SV Randomization & Functional Coverage – Extra

Q1. 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 the input is high then increment counter, else decrement the counter)
    5. ce (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 signal is asserted)
- **Outputs**:
    1. count_out (counter output)
    2. max_count (When the counter reaches the maximum value, this signal is high, else low)
    3. zero (When the counter reaches the minimum value, this signal is high, else low)

Requirements:

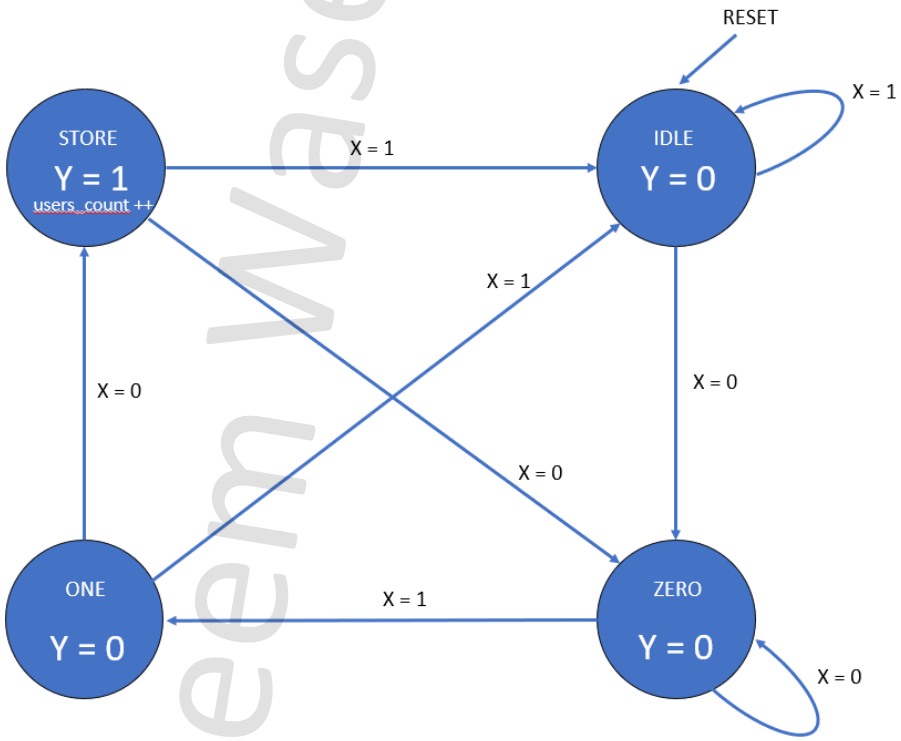
1. List your verification plan items.
2. Create a verification requirement document based on your verification plan items to support your verification planning, an example of the document can be found in the link here. Please copy this document to have your own version and fill the document with the design requirements.
3. Constrained Randomization
    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
4. Functional Coverage
    a. Coverpoint for load data when load is asserted
    b. Coverpoint for count out if the reset is deasserted, enable is active and up_down is high
        i. Autogenerate bins for all values
    c. Coverpoint for count out if the reset is deasserted, enable is active and up_down is high
        i. Transition bin to check when overflow occurs (maximum value to zero)
    d. Coverpoint for count out if the reset is deasserted, enable is active and up_down is low
        i. Autogenerate bins for all values
    e. Coverpoint for count out if the reset is deasserted, enable is active and up_down is low
        i. Transition bin to check when underflow occurs (zero to maximum value)

You are free to add more constraints or coverpoints to enrich your verification to reach 100% code coverage and functional coverage.

Q2. 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 package with a typedef enum for the states named state_e
2. 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
    a. After randomization, send the object to a task named check_result
    b. 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.
    c. 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%