Data types & Constrained Random – Extra

1)

Write the SystemVerilog code to:

1. Declare a 2-state array, my_array, that holds four 12-bit values
2. initialize my_array in initial block so that:
    - my_array[0] = 12'h012
    - my_array[1] = 12'h345
    - my_array[2] = 12'h678
    - my_array[3] = 12'h9AB
3. Traverse my_array and print out bits [5:4] of each 12-bit element
    - Using a for loop
    - Using a foreach loop

2) We will verify the same ALU given in assignment 1 but using randomization and typedef enum.

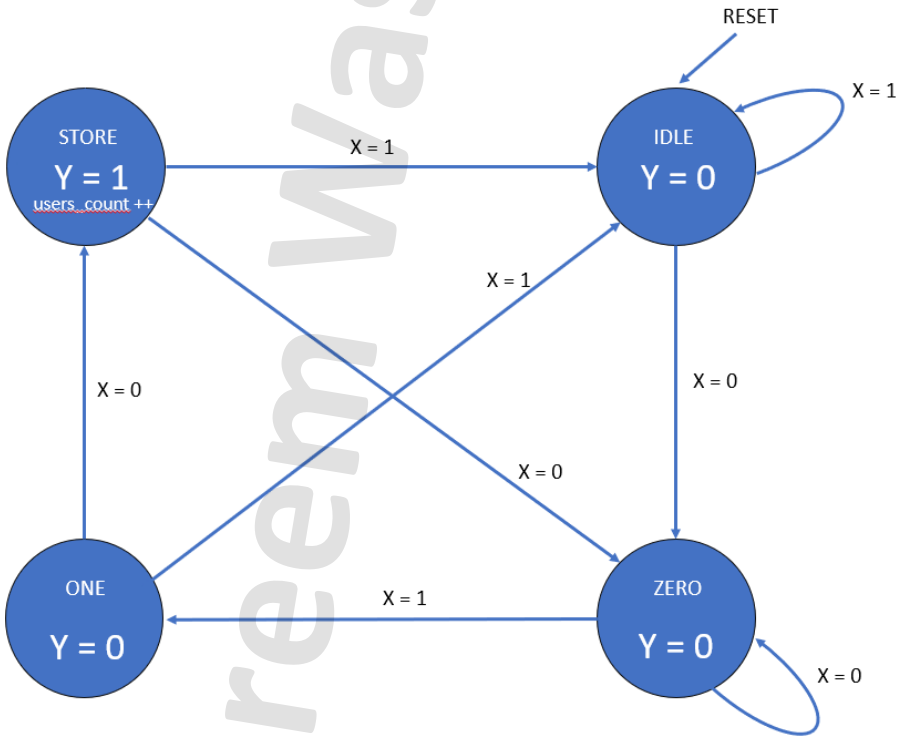
Follow the following steps:

- Create a package in a file that has the following
    - typedef enum for the opcode
    - Create a class to randomize all ALU inputs
        - Constraint the reset to be low most of the time
        - Use the typedef enum to declare the opcode variable of the class
- Create another file that has the testbench module
    - Import the above package
    - Use the typedef enum for the opcode variable
    - Your testbench will use constrained randomization to drive the stimulus inside of a repeat block
    - Make the testbench self-checking using a check_result task
    - Monitor the output to display errors if occurred
    - Use a do file to run the testbench
    - Generate a code coverage report (100% design code coverage is expected. Less than 100% must be justified.)

3) Verify the functionality of the following Moore FSM that detects “010” non-overlapped pattern.

Ports:

Name	Type	Size	Description
x	Input	1 bit	Input sequence
clk			Clock
rst			Active high asynchronous reset
y	Output	1 bit	Output that is HIGH when the sequence 010 is detected
count		10 bits	Outputs the number of time the pattern was detected



Requirements:

1. Create a verification plan
2. Create a package with a typedef enum for the states named state_e
3. Create a class inside of the pacakge named fsm_transaction
    1. Variables
        - x, rst, y_exp (1 bit)
        - user_count_exp (10 bits)
    2. Constraint the reset to be deactivated most of the time
    3. Constraint the x to have value ‘0’ 67% of the time
2. In your testbench, import the package and randomize the object created from the above class, use any of the methods below to self-check the output

a. Method 1 of self checking is creating a golden model (Verilog module) and instantiate it to make your testbench self-checking
b. Method 2 of self checking is to send the object to a task named check_result
    i. Inside of the check_result, you will send the object to another task named golden_model to evaluate the values of the y_exp and user_count_exp of the object. Golden_model task should have inside of it 2 variables declared as cs and ns of datatype state_e. Those will be used to model the FSM.
    ii. After returning from the golden_model task, the check_result task will compare the values of the y_exp and user_counts_exp of the object with the y and user_counts ports of the DUT.

3. Generate a code coverage report and make sure that the **statements, branch, toggle and FSM coverage** are 100%

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_exta for example Kareem_Waseem_Assignment2_extra