type : BAD_XTN=2 and GOOD_XTN=30
- ✓ Add code for new() constructor
- ✓ In post_andomize method assign address to 6000 if xtn_type is BAD_XTN

Working Directory **:** packages
Source Code **:** ram_pkg.sv

- ✓ Import uvm_pkg
- ✓ Include the following files
  - uvm_macros.svh, tb_defs.sv, write_xtn.sv

Working Directory **:** tb
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.

- ✓ Import ram_pkg
- ✓ Declare two handles of write_xtn class ***wr_copy_xtnh*** and ***wr_clone_xtnh***
- ✓ Declare a dynamic array of handles for write_xtn as ***wr_xtnh***
- ✓ Declare a variable ***no_trans*** as int data type and initialize it with value 10.
- ✓ Within initial block
  - Allocate the size of the above-declared array equal to 10
  - Within for loop, Generate ten random transactions
    - Create 10 write_xtn class objects with different strings using $sformatf
    - randomize & print the 10 transaction class objects
  - Copy the 5th transaction item into the 3rd transaction item using the copy method
  - Copy the 3rd transaction item into another item(wr_copy_xtnh) using the copy method
    - Note : Do create an instance for wr_copy_xtnh
  - Print the object wr_copy_xtnh in a tree format
  - Call Compare method on the 5th and 3rd transaction items
  - Using clone() method copy the 8th item to another item(wr_clone_xtnh)
    - Note : Do not create an instance for wr_clone_xtnh
  - Print the object wr_clone_xtnh in a line format

Simulation Process :

- ✓ Go to the directory: **cd UVM_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.
- ✓ Optionally add write_xtn_macros.sv in ram_pkg by removing write_xtn.sv and run the simulationby calling the target run_test: **make run_test**

Learning outcomes:
How to define the transaction class
How to use UVM built-in methods like do_copy, do_print and do_compare

Lab - 2 : Factory Overriding

Objective : *Understand factory registration and construction of objects using factory override methods*

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

Working Directory : wr_agt_top
Source Code : short_xtn.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.

- ✓ Extend short_xtn from write_xtn
- ✓ Add UVM Factory Registration Macro
- ✓ Override Constraint for address such that it generates address which is always equal to 10
- ✓ Add code for constructor new()

Working Directory : packages
Source Code : ram_pkg.sv

- ✓ Import uvm_pkg
- ✓ Include the following files
  - uvm_macros.svh, tb_defs.sv, write_xtn.sv, short_xtn.sv

Working Directory : tb
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.

- ✓ Import the ram_pkg
- ✓ Declare handle for write_xtn as ***wr_xtnh***
- ✓ Add build method
  - Create an instance of wr_xtnh using factory create()
  - Randomize and print the transactions
- ✓ Within initial begin
  - Call build function 5 times (Without Overriding)
  - Call factory overriding method
    - Hint : Use factory.set_type_override_by_type Override before calling build function
  - Call build function 5 times

Simulation Process :

- ✓ Go to the directory: **cd UVM_LABS/Lab02/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 do factory overriding

Lab - 3 : Phases

Objective : *Explore different UVM phases*

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

Working Directory : wr_agt_top
Source Code : ram_wr_driver.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.

- ✓ Extend ram_wr_driver from uvm_driver
- ✓ Add factory registration macro
- ✓ Add code for constructor new
- ✓ Add all the UVM phases:
  - Call super.*_phase() in every phase method ,* indicates build,connect,etc
  - Use `uvm_info in each of these phases and display the message
    - Hint : `uvm_info("RAM_DRIVER", "This is Build Phase ", UVM_LOW)
  - In run phase raise and drop objections
  - Within raising and dropping the objections add a delay of 10 in the run phase before printing

Working Directory : wr_agt_top
Source Code : ram_wr_agent.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.

- ✓ Extend ram_wr_agent from uvm_agent
- ✓ Declare the handle of ram_wr_driver
- ✓ Add factory registration macro
- ✓ Add code for constructor new
- ✓ Add all the UVM phases:
  - Call super.*_phase() in every phase method ,* indicates build,connect,etc
  - Use `uvm_info in each of these phases and display the message
    - Hint : `uvm_info("RAM_AGENT", "This is Build Phase ", UVM_LOW)
  - In the build phase create the instance of the driver
  - In run phase raise and drop objections
  - Within raising and dropping the objections add a delay of 100 in the run phase before printing

Working Directory : tb
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.

- ✓ Extend ram_env from uvm_env
- ✓ Declare the handle of ram_wr_agent
- ✓ Add factory registration macro
- ✓ Add code for constructor new

- ✓ Add all the UVM phases:
  - Call super.*_phase() in every phase method ,* indicates build,connect,etc
  - Use `uvm_info in each of these phases and display the message
    - ▪ Hint : `uvm_info(“RAM_ENV”, “This is Build Phase ”, UVM_LOW)
  - In the build phase create the instance of ram_wr_agent
  - In run phase raise and drop objections
  - Within raising and dropping the objections add a delay of 100 in the run phase before printing

Working Directory : test
Source Code : ram_wr_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.

- ✓ Extend ram_wr_test from uvm_test
- ✓ Declare the handle of ram_env
- ✓ Add factory registration macro
- ✓ Add code for constructor new
- ✓ Add all the UVM phases:
  - Call super.*_phase() in every phase method ,* indicates build,connect,etc
  - Use `uvm_info in each of these phases and display the message
    - ▪ Hint : `uvm_info(“RAM_TEST”, “This is Build Phase”, UVM_LOW)
  - In the build phase create the instance of ram_env
  - In run phase raise and drop objections
  - Within raising and dropping the objections add a delay of 100 in the run phase before printing

Working Directory : packages
Source Code : ram_pkg.sv

- ✓ Import uvm_pkg
- ✓ Include the following files
  - uvm_macros.svh, tb_defs.sv, ram_wr_driver, ram_wr_agent, ram_env,ram_wr_test

Working Directory : tb
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.

- ✓ Import ram_pkg.sv
- ✓ Import the UVM package
- ✓ Include the uvm_macros.svh
- ✓ Within initial block call run_test("ram_wr_test")

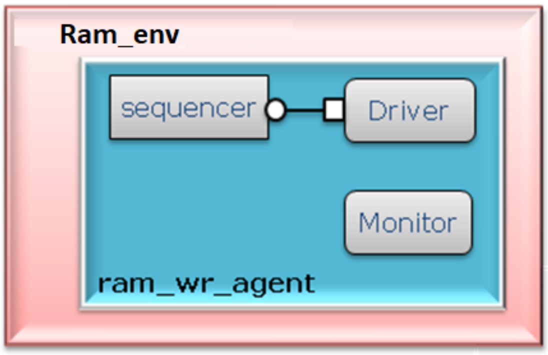
Simulation Process :

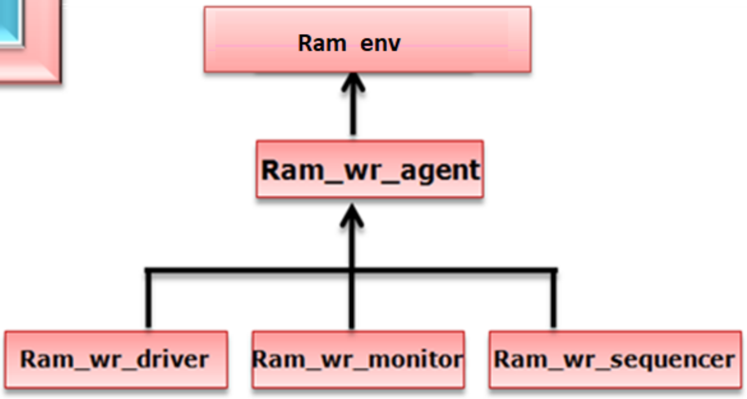
- ✓ Go to the directory: **cd UVM_LABS/Lab03/sim**
- ✓ Call the target run_test to run the simulation: **make run_test**
- ✓ Observe the output and understand the order of execution of phases
- ✓ Cross-check with solution source code

Learning outcomes :
Execution order of UVM phases

Lab - 4 : Creating Agent

Objective : *Build the Class based environment with all the components such as driver, monitor, agent, etc.*





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

Working Directory : wr_agent_top
Source Code : ram_wr_agent_config.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.

- ✓ Extend ram_wr_agent_config from uvm_object
- ✓ Add UVM Factory Registration Macro
- ✓ Declare parameter ***is_active*** of type uvm_active_passive_enum and assign it to UVM_ACTIVE
- ✓ Add the constructor new method

Working Directory : wr_agent_top
Source Code : ram_wr_sequencer.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.

- ✓ Extend ram_wr_sequencer from uvm_sequencer parameterized by write_xtn
- ✓ Add UVM Factory Registration Macro
- ✓ Add the constructor new method

Working Directory : wr_agent_top
Source Code : ram_wr_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.

- ✓ Extend ram_wr_monitor from uvm_monitor
- ✓ Add UVM factory Registration
- ✓ Add the constructor new method
- ✓ Add the run() phase method and print info message "This is write monitor run_phase"

Working Directory : wr_agent_top
Source Code : ram_wr_driver.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.

- ✓ Extend ram_wr_driver from uvm driver parameterized by write_xtn
- ✓ Add factory Registration
- ✓ Add constructor new() function
- ✓ Add run() phase method
  - In forever loop
    - Get the sequence item using seq_item_port
    - Call send_to_dut task
    - send the item_done to the sequencer using seq_item_port
- ✓ Add task send_to_dut(write_xtn handle as an input argument)
  - Print the transaction

Working Directory : wr_agent_top
Source Code : ram_wr_agent.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.

- ✓ Extend ram_wr_agent from uvm_agent
- ✓ Add UVM factory Registration
- ✓ Declare handle for configuration class ram_wr_agent_config
- ✓ Declare handles of ram_wr_monitor, ram_wr_sequencer and ram_wr_driver with handle names as ***monh, seqrh, drvh*** respectively
- ✓ Add constructor new method
- ✓ Add build() phase method
  - Get the config object using uvm_config_db
  - Create ram_wr_monitor instance
  - If config parameter is_active=UVM_ACTIVE
  - Create instance of driver and sequencer
- ✓ Add connect() phase method
  - If config parameter is_active=UVM_ACTIVE, connect driver(TLM seq_item_port) and sequencer(TLM seq_item_export)

Working Directory : wr_agent_top
Source Code : ram_wr_seqs.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.

- ✓ Extend ram_wbase_seq from uvm_sequence parameterized by write_xtn
- ✓ Add factory registration
- ✓ Add constructor new method
- ✓ Extend ram_rand_wr_xtns from ram_wbase_seq;
- ✓ Add Factory registration
- ✓ Add constructor new method

- ✓ Add task body method
  - Generate 10 transactions of type write_xtn
    - create req instance
    - start_item(req)
    - assert for randomization
    - finish_item(req)

Working Directory : tb
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.

- ✓ Extend ram_env from uvm_env
- ✓ Declare the handle of ram_wr_agent with handle name as ***wr_agnth***
- ✓ Add factory registration macro
- ✓ Add code for constructor new
- ✓ Add build phase
  - create the instance of ram_wr_agent

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.

- ✓ Extend ram_base_test from uvm_test
- ✓ Add factory Registration
- ✓ Declare the ram_env and ram_wr_agent_config handles as ***ram_envh*** and ***m_ram_cfg*** respectively
- ✓ Define Constructor new() function
- ✓ Add build() phase method
  - Create the instance of config_object
  - Set is_active in config object to UVM_ACTIVE
  - Set the config object into UVM config DB
  - Create the instance for env
- ✓ Extend ram_random_test from ram_base_test;
- ✓ Declare the handle for ram_rand_wr_xtns sequence
- ✓ Add constructor new method
- ✓ Add build() phase method
  - In build phase call build phase of ram_base_test
- ✓ Add end_of_elobration() phase method
  - print topology
- ✓ Add run() phase method

  - Raise objection
  - Create instance for sequence
  - Start the sequence on write agent sequencer
  - Drop objection

Working Directory : tb

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.

- ✓ Import ram_test_pkg.sv
- ✓ Import the UVM package
- ✓ Include the uvm_macros.svh
- ✓ Within initial block call run_test("ram_random_test")

Simulation Process :

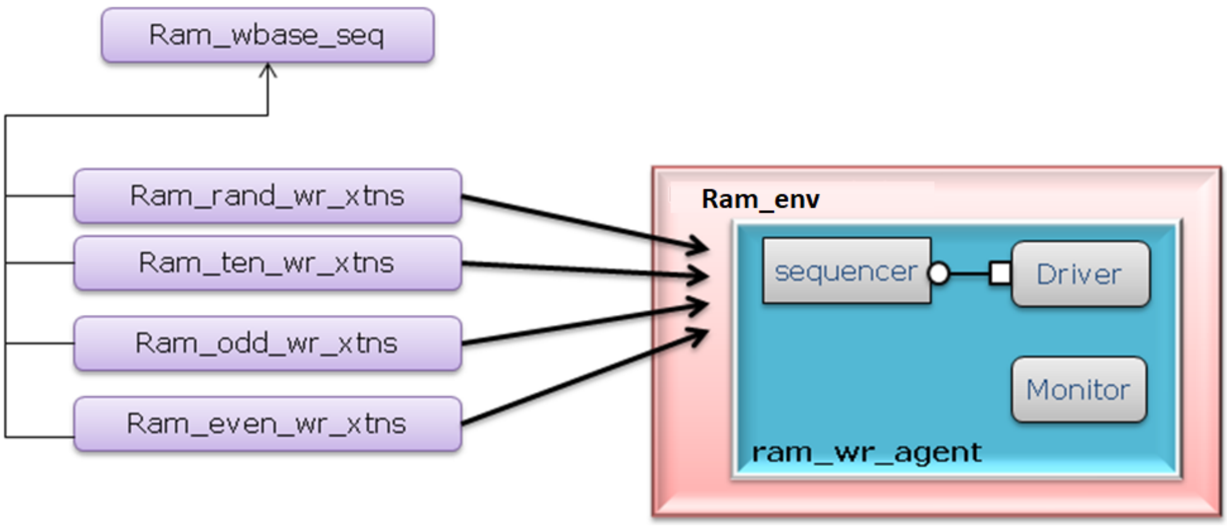
- ✓ Go to the directory: **cd UVM_LABS/Lab04/sim**
- ✓ Call the target run_test to run the simulation: **make run_test**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :

How to create a UVM agent

Lab - 5 : UVM Sequences

Objective : *Build sequences to generate stimulus for different scenarios*



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

Working Directory : wr_agt_top
Source Code : ram_wr_seqs.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.

- ✓ Extend ram_wbase_seq from uvm_sequence parametrized with write_xtn
- ✓ Add factory registration
- ✓ Add constructor new method
- ✓ Extend ram_rand_wr_xtns from ram_wbase_seq
- ✓ Add factory registration
- ✓ Add constructor new method
- ✓ Add task body
  - Generate 10 transactions of type write_xtn
  - Create req instance
  - start_item(req)
  - assert for randomization
  - finish_item(req)

- ✓ Extend ram_single_addr_wr_xtns from ram_wbase_seq
- ✓ Add factory registration
- ✓ Add constructor new method
- ✓ Add task body
  - Generate 10 sequence items with address always equal to 55

- Hint use create req, start item, assert for randomization with inline constraint (with) finish item inside repeat's begin end block

✓ Extend ram_ten_wr_xtns from ram_wbase_seq
✓ Add factory registration
✓ Add constructor new method
✓ Add task body
- Write the random data on memory address locations consecutively from 0 to 9
- Hint use create req, start item, assert for randomization with inline constraint (with) finish item inside for loop begin end block

✓ Extend ram_odd_wr_xtns from ram_wbase_seq
✓ Add factory registration
✓ Add constructor new method
✓ Add task body
- write the 10 random data in odd memory address locations
- Hint use create req, start item, assert for randomization with inline constraint (with) finish item inside repeat's begin end block

✓ Extend ram_even_wr_xtns from ram_wbase_seq
✓ Add factory registration
✓ Add constructor new method
✓ Add task body
- write the 10 random data in even memory address locations
- Hint use create req, start item, assert for randomization with inline constraint (with) finish item inside repeat's begin end block

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.

✓ Extend ram_single_addr_test from ram_base_test
✓ Declare the handle for ram_single_addr_wr_xtns sequence
✓ Add constructor new method
✓ Add build() phase method
- In build phase call build phase of ram_base_test

✓ Add run() phase method
- Raise objection
- Create instance for sequence
- Start the sequence on write agent sequencer
- Drop objection

✓ Extend ram_ten_addr_test from ram_base_test
✓ Declare the handle for ram_ten_wr_xtns sequence
✓ Add constructor new method
✓ Add build() phase method
- In build phase call build phase of ram_base_test

✓ Add run() phase method
- Raise objection
- Create instance for sequence
- Start the sequence on write agent sequencer
- Drop objection

✓ Extend ram_odd_addr_test from ram_base_test
✓ Declare the handle for ram_odd_wr_xtns sequence

- ✓ Add constructor new method
- ✓ Add build() phase method
    - In build phase call build phase of ram_base_test
- ✓ Add run() phase method
    - Raise objection
    - Create instance for sequence
    - Start the sequence on write agent sequencer
    - Drop objection

- ✓ Extend ram_even_addr_test from ram_base_test
- ✓ Declare the handle for ram_even_wr_xtns sequence
- ✓ Add constructor new method
- ✓ Add build() phase method
    - In build phase call build phase of ram_base_test
- ✓ Add run() phase method
    - Raise objection
    - Create instance for sequence
    - Start the sequence on write agent sequencer
    - Drop objection

Working Directory : tb

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.

- ✓ Import ram_test_pkg.sv
- ✓ Import the UVM package
- ✓ Include the uvm_macros.svh
- ✓ Within initial block call run_test( )

Simulation Process :

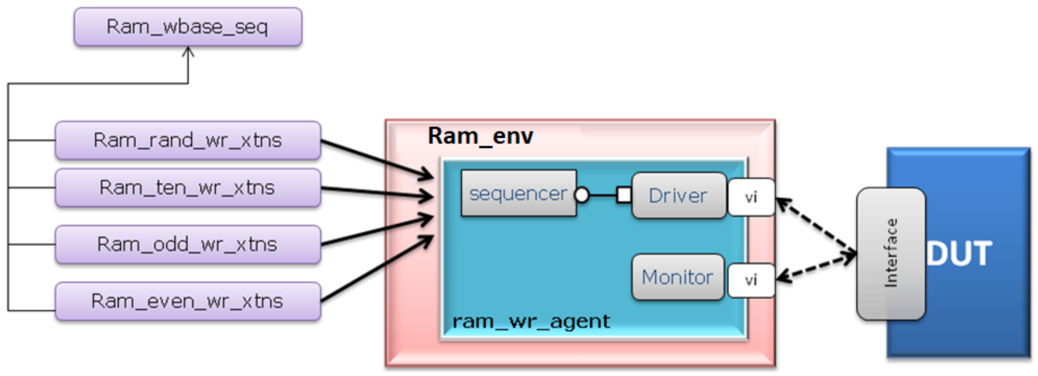
- ✓ Go to the directory: **cd UVM_LABS/Lab05/sim**
- ✓ Call the target run_test to run ram_single_addr_test: **make run_test**
- ✓ Call the target run_test1 to run ram_ten_addr_test: **make run_test1**
- ✓ Call the target run_test2 to run ram_odd_addr_test: **make run_test2**
- ✓ Call the target run_test3 to run ram_even_addr_test: **make run_test3**
- ✓ Observe the output and also cross check with the solution source code.

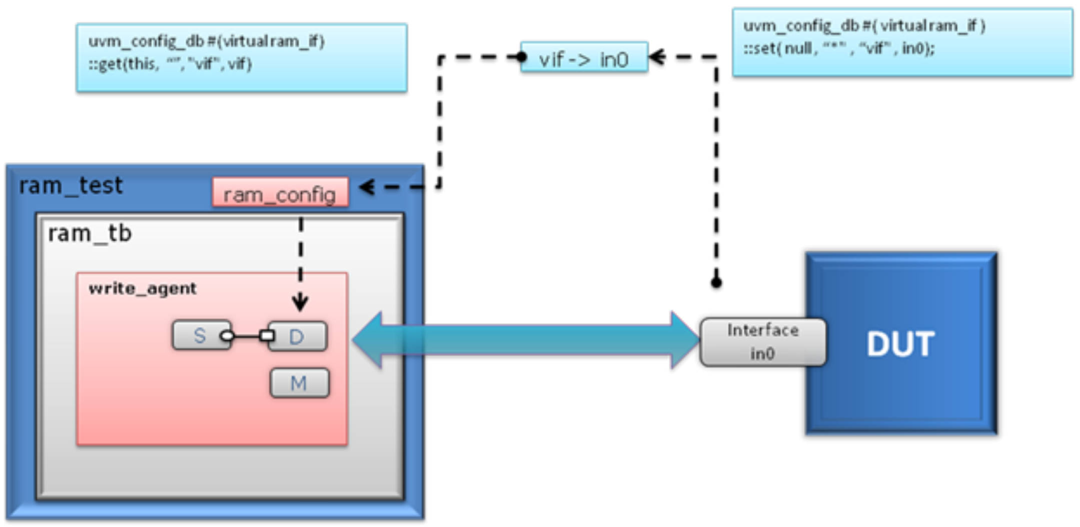
Learning outcomes :

Generating different scenarios[Testcases] by defining various random sequences[Stimulus] using a UVM sequence item[Transaction].

Lab - 6 : Virtual Interface

Objective : *Connecting virtual interface with a static interface using config_db*




uvm_config_db #(virtual ram_if)
::get(this, "", "vif", vif)
vif -> in0
uvm_config_db #(virtual ram_if)
::set(null, "*" , "vif" , in0);
ram_test
ram_config
ram_tb
write_agent
S
D
M
Interface
in0
DUT

Main Working Directory: $HOME/VLSI_RN/UVM_LABS/Lab06

Working Directory : wr_agent_top
Source Code : ram_wr_agent_config.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.

- ✓ Extend ram_wr_agent_config from uvm_object
- ✓ Add UVM Factory Registration Macro

- ✓ Declare parameter ***is_active*** of type uvm_active_passive_enum and assign it to UVM_ACTIVE
- ✓ Declare virtual interface handle for ram_if as ***vif***
- ✓ Add the constructor new method

Working Directory : wr_agent_top
Source Code : ram_wr_driver.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.

- ✓ Extend ram_wr_driver from uvm_driver parameterized by write_xtn
- ✓ Add factory Registration
- ✓ Declare virtual interface handle with WDR_MP as modport
- ✓ Declare the ram_wr_agent_config handle as ***m_cfg***
- ✓ Add constructor new() function
- ✓ Add build phase
  - Call super.build_phase(phase)
  - Get the config object using uvm_config_db
- ✓ Add connect phase
  - In connect phase assign the configuration object's virtual interface to the driver's virtual interface instance
- ✓ Add run() phase method
  - In forever loop
    - ▪ Get the sequence item using seq_item_port
    - ▪ Call send_to_dut task provided
    - ▪ send the item_done to the sequencer using seq_item_port
- ✓ Understand the send_to_dut(write_xtn handle as an input argument)provided

Working Directory : wr_agent_top
Source Code : ram_wr_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.

- ✓ Extend ram_wr_monitor from uvm_monitor
- ✓ Add factory Registration
- ✓ Declare virtual interface handle with WMON_MP as modport
- ✓ Declare the ram_wr_agent_config handle as ***m_cfg***
- ✓ Add constructor new() function
- ✓ Add build phase
  - Call super.build_phase(phase)
  - Get the config object using uvm_config_db
- ✓ Add connect phase
  - In connect phase assign the configuration object's virtual interface to the monitor’s virtual interface instance
- ✓ Add run() phase method
  - In forever loop
    - ▪ Call collect_data task provided
- ✓ Understand the collect_data task provided

Working Directory **:** tb
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.

- ✓ Extend ram_env from uvm_env
- ✓ Declare the handle of ram_wr_agent with handle name as ***wr_agnth***
- ✓ Add factory registration macro
- ✓ Add code for constructor new
- ✓ Add build phase
    - create the instance of ram_wr_agent

Working Directory **:** test
Source Code **:** ram_vtest_lib.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.

- ✓ Extend ram_base_test from uvm_test
- ✓ Add factory Registration
- ✓ Declare the ram_env and ram_wr_agent_config handles as ***ram_envh*** and ***m_ram_cfg*** respectively
- ✓ Define Constructor new() function
- ✓ Add the method config_ram
    - Set is_active to UVM_ACTIVE
    - Get the virtual interface from the config database "vif"
- ✓ Add build() phase method
    - Create the instance of ram_wr_agent_config
    - Call the config_ram method
    - Set the config object into UVM config DB
    - Create the instance for ram_env

Working Directory **:** tb
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.

- ✓ Import ram_test_pkg.sv
- ✓ Import the UVM package
- ✓ Instantiate ram_if with clock as input
- ✓ Within initial
- ✓ Set the virtual interface into config database "vif" using the uvm_config_db
- ✓ Call run_test

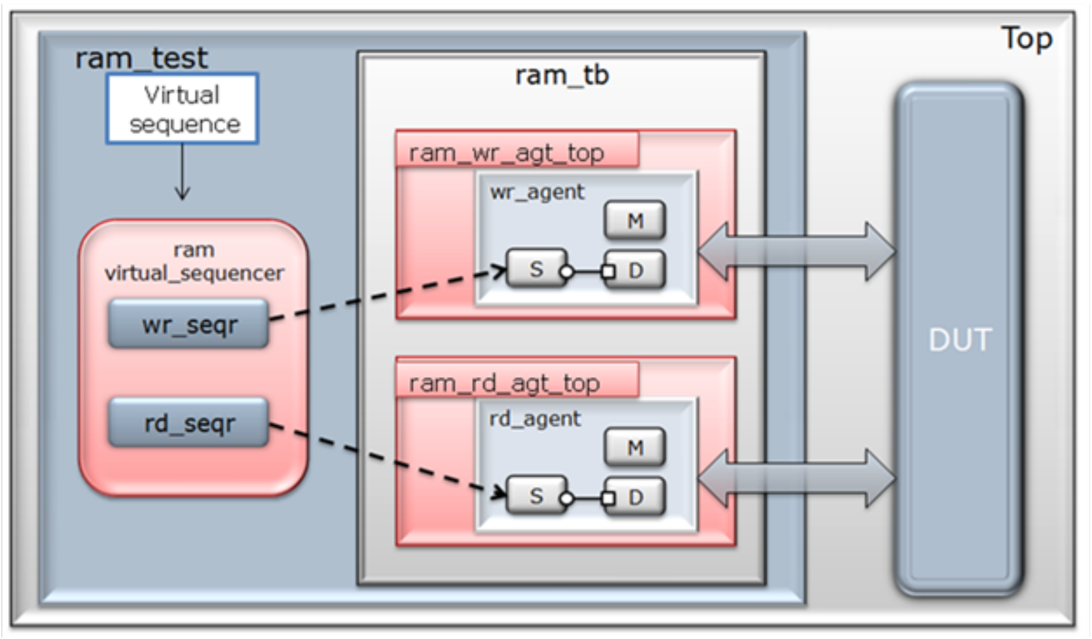
Simulation Process :

- ✓ Go to the directory **cd UVM_LABS/Lab06/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 connect the virtual interface with static interface

Lab - 7 : Agent Integration

Objective : *Integration of write agent and read agent*



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

Working Directory : wr_agt_top
Source Code : ram_wr_agent_config.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.

- ✓ Extend ram_wr_agent_config from uvm_object
- ✓ Add UVM Factory Registration Macro
- ✓ Declare parameter ***is_active*** of type uvm_active_passive_enum and assign it to UVM_ACTIVE
- ✓ Declare virtual interface handle for ram_if as ***vif***
- ✓ Declare the ***mon_rcvd_xtn_cnt*** as static int and initialize it to zero
- ✓ Declare the ***drv_data_sent_cnt*** as static int and initialize it to zero
- ✓ Add the constructor new method

Working Directory : wr_agt_top
Source Code : ram_wr_driver.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.

- ✓ Extend ram_wr_driver from uvm_driver parameterized by write_xtn
- ✓ Add factory Registration
- ✓ Declare virtual interface handle with WDR_MP as modport
- ✓ Declare the ram_wr_agent_config handle as ***m_cfg***
- ✓ Add constructor new () function
- ✓ Add build phase

- Call super.build_phase(phase)
- Get the config object using uvm_config_db

✓ Add connect phase
- In connect phase assign the configuration object's virtual interface to the driver's virtual interface instance

✓ Add run() phase method
- In forever loop
  - Get the sequence item using seq_item_port
  - Call send_to_dut task provided
  - send the item_done to the sequencer using seq_item_port

✓ Understand the send_to_dut(write_xtn handle as an input argument)provided
✓ In send_to_dut task after driving logic increment the count drv_data_sent_cnt in the configuration object
✓ Add report phase and display the drv_data_sent_cnt

Working Directory : wr_agt_top
Source Code : ram_wr_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.

✓ Extend ram_wr_monitor from uvm_monitor
✓ Add factory Registration
✓ Declare virtual interface handle with WMON_MP as modport
✓ Declare the ram_wr_agent_config handle as ***m_cfg***
✓ Add constructor new() function
✓ Add build phase
- Call super.build_phase(phase)
- Get the config object using uvm_config_db

✓ Add connect phase
- In connect phase assign the configuration object's virtual interface to the driver's virtual interface instance

✓ Add run() phase method
- In forever loop
  - Call collect_data task provided

✓ Understand the collect_data task provided
✓ Increment mon_rcvd_xtn_cnt which is in configuration class
✓ Add report phase and display the mon_rcvd_xtn_cnt

Working Directory : rd_agt_top
Source Code : ram_rd_agent_config.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.

✓ Extend ram_rd_agent_config from uvm_object
✓ Add UVM Factory Registration Macro
✓ Declare parameter ***is_active*** of type uvm_active_passive_enum and assign it to UVM_ACTIVE
✓ Declare virtual interface handle for ram_if as ***vif***
✓ Declare the ***mon_rcvd_xtn_cnt*** as static int and initialize it to zero
✓ Declare the ***drv_data_sent_cnt*** as static int and initialize it to zero
✓ Add the constructor new method

Working Directory : rd_agt_top
Source Code : ram_rd_driver.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.

- ✓ Extend ram_rd_driver from uvm_driver parameterized by read_xtn
- ✓ Add factory Registration
- ✓ Declare virtual interface handle with RDR_MP as modport
- ✓ Declare the ram_rd_agent_config handle as ***m_cfg***
- ✓ Add constructor new () function
- ✓ Add build phase
  - Call super.build_phase(phase)
  - Get the config object using uvm_config_db
- ✓ Add connect phase
  - In connect phase assign the configuration object's virtual interface to the driver's virtual interface instance
- ✓ Add run() phase method
  - In forever loop
    - Get the sequence item using seq_item_port
    - Call send_to_dut task provided
    - send the item_done to the sequencer using seq_item_port
- ✓ Understand the send_to_dut(read_xtn handle as an input argument)provided
- ✓ In send_to_dut task after driving logic increment the count drv_data_sent_cnt in the configuration object
- ✓ Add report phase and display the drv_data_sent_cnt

Working Directory : rd_agt_top
Source Code : ram_rd_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.

- ✓ Extend ram_rd_monitor from uvm_monitor
- ✓ Add factory Registration
- ✓ Declare virtual interface handle with RMON_MP as modport
- ✓ Declare the ram_rd_agent_config handle as ***m_cfg***
- ✓ Add constructor new() function
- ✓ Add build phase
  - Call super.build_phase(phase)
  - Get the config object using uvm_config_db
- ✓ Add connect phase
  - In connect phase assign the configuration object's virtual interface to the monitor's virtual interface instance
- ✓ Add run() phase method
  - In forever loop
    - Call collect_data task provided
- ✓ Understand the collect_data task provided
- ✓ Increment mon_rcvd_xtn_cnt which is in configuration class
- ✓ Add report phase and display the mon_rcvd_xtn_cnt

Working Directory : tb
Source Code : ram_virtual_sequencer.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.

- ✓ Extend ram_virtual_sequencer from uvm_sequencer parametrized with uvm_sequence_item
- ✓ Add factory Registration
- ✓ Declare handles for ram_wr_sequencer and ram_rd_sequencer as ***wr_seqrh*** & ***rd_seqrh***
- ✓ Add constructor new() function

Working Directory : tb
Source Code : ram_virtual_seqs.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.

- ✓ Extend ram_vbase_seq from uvm_sequence parametrized with uvm_sequence_item
- ✓ Add factory Registration
- ✓ Declare handles for ram_wr_sequencer, ram_rd_sequencer and ram_virtual_sequencer as ***wr_seqrh, rd_seqrh, vsqrh***
- ✓ Add constructor new() function
- ✓ Add task body
  - Assign virtual sequencer handle(m_sequencer) to vsqrh
    - Hint : Use $cast(vsqrh, m_sequencer)
  - Assign the sub-sequencer handles with the sub-sequencer handles of the virtual sequencer
- ✓ Extend ram_single_vseq from ram_vbase_seq
- ✓ Add factory Registration
- ✓ Add constructor new() function
- ✓ Add task body
  - Call super.body();
  - Create the instance of ram_single_addr_wr_xtns & ram_single_addr_rd_xtns
  - Start write and read sequences on respective sequencers
- ✓ Repeat the above steps for the remaining 3 sequences i.e ten, odd & even vsequences

Working Directory : tb
Source Code : ram_tb.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

- ✓ Extend ram_tb from uvm_env
- ✓ Add Factory Registration
- ✓ Declare handles for ram_wr_agt_top, ram_rd_agt_top and ram_virtual_sequencer as ***wagt_top,ragt_top and v_sequencer*** respectively
- ✓ Declare handle for ram_env_config object as ***m_cfg***
- ✓ Add constructor new() function
- ✓ Add build phase
  - Get configuration object ram_env_config from database using uvm_config_db()
  - If ram_env_config parameter has_wagent=1
    - set m_cfg.m_wr_cfg into config database "ram_wr_agent_config" using uvm_config_db
    - Create instance for ram_wr_agt_top

- If ram_env_config parameter has_ragent=1
  - ▪ Set m_cfg.m_rd_cfg into config database "ram_rd_agent_config" using uvm_config_d
  - ▪ Create instance for ram_rd_agt_top
- If ram_env_config parameter has_virtual_sequencer=1
  - ▪ Create the instance of v_sequencer handle

✓ Add connect phase
- If ram_env_config parameter has_virtual_sequencer=1
  - ▪ Connect virtual sequencers to UVC sequencers
  - ▪ Hint : v_sequencer.wr_seqr = wagt_top.agnth.seqrh
  - ▪ v_sequencer.rd_seqr = ragt_top.agnth.seqrh

Working Directory : test
Source Code : ram_vtest_lib.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.

✓ Extend ram_base_test from uvm_test
✓ Add factory Registration
✓ Declare the handles ram_tb, ram_env_config, ram_wr_agent_config and rd_agent_config as ***ram_envh, m_tb_cfg, m_wr_cfg & m_rd_cfg***
✓ Declare ***has_ragent=1 & has_wagent=1*** as int data type
✓ Define Constructor new() function
✓ Add the method config_ram
- In ram_wr_agent_config object set is_active to UVM_ACTIVE
- Get the virtual interface from the config database "vif"
- Assign m_wr_cfg to m_tb_cfg.m_wr_cfg
- In ram_rd_agent_config object set is_active to UVM_ACTIVE
- Get the virtual interface from the config database "vif"
- Assign m_rd_cfg to m_tb_cfg.m_rd_cfg
- Assign local has_wagent & has_ragent variables to the variables in ram_env_config
- set the m_tb_cfg object into UVM config DB "ram_env_config"

✓ Add build() phase method
- Create the instance for ram_env_config
- Create the instance for ram_wr_agent_config
- Create the instance for ram_rd_agent_config
- Call the config_ram method
- Create the instance for ram_tb

✓ Extend ram_single_addr_test from ram_base_test
✓ Add Factory Registration
✓ Declare the handle for ram_single_vseq virtual sequence
✓ Add constructor new() function
✓ Add build phase
- Call build phase of ram_base_test

✓ Add run_phase
- Raise objection
- Create an instance for sequence
- Start the sequence on virtual sequencer
- Drop objection

✓ Repeat the above steps for the remaining 3 test cases i.e ten, odd & even test cases

Working Directory : tb

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.

- ✓ Within initial
- ✓ Set the virtual interface into config database "vif" using the uvm_config_db
- ✓ Call run_test

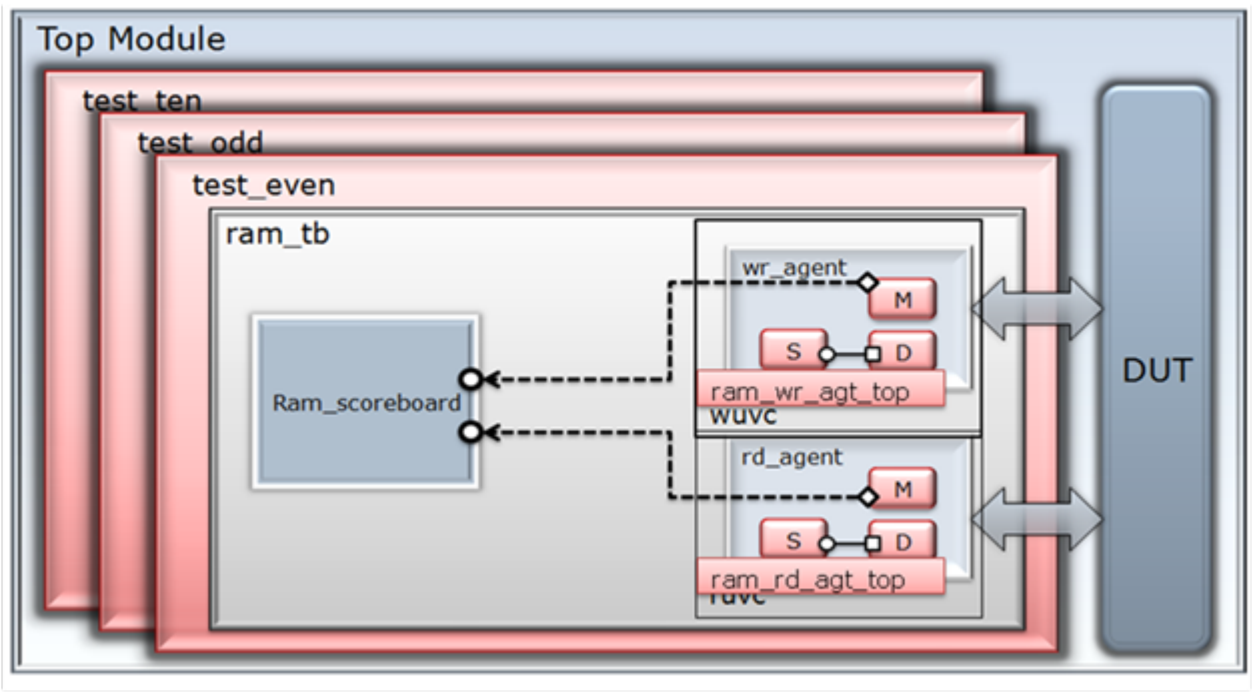
Simulation Process :

- ✓ Go to the directory **cd UVM_LABS/Lab07/sim**
- ✓ Call the target run_test to run ram_single_addr_test: **make run_test**
- ✓ Call the target run_test1 to run ram_ten_addr_test: **make run_test1**
- ✓ Call the target run_test2 to run ram_odd_addr_test: **make run_test2**
- ✓ Call the target run_test3 to run ram_even_addr_test: **make run_test3**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes :
How to integrate write agent, read agent and virtual sequencer

Lab - 8 : Scoreboard

Objective : *Scoreboard Implementation and Integration*



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

Working Directory : tb

Source Code : ram_scoreboard.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.

- ✓ Extend ram_scoreboard from uvm_scoreboard
- ✓ Declare handles for uvm_tlm_analysis_fifos parameterized by read & write transactions as ***fifo_rdh & fifo_wrh*** respectively
  - Hint: uvm_tlm_analysis_fifo #(read_xtn) fifo_rdh
  - uvm_tlm_analysis_fifo #(write_xtn) fifo_wrh
- ✓ Add the following integers for Scoreboard Statistics
  - ***wr_xtns_in*** : calculates number of write transactions
  - ***rd_xtns_in :*** calculates number of read transactions
  - ***xtns_compared*** : number of xtns compared
  - ***xtns_dropped*** : calculates number of xtns failed
  - Add factory registration
  - Declare an Associative array as a reference model type logic [63:0] and index type int
  - Declare handles of type write_xtn & read_xtn as ***wr_data & rd_data*** to store the tlm_analysis_fifo data
  - Add Constructor function
  - Create instances of uvm_tlm_analysis fifos inside the constructor using new("fifo_h", this)

  - Explore mem_write method to write write_xtn into ref model
  - Explore mem_read method to read read_xtn from ref model

- ✓ Add run phase
  - In forever loop
    - ▪ Get and print the write data using the tlm fifo
    - ▪ Call the method mem_write
  - In forever loop
    - ▪ Get and print the read data using the tlm fifo
    - ▪ Call the method check_data
- ✓ Explore method check_data

Working Directory : tb
Source Code : ram_tb.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

- ✓ Extend ram_tb from uvm_env
- ✓ Add Factory Registration
- ✓ Declare handles for ram_wr_agt_top, ram_rd_agt_top and ram_virtual_sequencer as ***wagt_top,ragt_top and v_sequencer*** respectively
- ✓ Declare handle for ram scoreboard as ***sb***
- ✓ Declare handle for ram_env configuration class as ***m_cfg***
- ✓ Add constructor new() function
- ✓ Add build phase
  - Get configuration object ram_env_config from database using uvm_config_db()
  - If ram_env_config parameter has_wagent=1
    - ▪ set m_cfg.m_wr_cfg into config database "ram_wr_agent_config" using uvm_config_db
    - ▪ Create instance for ram_wr_agt_top
  - If ram_env_config parameter has_ragent=1
    - ▪ Set m_cfg.m_rd_cfg into config database "ram_rd_agent_config" using uvm_config_d
    - ▪ Create instance for ram_rd_agt_top
  - If ram_env_config parameter has_virtual_sequencer=1
    - ▪ Create the instance of v_sequencer handle
  - If ram_env_config parameter has_scoreboard=1
    - ▪ Create the instance of scoreboard handle

- ✓ Add connect phase
  - If ram_env_config parameter has_virtual_sequencer=1
    - ▪ Connect virtual sequencers to UVC sequencers
    - ▪ Hint : v_sequencer.wr_seqr = wagt_top.wr_agnth.seqrh
    - ▪ v_sequencer.rd_seqr = ragt_top.rd_agnth.seqrh
  - If ram_env_config parameter has_scoreboard=1
    - ▪ Connect the monitor analysis port to scoreboard's uvm_tlm fifo's analysis export
    - ▪ Hint:
    - ▪ wagt_top.agnth.monh.monitor_port.connect(sb.fifo_wrh.analysis_export)
    - ▪ ragt_top.agnth.monh.monitor_port.connect(sb.fifo_rdh.analysis_export)

Working Directory : rd_agt_top
Source Code : ram_rd_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.

- ✓ Declare analysis TLM port to connect the monitor to the scoreboard
- ✓ Create an object for handle monitor_port using new in the class constructor
- ✓ In the collect_data task provided call write method of analysis port after collecting the data

Working Directory : wr_agt_top
Source Code : ram_wr_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.

- ✓ Declare analysis TLM port to connect the monitor to the scoreboard
- ✓ Create an object for handle monitor_port using new in the class constructor
- ✓ In the collect_data task provided call write method of analysis port after collecting the data

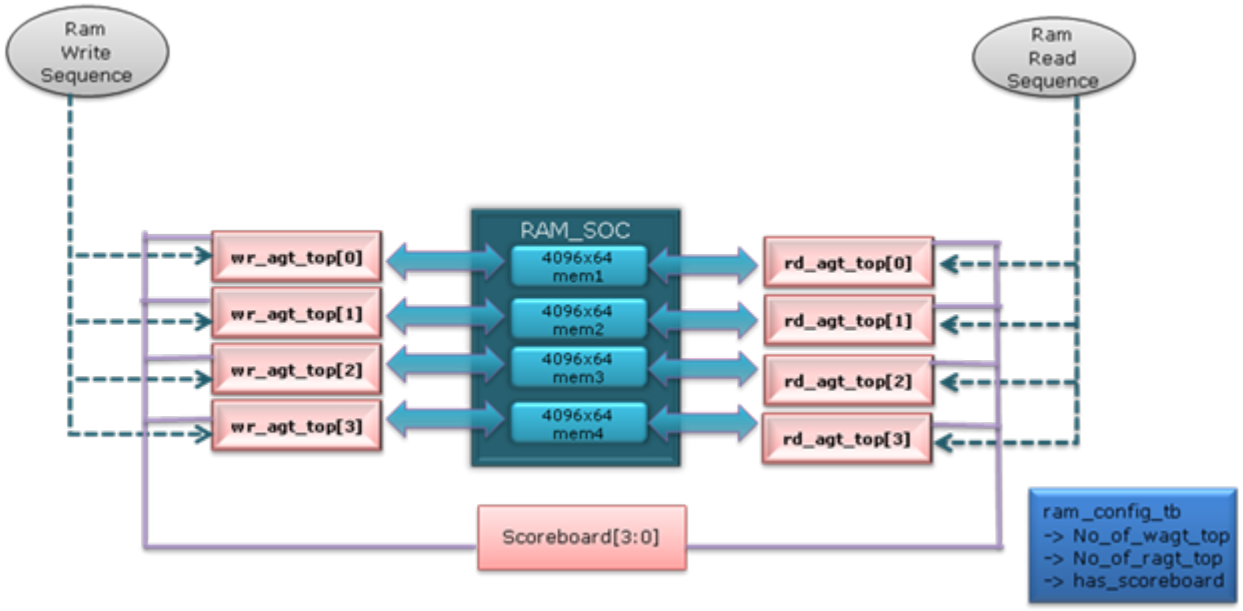
Simulation Process :

- ✓ Go to the directory **cd UVM_LABS/Lab08/sim**
- ✓ Call the target run_test to run ram_single_addr_test: **make run_test**
- ✓ Call the target run_test1 to run ram_ten_addr_test: **make run_test1**
- ✓ Call the target run_test2 to run ram_odd_addr_test: **make run_test2**
- ✓ Call the target run_test3 to run ram_even_addr_test: **make run_test3**
- ✓ Observe the output and also cross-check with the solution source code.

Learning outcomes:
How to Implement the scoreboard & connect with the agent's monitor analysis port

Lab - 9 : SOC Implementation

Objective : *Integration of Multiple Read Agents and Write Agents*



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

Working Directory : tb
Source Code : ram_virtual_sequencer.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 ram_virtual_sequencer declare dynamic array of handles for ram_wr_sequencer and ram_rd_sequencer as ***wr_seqrh[] & rd_seqrh[]***
- ✓ Declare handle for ram_env_config
- ✓ In build phase
  - Create dynamic array handles wr_seqrh & rd_seqrh equal to the config parameter no_of_duts

Working Directory : tb
Source Code : ram_virtual_seqs.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 ram_vbase_seq declare dynamic array of handles for ram_wr_sequencer and ram_rd_sequencer as ***wr_seqrh[] & rd_seqrh[]***
- ✓ Declare handle for ram_env_config
- ✓ In task body get the config object ram_env_config from database using uvm_config_db
- ✓ Initialize the dynamic arrays for write & read sequencers to m_cfg.no_of_duts
- ✓ Assign ram_wr_sequencer & ram_rd_sequencer handles to virtual sequencer's ram_wr_sequencer & ram_rd_sequencer handles
  - Hint : use foreach loop
- ✓ In the extended class ram_single_vseq

- In task body()
  - Create the instances for ram_single_addr_wr_xtns & ram_single_addr_rd_xtns

  - Within for loop start the write and read sequences on all the corresponding write and read sequencers

✓ Repeat the above steps for the remaining 3 sequences i.e ten, odd & even vsequences

Working Directory : tb
Source Code : ram_env_config.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 of handles for the sub_components ram_wr_agent_config & ram_rd_agent_config as ***m_wr_agent_cfg & m_rd_agent_cfg***
✓ Declare variable no_of_duts as int which can be set to the required dut value

Working Directory : tb
Source Code : ram_tb.sv
Instructions : The following instructions have been included in the source code as comments. Refer the comments in the source code and edit the source code

✓ In ram_tb class
✓ Declare dynamic array of handles for ram_wr_agt_top, ram_rd_agt_top as ***wagt_top,ragt_top*** and respectively
✓ Declare dynamic array of handles for ram scoreboard as ***sb***
✓ In build phase
- If ram_env_config parameter has_wagent=1
  - Initialize the dynamic array wagt_top[] to m_cfg.no_of_duts
  - Inside a foreach loop of wagt_top[i]
  - set ram_wr_agent_config into the database using the ram_env_config's ram_wr_agent_config object
  - Create the instances of wagt_top[i] handles
- If ram_env_config parameter has_ragent=1
  - Initialize the dynamic array ragt_top[] to m_cfg.no_of_duts
  - Inside a foreach loop of ragt_top[i]
  - set ram_rd_agent_config into the database using the ram_env_config's ram_rd_agent_config object
  - Create the instances of ragt_top[i] handles
- If ram_env_config parameter has_scoreboard=1
  - Initialize the dynamic array sb[] to m_cfg.no_of_duts
  - Inside a foreach loop of ragt_top[i]
  - Create the instances of sb[i] handles

✓ Add connect phase
- Connect virtual sequencer's sub sequencers to the envirnoment's write & read sequencers
  - Inside a foreach loops for *agt_top[i]
  - Hint : v_sequencer.wr_seqrh[i] = wagt_top[i].agnth
  - v_sequencer.rd_seqrh[i] = ragt_top[i].agnth
- Connect the corressponding analysis port of all the monitors to the analysis export of all the tlm analysis fifo's in the scoreboard
  - Inside a foreach loops for *agt_top[i]
  - Hint:wagt_top[i].agnth.monh.monitor_port.connect(sb[i].fifo_wrh.analysis_export)

- ragt_top[i].agnth.monh.monitor_port.connect(sb[i].fifo_rdh.analysis_export)

Working Directory : test

Source Code : ram_vtest_lib.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 ram_base_test
  - Declare dynamic array of handles for ram_wr_agent_config & ram_rd_agent_config as ***m_wr_cfg[] & m_rd_cfg[]***
- ✓ In function config_ram
  - Initialize the dynamic array of handles for ram_rd_agent_config equal to no_of_duts(As shown for ram_wr_agent_config)
  - Create instances for ram_rd_agent_config
  - For all the configuration objects, set the following parameters
    - is_active to UVM_ACTIVE
    - Get the virtual interface from the config database
  - Assign the ram_rd_agent_config handle to the environment config's (ram_env_config) ram_rd_agent_config handle
  - Assign no_of_duts to local m_tb_cfg.no_of_duts
  - Assign has_ragent to local m_tb_cfg.has_ragent
  - Assign has_wagent to local m_tb_cfg.has_wagent
- ✓ In build() phase method
  - Initialize the dynamic array of handles for ram_rd_agent_config equal to no_of_duts (as shown for ram_wr_agent_config)
  - Call function config_ram which configures all the parameters
  - Set the env config object into UVM config DB

Working Directory : tb

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.

- ✓ Instantiate 4 ram_if interface instances in0,in1,in2,in3 with clock as input
- ✓ Within initial begin
  - Set the virtual interface instances as strings vif_0,vif_1,vif_2,vif_3 using the uvm_config_db
  - Call run_test

Simulation Process :

- ✓ Go to the directory **cd UVM_LABS/Lab09/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 build a reusable configurable testbench environment with multiple agents

Lab - 10 : Coverage and Regression

Objective : *Adding Functional Coverage and running the Regression Simulation*

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

Working Directory : tb
Source Code : ram_scoreboard.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 ram_scoreboard class
  - Declare handles for read & write coverage data as read_cov_data & write_cov_data of type read_xtn & write_xtn respectively
  - Write the covergroup ***ram_fcov1*** for write transactions
  - Write the covergroup ***ram_fcov2*** for read transactions
  - (Note : The covergroup is provided)
  - In constructor create the instances for covergroup ram_fcov1 & ram_fcov2
  - In the run phase
  - Sample the covergroups appropriately as per comments showed in lab exercise based on write or read

Simulation Process :

- ✓ Go to the directory **cd UVM_LABS/Lab10/sim**
- ✓ Call the target run_test to run ram_single_addr_test: **make run_test**
- ✓ Call the target run_test1 to run ram_ten_addr_test: **make run_test1**
- ✓ Call the target run_test2 to run ram_odd_addr_test: **make run_test2**
- ✓ Call the target run_test3 to run ram_even_addr_test: **make run_test3**
- ✓ Call make regress finally and check the coverage report
- ✓ Observe the output and also cross check with the solution source code.

Learning outcomes:
Verification Sign-off: Achieving the coverage closure by writing various test cases