Assignment 5 - UVM

This assignment is to practice writing a UVM testbench environment for the ALSU design. Check the notes and classwork codes uploaded on Google classroom to do the assignment. The assignment will be split into three parts. Use a do file in the 3 parts (I have attached a do file for you to use and modify). Don't forget to import uvm_pkg and uvm_macros in **all** files. **Each class created will be in a separate file and in a package.** Note that in the do file, the vlog command is " vlog -f src_files.list", which means that you need to create a file named src_files.list that has the files names to be compiled. Check the classwork files for your reference.

Part #1:

In this part, you will create a top module that will start a UVM test. UVM test will build the UVM environment and then displays a message.

Files to create:

1. top module
2. alsu_test
3. alsu_env

Steps:

1. Create top module where you will instantiate the DUT, interface, assign the interface to the DUT, generate the clock and in an initial block use the global task run_test to run your uvm test environment (alsu_test).
2. Create a class named alsu_test in a separate file in a package, Inside the test class, you will build the environment component (alsu_env) in the build_phase. In the run phase, raise the objection to display a message "Inside the ALSU test" using `uvm_info. Drop the objection after it. Don't forget to import the alsu_env package.
3. Create a class named alsu_env. No phases will be overridden in the environment.

Deliverables:

1. Simulate the top module and make sure that the message is displayed. Take a screenshot to use it in your PDF.

Part #2:

In this part, you will pass the virtual interface to the UVM driver to drive the interface.

Files to create:

1. top module
2. alsu_test

3. alsu_env
4. alsu_driver

Steps:

1. Create top module where you will instantiate the DUT, interface, assign the interface to the DUT, generate the clock. Inside the initial block, set the virtual interface in the configuration database (uvm_config_db) using the set method then use the global task run_test to run your uvm test environment (alsu_test).
2. Create a class named alsu_test in a separate file in a package, Inside the test class
    a. Declare a virtual interface named **alsu_test_vif** and handle of the alsu_env
    b. In the build_phase, you will do the following
        i. Retrieve the virtual interface from the configuration database (uvm_config_db) and pass the value of the virtual interface to **alsu_test_vif**
        ii. Set the virtual interface **alsu_test_vif** in the configuration database to all the components under the test class so that any component can retrieve it.
        iii. Build the environment component (alsu_env).
    c. In the run phase, raise the objection, wait for #100 then display a message "Inside the ALSU test" using `uvm_info. Drop the objection after it. Don't forget to import the alsu_env package.
3. Create a class named alsu_env. Inside the build_phase, build the driver (Don't forget to import the driver package)
4. Create a class named alsu_driver extends uvm_driver
    a. Declare a virtual interface named **alsu_driver_vif**
    b. In the build_phase:
        i. Retrieve the virtual interface set by the alsu_test from the configuration database (uvm_config_db) and pass the value of the virtual interface to **alsu_driver_vif**
    c. In the run_phase:
        i. Reset the ALSU, then in a forever loop use $random to randomize the inputs of the ALSU using the virtual interface **alsu_driver_vif.**

Deliverables:

1. Simulate the top module and make sure that the message is displayed. Take a screenshot to use it in your PDF.
2. Add the signals of the interface to wave and make sure that the inputs are driven. Take a screenshot and add it to you PDF.

Part #3:

In this part, you will use a configuration object to store the virtual interface value such that any component can retrieve the configuration object and read the values set in it.

Files to create:

1. top module
2. alsu_test
3. alsu_env
4. alsu_driver
5. alsu_config_obj

Steps:

1. Create top module where you will instantiate the DUT, interface, assign the interface to the DUT, generate the clock. Inside the initial block, set the virtual interface in the configuration database (uvm_config_db) using the set method then use the global task run_test to run your uvm test environment (alsu_test).
2. Create a class named alsu_config_obj that extends uvm_object
    a. Inside this UVM object, you will declare a virtual interface named **alsu_config_vif.**
3. Create a class named alsu_test in a separate file in a package, Inside the test class
    a. Declare a configuration object handle from alsu_config_obj named **alsu_config_obj_test** and UVM env handle from alsu_env
    b. In the build_phase, you will do the following
        i. Retrieve the virtual interface from the configuration database (uvm_config_db) and pass the value of the virtual interface to **alsu_config_obj_test.alsu_config_vif**
        ii. Set the configuration object in the configuration database to all the components under the test class so that any component can retrieve it.
        iii. Build the environment component (alsu_env).
    c. In the run phase, raise the objection, wait for #100 then display a message "Inside the ALSU test" using `uvm_info. Drop the objection after it. Don't forget to import the alsu_env package.
4. Create a class named alsu_env. Inside the build_phase, build the driver (Don't forget to import the driver package)
5. Create a class named alsu_driver extends uvm_driver
    a. Declare a virtual interface named **alsu_driver_vif** and a configuration object named **alsu_config_obj_driver**
    b. In the build_phase:

    i. Retrieve the configuration object set by the alsu_test from the configuration database (uvm_config_db) and pass the value of the configuration object to **alsu_config_obj_driver**

c. In the connect phase:
    i. Assign **alsu_config_obj_driver.** alsu_config_vif to **alsu_driver_vif**

d. In the run_phase:
    i. Reset the ALSU, then in a forever loop use $random to randomize the inputs of the ALSU using the virtual interface **alsu_driver_vif.**

Deliverables:

1. Simulate the top module and make sure that the message is displayed. Take a screenshot to use it in your PDF.
2. Add the signals of the interface to wave and make sure that the inputs are driven. Take a screenshot and add it to you PDF.