Assignment 6 - 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 ALSU design was shared in assignment 3. The assignment will be split into two parts. 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.**

Part #1:

Full ALSU environment:

- Create a full environment for the ALSU design, the same environment done for the shift register.
- Copy the constraints done in assignment 3 in the sequence item class.
- Copy the covergroups, coverpoints done in the coverage collector class.
- Add assertions file and bind it in the top module, assertions should check the functionality for the out and leds checking all cases, valid and invalid.

Part #2:

Adding the shift register to the ALSU:

- After completing the ALSU env, you will have 2 environments created, shift reg and ALSU environments.
- Adjust the ALSU to instantiate the shift register to execute the shift and rotate operations.
    - For simplicity, remove the clock and reset from the design and verification environment of the shift register, replace @(negedge clk) in the shift register driver and monitor and replace it with #2. Remove the reset sequence.
    - Datain port of the shift register will be connected to the out port of the ALSU
    - Dataout port of the shift register will be connected to a new internal 6-bit bus that you will create named out_shift_reg
    - Assign the value of out_shift_reg to the out in shift and rotate valid operations
- Since the shift register is now part of the ALSU, and we have 2 separate environments for both, you will add the 2 environments in the ALSU test, where the ALSU environment will drive the interface of the ALSU, and the shift register environment will be passive and will not drive data to the shift register. The shift register environment will only monitor the interface of the shift register and send the data to its scoreboard and coverage collector.

Steps to add the 2 environments in the ALSU test:

1. Add the shift register and ALSU interfaces in the top module.
2. Pass the interfaces in config db.
3. Connect shift register interface using assign statements. For example:
    - assign shift_if.serial_in = alsu_if.serial_in;
4. Add 2 environments to the test.
5. The test will create 2 config objects, one for each environment.
6. Each config object will have 2 variables.
    - Virtual interface
    - Variable named active of datatype uvm_active_passive_enum, this is an enum defined in the uvm package that will take the values UVM_ACTIVE or UVM_PASSIVE
7. The test will set the 2 interfaces to each configuration object
8. The test will set the active variable of the ALSU configuration object to UVM_ACTIVE and will set the active variable of the shift register configuration object to UVM_PASSIVE
9. ALSU will be an active agent, driving the interface while the shift register agent will be a passive agent.
10. In each agent, you will retrieve the configuration object in the build phase
    - In the build phase, add if condition checking if the config_obj.active variable is equal to UVM_ACTIVE, if yes, then build the driver and sequencer
    - In the connect phase, add if condition checking if the config_obj.active variable is equal to UVM_ACTIVE, if yes, then connect the driver and sequencer and also connect the virtual interface of the driver

Deliverables:

1. Take the screenshots for part #1 and part #2 UVM testbench, 2 designs, and assertion file of the ALSU.
2. Simulate the top module and make sure that the message is displayed. Take a screenshot to use it in your PDF.
3. Add the signals of the two interfaces to wave and make sure that the inputs are driven. Take a screenshot and add it to your PDF.
4. Take screenshots of the code coverage, functional coverage, sequential domain (assertions) coverage for both ALSU and shift register

Extra:

This part is to exercise how to use the UVM factory to substitute one data type by another data type.

In the previous part, you have created one sequence item class, let's assume that you have named it alsu_seq_item. This class now has constraint blocks that generate valid and invalid cases. Now you will split them into 2 sequence item classes. The first sequence item class is the same class that you have created alsu_seq_item which extends uvm_sequence_item, and the second class is alsu_seq_item_valid_invalid which will extends alsu_seq_item.

Revisit the constraint blocks of the alsu_seq_item, if the constraint block generates only valid cases then leave it. If the constraint block generates valid and invalid cases then copy it into alsu_seq_item_valid_invalid class, and then adjust it to send only valid cases

How to replace the alsu_seq_item that you are using in the ALSU environment by alsu_seq_item_valid_invalid?

Since both classes are registered in the UVM factory then add the following method call in the test build_phase:

set_type_override_by_type(alsu_seq_item::get_type(), alsu_seq_item_valid_invalid::get_type());

The above call will replace one type with another type whenever any object is created using the type_id::create method.

Deliverables:

Run the UVM testbench without the override method call and then rerun after adding the override method. Take screenshots of the waveforms showing that the first run generates valid cases only, then take screenshots of the waveforms that the second run generates valid and invalid cases