Data Types & Constrained Random Assignment

Check the deliverables section by the end of the document for questions 2, and 3.

1) Write a module to test dynamic array data type and its predefined methods. Run Questasim to make sure the display statements are working as expected.
   - declare two dynamic arrays dyn_arr1, dyn_arr2 of type int
   - initialize dyn_arr2 array elements with (9,1,8,3,4,4)
   - allocate six elements in array dyn_arr1 using new operator
   - initialize array dyn_arr1 with index as its value using foreach
   - display dyn_arr1 and its size
     - Expected output: (0,1,2,3,4,5), 6
   - delete array dyn_arr1
   - reverse, sort, reverse sort and shuffle the array dyn_arr2 and display dyn_arr2 after using each method
     - Expected output: (4,4,3,8,1,9), (1,3,4,4,8,9), (9,8,4,4,3,1), <shuffled_array>

2) Verify the functionality of the following counter:
   - **Parameters:**
     1. WIDTH: width of the data_load and count_out ports (Valid values: 4, 6, 8, default: 4)
   - **Inputs**:
     1. clk
     2. rst_n (active low sync rst)
     3. load_n (active low load)
     4. up_down (When this input is high then increment counter, else decrement the counter)
     5. ce (clock enable signal to increment or decrement the counter depending on the up_down)
     6. data_load (load data to count_out output when the load_n signal is asserted)
   - **Outputs**:
     1. count_out (counter output)
     2. max_count (1-bit, When the counter reaches the maximum value, this signal is high, else low)
     3. zero (1-bit, When the counter reaches the minimum value, this signal is high, else low)

Requirements:

1. Create a verification plan document based on your verification plan items to support your verification planning, an example of the document can be found in the link here. Please copy this document to have your own version.

Below is a snippet of the verification plan for reference:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
COUNTER_1	When the reset is asserted, the output counter value should be low	Directed at the start of the sim, then randomized with constraint that drive the reset to be off most of the simulation time.	-	A checker in the testbench to make sure the output is correct
COUNTER_2	When the load is asserted, the output count_out should take the value of the load_data input	Randomization under constraints on the load signal to be off 70% of the time	-	A checker in the testbench to make sure the output is correct

2. Create a package that parameter WIDTH with value 4, and add a class with the following constraints with the labels from the verification plan above each relevant item below
    a. Constraint the reset to be deactivated most of the time
    b. Constraint the load signal to be active 70% of the time
    c. Constraint the enable signal to be active 70% of the time

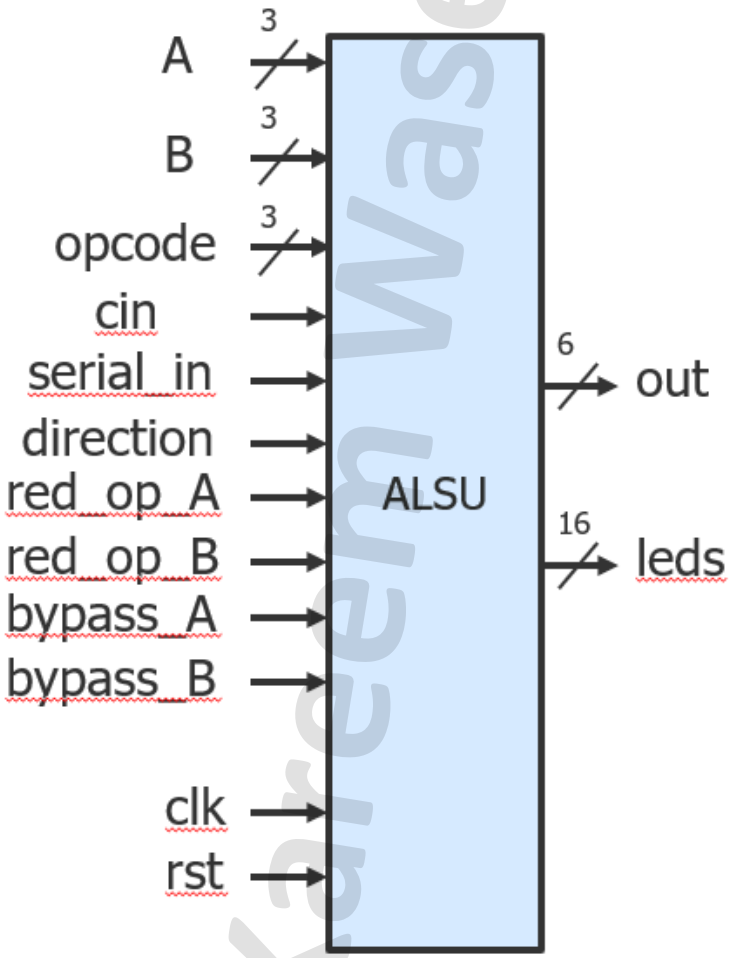
3. Create a self-checking testbench and in the initial block do the following steps:
    - **Assert the reset** using the assert_reset task.
    - **Repeat the following steps** in a loop:
        - Randomize the class object to apply the above constraints.
        - Pass the randomized values to the design stimulus.
        - Compute the expected outputs using the golden_model task.
        - Call the check_result task to compare the design and expected output values.

Notes:

- Override the parameter of the design instantiated in the testbench with WIDTH parameter declared in the package.
- You are free to add more constraints to enrich your verification to reach **100% statement, branch, and toggle coverage.**

3) ALSU is a logic unit that can perform logical, arithmetic, and shift operations on input ports

- Input ports A and B have various operations that can take place depending on the value of the opcode.
- Each input bit except for the clk and rst will be sampled at the rising edge before any processing so a D-FF is expected for each input bit at the design entry.
- The output of the ALSU is registered and is available at the rising edge of the clock.



Inputs

Each input bit except for the clk and rst will have a DFF in front of its port. Any processing will take place from the DFF output.

Input	Width	Description
clk	1	Input clock
rst	1	Active high asynchronous reset
A	3	Input port A
B	3	Input port B
cin	1	Carry in bit, only valid to be used if the parameter FULL_ADDER is “ON”
serial_in	1	Serial in bit, used in shift operations only
red_op_A	1	When set to high, this indicates that reduction operation would be executed on A rather than bitwise operations on A and B when the opcode indicates OR and XOR operations
red_op_B	1	When set to high, this indicates that reduction operation would be executed on B rather than bitwise operations on A and B when the opcode indicates OR and XOR operations
opcode	3	Opcode has a separate table to describe the different operations executed
bypass_A	1	When set to high, this indicates that port A will be registered to the output ignoring the opcode operation
bypass_B	1	When set to high, this indicates that port B will be registered to the output ignoring the opcode operation
direction	1	The direction of the shift or rotation operation is left when this input is set to high; otherwise, it is right.

Outputs and parameters

Output	Width	Description
leds	16	When an invalid operation occurs, all bits blink (bits turn on and then off with each clock cycle). Blinking serves as a warning; otherwise, if a valid operation occurs, it is set to low.
out	6	Output of the ALSU

Parameter	Default value	Description
INPUT_PRIORITY	A	Priority is given to the port set by this parameter whenever there is a conflict. Conflicts can occur in two scenarios, red_op_A and red_op_B are both set to high or bypass_A and bypass_B are both set to high. Legal values for this parameter are A and B
FULL_ADDER	ON	When this parameter has value “ON” then cin input must be considered in the addition operation between A and B. Legal values for this parameter are ON and OFF

Opcodes & Handling invalid cases

Invalid cases

1. Opcode bits are set to 110 or 111
2. red_op_A or red_op_B are set to high and the opcode is not OR or XOR operation

Output when invalid cases occurs

1. leds are blinking
2. out bits are set to low, but if the bypass_A or bypass_B are high then the output will take the value of A or B depending on the parameter INPUT_PRIORITY

Opcode	Operation
000	OR
001	XOR
010	ADD
011	MULT
100	SHIFT (Shift output by 1 bit)
101	ROTATE (Rotate output by 1 bit)
110	Invalid opcode
111	Invalid opcode

Testbench:

- You are required to verify the functionality of the ALSU under the default parameters.

Requirements:

1. Create a verification plan document to support your verification planning, an example of the document can be found in the link here. Please copy this document to have your own version and fill the document with your verification requirements.
    a. Below is a snippet of the verification plan for reference:

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
ALSU_1	When the reset is asserted, the outputs should be low	Directed at the start of the sim, then randomized with constraint that drive the reset to be off most of the simulation time.	-	A checker in the testbench to make sure the output is correct
ALSU_2	Incase of invalid cases do not occur, when opcode is add, then out should perform the addition on ports A and B taking cin if parameter FULL_ADDER is high	Randomization under constraints on the A and B to have the maximum, minimum and zero values most of the time	-	Output Checked against golden model

2. Add comments in your testbench and class with the labels taken from your verification plan document for easy tracking of the implementation of constraints.

Create a package that have a user defined enum opcode_e that takes the value of the opcode, you can name the invalid cases INVALID_6 and INVALID_7.

3. Create a class in the package to randomize the design inputs under the following constraints with the labels from the verification plan above each relevant item below
    1. Reset to be asserted with a low probability that you decide.
    2. Constraint for adder inputs (A, B) to take the values (MAXPOS, ZERO and MAXNEG) more often than the other values when the opcode is addition or multiplication.
    3. If the opcode is OR or XOR and red_op_A is high, constraint the input A most of the time to have one bit high in its 3 bits while constraining the B bits to be low
    4. If the opcode is OR or XOR and red_op_B is high, constraint the input B most of the time to have one bit high in its 3 bits while constraining the A bits to be low
    5. Invalid cases should occur less frequent than the valid cases
    6. bypass_A and bypass_B should be disabled most of the time
    7. Do not constraint the inputs A or B when the operation is shift or rotate

4. In your testbench:
    - Loop to randomize the inputs applying the above constraints.
    - Check the code coverage report and modify the testbench as necessary to achieve 100% code coverage for statement, branch and toggle. Any exclusions must be justified.
    - You can either create a Verilog design to instantiate as a golden model or create a task named golden_model to return the expected output values.
    - Report any bugs detected in the design and fix them.

Note: You are free to add more constraints to enrich your verification. If you will add any, you must document this in your verification document.

Deliverables:

One PDF file having the following

1. Testbench code
2. Package code
3. Design code
4. Report any bugs detected in the design and fix them
5. Snippet to your verification plan document
6. Do file
7. Code Coverage report snippets
8. **Clear and neat** QuestaSim waveform snippets showing the functionality of the design
9. Your PDF file must have this format <your_name>_Assignment2 for example Kareem_Waseem_Assignment2