MARIAM MOHAMED MARZOUK

ASSIGNMENT_2_Extended

Question 1

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

I. RTL Design

```
module my_array;

    // Declare a 2-state array holding four 12-bit values
    logic [11:0] my_array[0:3];

    initial begin
        // Initialize the array with specified values
        my_array[0] = 12'h012;  // 0000 0001 0010
        my_array[1] = 12'h345;  // 0011 0100 0101
        my_array[2] = 12'h678;  // 0110 0111 1000
        my_array[3] = 12'h9AB;  // 1001 1010 1011

       // Traverse my_array using a for loop and print bits [5:4]
        $display("Using for loop:");
        for (int i = 0; i < 4; i++) begin
            $display("my_array[
][5:4] =e%b", i, my_array[i][5:4]);

        // Traverse my_array using a foreach loop and print bits [5:4]
        $display("Using foreach loop:");
        foreach (my_array[i]) begin
            $display("my_array[
][5:4] =e%b", i, my_array[i][5:4]);
    end

endmodule
```

II. Print

```
# Using for loop:
# my_array[0][5:4] = 01
# my_array[1][5:4] = 00
# my_array[2][5:4] = 11
# my_array[3][5:4] = 10
# Using foreach loop:
# my_array[0][5:4] = 01
# my_array[1][5:4] = 00
# my_array[2][5:4] = 11
# my_array[3][5:4] = 10
```

Question 2

2) We will test 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.)

I. RTL Design

```
module ALU (
    input  clk,
    input  reset,
    input  [1:0] Opcode,   // The opcode
    input  signed [3:0] A, // Input data A in 2's complement
    input  signed [3:0] B, // Input data B in 2's complement
    output reg signed [4:0] C // ALU output in 2's complement
        );

   reg signed [4:0]      Alu_out; // ALU output in 2's complement

   localparam         Add             = 2'b00; // A + B
   localparam              Sub              = 2'b01; // A - B
   localparam         Not_A      = 2'b10; // ~A
   localparam         ReductionOR_B  = 2'b11; // |B
   // Do the operation
   always @* begin
      case (Opcode)
         Add:            Alu_out = A + B;
         Sub:            Alu_out = A - B;
         Not_A:          Alu_out =~A;
         ReductionOR_B:  Alu_out = |B;
        default:  Alu_out = 5'b0;
      endcase
   end // always @ *
   // Register output C
   always @(posedge clk or posedge reset) begin
      if (reset)
        C <= 5'b0;
      else
        C<= Alu_out;
   end
endmodule

```

II. Package

```
package pck;

typedef enum logic [1:0] {Add , Sub ,Not_A , ReductionOR_B} opcode_e;
class M;
rand bit signed [3: 0] a , b;
rand opcode_e opcode;
rand bit rst;
constraint c_rst {rst dist{0 :/ 90, 1 :/ 10};}
constraint c_input {
        a dist {[-8:-1] :/ 40, [0:7] :/ 60};
        b dist {[-8:-1] :/ 40, [0:7] :/ 60}; }

endclass

endpackage
```

III. Test Bench

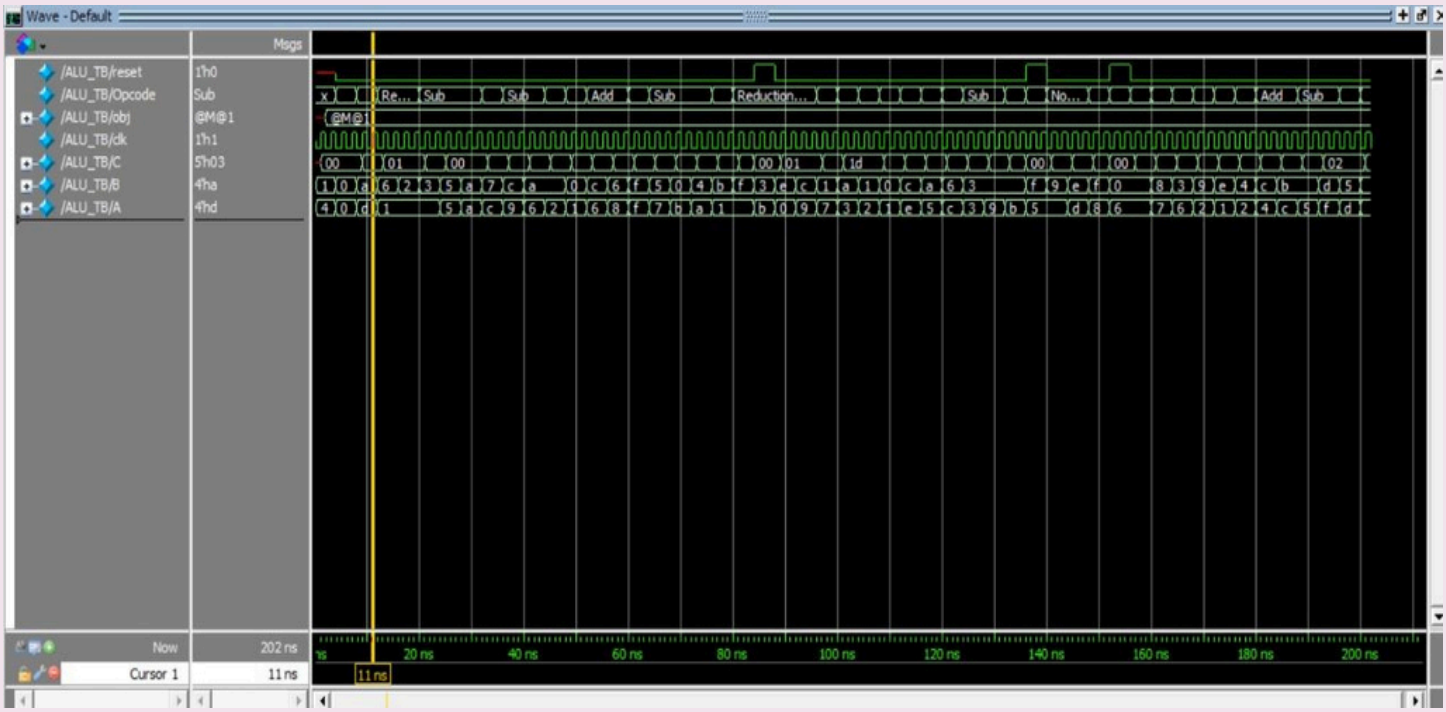
```
import pck::*;
module ALU_TB;
// input and output declaration
logic clk, reset;
opcode_e Opcode;
logic signed [3: 0] A , B;
logic signed [4: 0] C;
// creation of object from class
M obj;
//Module Instantiation
ALU DUT(.*);
// clock declaration
initial begin
clk = 0;
forever
#1 clk = ~clk; // clock period = 2 ns
end
initial begin
// Check the Default Operation Where there is No Opcode & enum of time Logic x
A = $random;
B = $random;
Check_result();
obj = new();
repeat (50) begin
@(negedge clk);
assert(obj. randomize());
A = obj.a;
B = obj.b;
Opcode = obj.opcode;
reset = obj.rst;
Check_result();
$display("%0t: For Inputs A = %0d, B = %0d,Opcode = %0b,rst = %0b,Output C = %0d",$time, A, B,Opcode, reset, C);
end
$stop;
end
task Check_result();
logic signed [4: 0] C_expected;
```

```
    @(negedge clk);
    if(reset)
    C_expected = 0 ;
  else begin
  case(Opcode)
    Add: C_expected = A + B;
    Sub: C_expected = A - B;
    Not_A: C_expected = ~A;
    ReductionOR_B: C_expected = |B;
    default: C_expected = 0 ;
    endcase
    end
  if(C !== C_expected) begin
    $display("ALU Design is Not Working");
    $stop;
    end
    endtask
    endmodule
```

IV. Do file

```
vlib work
vlog ALU.v pck.svh ALU_TB.svh  +cover -covercells
vsim -voptargs=+acc work.ALU_TB -cover
add wave *
coverage save ALU_TB.ucdb -onexit -du work.ALU
run -all
```

V. Waveform



VI. Print

```
# 6: For Inputs A = 0, B = 0,Opcode = 11,rst = 0,Output C = 0
# 10: For Inputs A = -3, B = -6,Opcode = 1,rst = 0,Output C = 3
# 14: For Inputs A = 1, B = 6,Opcode = 11,rst = 0,Output C = 1
# 18: For Inputs A = 1, B = 2,Opcode = 11,rst = 0,Output C = 1
# 22: For Inputs A = 1, B = 3,Opcode = 1,rst = 0,Output C = -2
# 26: For Inputs A = 5, B = 5,Opcode = 1,rst = 0,Output C = 0
# 30: For Inputs A = -6, B = -6,Opcode = 1,rst = 0,Output C = 0
# 34: For Inputs A = -4, B = 7,Opcode = 10,rst = 0,Output C = 3
# 38: For Inputs A = -7, B = -4,Opcode = 1,rst = 0,Output C = -3
# 42: For Inputs A = 6, B = -6,Opcode = 1,rst = 0,Output C = 12
# 46: For Inputs A = 2, B = -6,Opcode = 0,rst = 0,Output C = -4
# 50: For Inputs A = 1, B = 0,Opcode = 10,rst = 0,Output C = -2
# 54: For Inputs A = 6, B = -4,Opcode = 0,rst = 0,Output C = 2
# 58: For Inputs A = -8, B = 6,Opcode = 0,rst = 0,Output C = -2
# 62: For Inputs A = -1, B = -1,Opcode = 11,rst = 0,Output C = 1
# 66: For Inputs A = 7, B = 5,Opcode = 1,rst = 0,Output C = 2
# 70: For Inputs A = -5, B = 0,Opcode = 1,rst = 0,Output C = -5
# 74: For Inputs A = -6, B = 4,Opcode = 1,rst = 0,Output C = -10
# 78: For Inputs A = 1, B = -5,Opcode = 0,rst = 0,Output C = -4
# 82: For Inputs A = 1, B = -1,Opcode = 11,rst = 0,Output C = 1
# 86: For Inputs A = -5, B = 3,Opcode = 11,rst = 1,Output C = 0
# 90: For Inputs A = 0, B = -2,Opcode = 11,rst = 0,Output C = 1
# 94: For Inputs A = -7, B = -4,Opcode = 11,rst = 0,Output C = 1
# 98: For Inputs A = 7, B = 1,Opcode = 1,rst = 0,Output C = 6
# 102: For Inputs A = 3, B = -6,Opcode = 0,rst = 0,Output C = -3
# 106: For Inputs A = 2, B = 1,Opcode = 10,rst = 0,Output C = -3
# 110: For Inputs A = 1, B = 0,Opcode = 0,rst = 0,Output C = 1
# 114: For Inputs A = -2, B = -4,Opcode = 1,rst = 0,Output C = 2
# 118: For Inputs A = 5, B = -6,Opcode = 10,rst = 0,Output C = -6
# 122: For Inputs A = -4, B = 6,Opcode = 0,rst = 0,Output C = 2
# 126: For Inputs A = 3, B = 3,Opcode = 1,rst = 0,Output C = 0
# 130: For Inputs A = -7, B = 3,Opcode = 1,rst = 0,Output C = -10
# 134: For Inputs A = -5, B = 3,Opcode = 10,rst = 0,Output C = 4
# 138: For Inputs A = 5, B = -1,Opcode = 0,rst = 1,Output C = 0
# 142: For Inputs A = 5, B = -7,Opcode = 10,rst = 0,Output C = -6
# 146: For Inputs A = -3, B = -2,Opcode = 10,rst = 0,Output C = 2
# 150: For Inputs A = -8, B = -1,Opcode = 1,rst = 0,Output C = -7
# 154: For Inputs A = 6, B = 0,Opcode = 10,rst = 1,Output C = 0
# 158: For Inputs A = 6, B = 0,Opcode = 1,rst = 0,Output C = 6
# 162: For Inputs A = 7, B = -8,Opcode = 0,rst = 0,Output C = -1
# 166: For Inputs A = 6, B = 3,Opcode = 1,rst = 0,Output C = 3
# 170: For Inputs A = 2, B = -7,Opcode = 0,rst = 0,Output C = -5
# 174: For Inputs A = 1, B = -2,Opcode = 11,rst = 0,Output C = 1
# 178: For Inputs A = 2, B = 4,Opcode = 1,rst = 0,Output C = -2
# 182: For Inputs A = 4, B = -4,Opcode = 0,rst = 0,Output C = 0
# 186: For Inputs A = -4, B = -5,Opcode = 0,rst = 0,Output C = -9
```

```
190: For Inputs A = 5, B = -5,Opcode = 1,rst = 0,Output C = 10
194: For Inputs A = -1, B = -3,Opcode = 1,rst = 0,Output C = 2
198: For Inputs A = -3, B = 5,Opcode = 0,rst = 0,Output C = 2
202: For Inputs A = -4, B = 0,Opcode = 1,rst = 0,Output C = -4
```

VII. Coverage Report

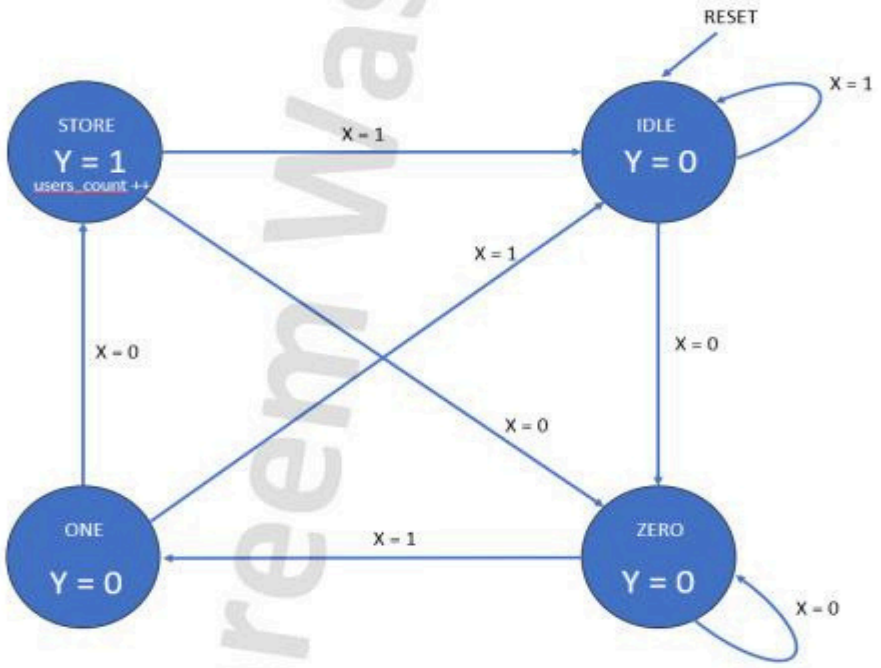
```
Branch Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Branches                        7         7         0   100.00%

================================Branch Details================================

Branch Coverage for instance /\ALU_TB#DUT

    Line         Item                    Count     Source
    ----         ----                    -----     ------
  File ALU.v
-----------------------------------CASE Branch-----------------------------------
    18                                      51     Count coming in to CASE
    19              1                       10
    20              1                        8
    21              1                       14
    22              1                       18
    23              1                        1
Branch totals: 5 hits of 5 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    28                                      91     Count coming in to IF
    28              1                        6
    30              1                       85
Branch totals: 2 hits of 2 branches = 100.00%


Statement Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Statements                      9         9         0   100.00%

================================Statement Details================================

Statement Coverage for instance /\ALU_TB#DUT  --

    Line         Item                    Count     Source
    ----         ----                    -----     ------
  File ALU.v
    17              1                       51
    19              1                       10
    20              1                        8
    21              1                       14
    22              1                       18
    23              1                        1
    27              1                       91
    29              1                        6
    31              1                       85
Toggle Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Toggles                        44        44         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /\ALU_TB#DUT  --

                                       Node      1H->0L      0L->1H                          "Coverage"
                                       ----------------------------------------------------------------
                                     A[0-3]           1           1                              100.00
                                Alu_out[4-0]          1           1                              100.00
                                     B[0-3]           1           1                              100.00
                                     C[4-0]           1           1                              100.00
                                Opcode[0-1]           1           1                              100.00
                                        clk           1           1                              100.00
                                      reset           1           1                              100.00

Total Node Count     =         22
Toggled Node Count   =         22
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (44 of 44 bins)
Total Coverage By Instance (filtered view): 100.00%
```

Question 3

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 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, 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

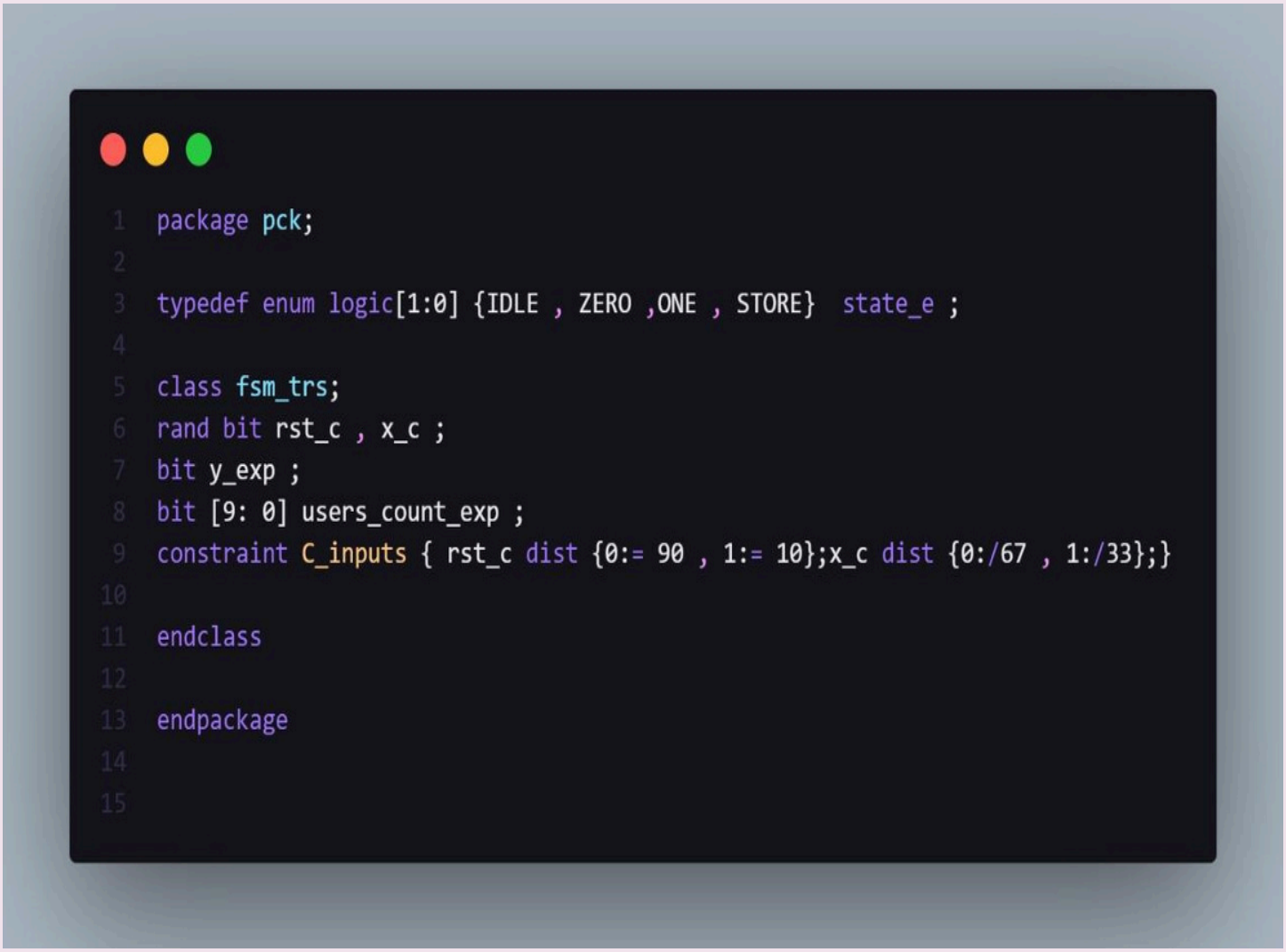
I. RTL Design

```
module FSM(clk, rst, x, y, users_count);
    parameter IDLE  = 2'b00;
    parameter ZERO  = 2'b01;
    parameter ONE   = 2'b10;
    parameter STORE = 2'b11;
    input clk, rst, x;
    output y;
    output reg [9:0] users_count;
    reg [1:0] cs, ns;
    always @(*) begin
        case (cs)
            IDLE:
                if (x)
                    ns = IDLE;
                else
                    ns = ZERO;
            ZERO:
                if (x)
                    ns = ONE;
                else
                    ns = ZERO;
            ONE:
                if (x)
                    ns = IDLE;
                else
                    ns = STORE;
            STORE:
                if (x)
                    ns = IDLE;
                else
                    ns = ZERO;
            default: ns = IDLE;
        endcase
    end
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            cs <= IDLE;
            users_count <= 10'b0;
        end else begin
            cs <= ns;
        end
    end
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            users_count <= 10'b0;
        end else begin
            if (cs == STORE)
                users_count <= users_count + 1;
        end
    end
    assign y = (cs == STORE) ? 1 : 0;
endmodule
```

II. Golden Model

```
module FSM_GM(x, clk, rst, y_exp, count_exp);
    parameter STORE = 2'b00;
    parameter IDLE = 2'b01;
    parameter ZERO = 2'b10;
    parameter ONE = 2'b11;
    input x, clk, rst;
    output reg y_exp;
    output reg [9:0] count_exp;
    reg [1:0] cs, ns;
    // Next state Logic
    always @(cs or x) begin
        case (cs)
            STORE :
                if (x)
                    ns = IDLE;
                else
                    ns = ZERO;
            IDLE :
                if (x)
                    ns = IDLE;
                else
                    ns = ZERO;
            ZERO :
                if (x)
                    ns = ONE;
                else
                    ns = ZERO;
            ONE :
                if (x)
                    ns = IDLE;
                else
                    ns = STORE;
            default : ns = IDLE;
        endcase
    end
    // State Memory
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            cs <= IDLE;
            count_exp <= 10'b0;
        end else
            cs <= ns;
    end
    always @(cs) begin
        case (cs)
            STORE : begin
                y_exp = 1;
                count_exp = count_exp + 1;
            end
            IDLE, ZERO, ONE : y_exp = 0;
        endcase
    end
endmodule
```

III. Package



```
1  package pck;
2
3  typedef enum logic[1:0] {IDLE , ZERO ,ONE , STORE}  state_e ;
4
5  class fsm_trs;
6  rand bit rst_c , x_c ;
7  bit y_exp ;
8  bit [9: 0] users_count_exp ;
9  constraint C_inputs { rst_c dist {0:= 90 , 1:= 10};x_c dist {0:/67 , 1:/33};}
10
11 endclass
12
13 endpackage
14
15
```

IV. Test Bench

```
import pck::*;
module FSM_TB;
// Parameters
parameter IDLE = 2'b00;
parameter ZERO = 2'b01;
parameter ONE = 2'b10;
parameter STORE = 2'b11;
// Signal declarations
logic clk, rst, x;
logic y, y_expected;
logic [9:0] users_count, users_count_expected;
int error_counter, correct_counter;
// FSM transaction object
fsm_trs obj; // create object from the class
// DUT instantiation
FSM #(.IDLE(IDLE), .ZERO(ZERO), .ONE(ONE), .STORE(STORE))
DUT_Design (clk, rst, x, y, users_count);
FSM_GM #(.IDLE(IDLE), .ZERO(ZERO), .ONE(ONE), .STORE(STORE))
DUT_GOLD (x, clk, rst, y_expected, users_count_expected);
// Clock generation
initial begin
    clk = 0;
    forever #1 clk = ~clk;
end
// Testbench logic
initial begin
    error_counter = 0;
    correct_counter = 0;
    obj = new();
    // Reset sequence
    rst = 1;
    #10;
    rst = 0;
    #10; // Allow DUT to initialize
    // Generate random inputs and apply them
    repeat (20000) begin
        assert(obj.randomize());  // Randomize inputs
          rst = obj.rst_c;
          x = obj.x_c;
        obj.users_count_exp = users_count_expected;
        obj.y_exp = y_expected;
        check_result();          // Check results
    end
    // Directed test to cover specific scenarios
    rst = 1;
    #10;
    rst = 0;
   repeat (10000) begin
    // IDLE (00) State transitions
    x = 0; #10;  // IDLE (00) -> ZERO (01)
    x = 1; #10;  // IDLE (00) -> IDLE (00)
    // ZERO (01) State transitions
    x = 0; #10;  // ZERO (01) -> ZERO (01)
    x = 1; #10;  // ZERO (01) -> ONE (10)
    // ONE (10) State transitions
    x = 0; #10;  // ONE (10) -> STORE (11)
    x = 1; #10;  // ONE (10) -> ONE (10)
    // STORE (11) State transitions
    x = 0; #10;  // STORE (11) -> ZERO (01)
    x = 1; #10;  // STORE (11) -> IDLE (00)
end
    $display("Error_Counter =
 , C@stopt_Counter =
@gderror_counter, correct_counter);
    // Check Result Task
    task check_result();
        @(negedge clk);
       if ((y !== y_expected) && (users_count !== users_count_expected)) begin
    error_counter++;
end
else
correct_counter++;
    endtask
endmodule
```

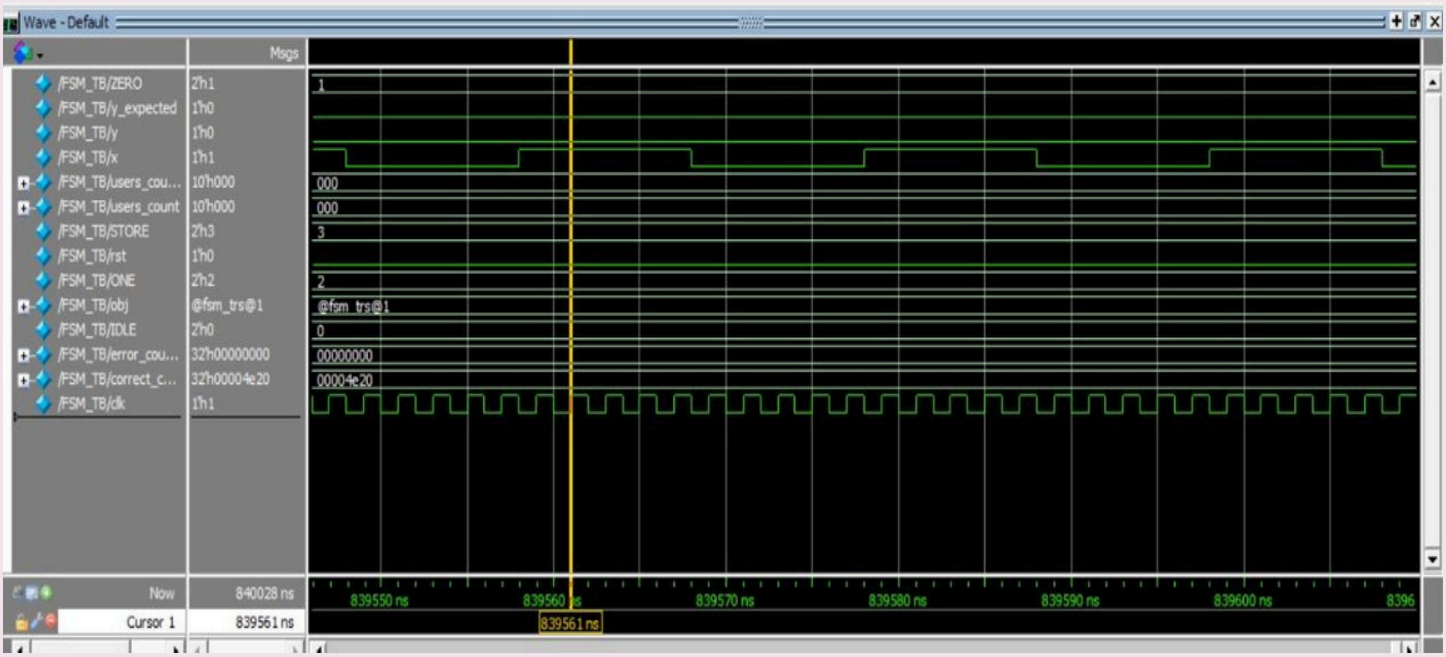
V. Counters

```
# Error_Counter = 0 , Correct_Counter = 20000
```

VI. Do file

```
vlib work
vlog FSM.v FSM_TB.sv FSM_GM.v pck.sv  +cover -covercells
vsim -voptargs=+acc work.FSM_TB -cover
add wave *
coverage save FSM_TB.ucdb -onexit -du work.FSM
run -all
```

VII. Waveform



VIII. Coverage Report

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        20        20         0   100.00%

================================Branch Details================================

Branch Coverage for instance /\FSM_TB#DUT_Design

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File FSM.v
-----------------------------------CASE Branch-----------------------------------
    21                                    220652     Count coming in to CASE
    22              1                      85752
    27              1                      87412
    32              1                      45051
    37              1                       2437
Branch totals: 4 hits of 4 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    23                                     85752     Count coming in to IF
    23              1                      42349
    25              1                      43403
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    28                                     87412     Count coming in to IF
    28              1                      43345
    30              1                      44067
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    33                                     45051     Count coming in to IF
    33              1                      43017
    35              1                       2034
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    38                                      2437     Count coming in to IF
    38              1                        599
    40              1                       1838
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    47                                    217566     Count coming in to IF
    47              1                       3767
    50              1                     213799
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    56                                    134827     Count coming in to IF
    56              1                       3688
    58              1                     131139
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    59                                    131139     Count coming in to IF
```

```
    59             1                       1645
                                         129494     All False Count
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch------------------------------------
    64                                   131886     Count coming in to IF
    64             1                       1838
    64             2                     130048
Branch totals: 2 hits of 2 branches = 100.00%


Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                       2         2         0   100.00%

================================Condition Details================================

Condition Coverage for instance /\FSM_TB#DUT_Design  --

  File FSM.v
----------------Focused Condition View-------------------
Line       59 Item    1  (cs == 3)
Condition totals: 1 of 1 input term covered = 100.00%

  Input Term   Covered  Reason for no coverage   Hint
 -----------  --------  -----------------------  --------------
   (cs == 3)         Y

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (cs == 3)_0           -
  Row   2:          1  (cs == 3)_1           -

----------------Focused Condition View-------------------
Line       64 Item    1  (cs == 3)
Condition totals: 1 of 1 input term covered = 100.00%

  Input Term   Covered  Reason for no coverage   Hint
 -----------  --------  -----------------------  --------------
   (cs == 3)         Y

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (cs == 3)_0           -
  Row   2:          1  (cs == 3)_1           -


FSM Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    FSM States                       4         4         0   100.00%
    FSM Transitions                  7         7         0   100.00%

================================FSM Details================================

FSM Coverage for instance /\FSM_TB#DUT_Design  --

FSM_ID: cs
    Current State Object : cs
```

```
   ----------------------
   State Value MapInfo :
   --------------------
ine          State Name              Value
---          ----------              -----
 22                IDLE                  0
 27                ZERO                  1
 32                 ONE                  2
 37               STORE                  3
   Covered States :
   ---------------
                  State          Hit_count
                  -----          ---------
                   IDLE              86166
                   ZERO              86545
                    ONE              43017
                  STORE               1838
   Covered Transitions :
   --------------------
ine            Trans_ID          Hit_count         Transition
---            --------          ---------         ----------
 26                   0              42964         IDLE -> ZERO
 29                   1              43017         ZERO -> ONE
 48                   2               1049         ZERO -> IDLE
 36                   3               1838         ONE -> STORE
 34                   4              41179         ONE -> IDLE
 41                   5               1102         STORE -> ZERO
 39                   6                736         STORE -> IDLE


   Summary                       Bins      Hits    Misses  Coverage
   -------                       ----      ----    ------  --------
       FSM States                   4         4         0   100.00%
       FSM Transitions              7         7         0   100.00%
tatement Coverage:
   Enabled Coverage              Bins      Hits    Misses  Coverage
   ----------------              ----      ----    ------  --------
   Statements                    17        17         0   100.00%

================================Statement Details================================

tatement Coverage for instance /\FSM_TB#DUT_Design  --

   Line         Item                      Count     Source
   ----         ----                      -----     ------
 File FSM.v
   20              1                     220652
   24              1                      42349
   26              1                      43403
   29              1                      43345
   31              1                      44067
   34              1                      43017
   36              1                       2034
   39              1                        599
   41              1                       1838
   46              1                     217566
   48              1                       3767
   49              1                       3767
   51              1                     213799
   55              1                     134827
```

```
    51              1                    213799
    55              1                    134827
    57              1                      3688
    60              1                      1645
    64              1                    131887
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         24        24         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /\FSM_TB#DUT_Design  --

                                          Node      1H->0L      0L->1H  "Coverage"
                                          ----------------------------------------
                                           clk           1           1      100.00
                                        cs[1-0]          1           1      100.00
                                        ns[1-0]          1           1      100.00
                                           rst           1           1      100.00
                                  users_count[3-0]       1           1      100.00
                                             x           1           1      100.00
                                             y           1           1      100.00

Total Node Count     =         12
Toggled Node Count   =         12
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (24 of 24 bins)
Total Coverage By Instance (filtered view): 100.00%
```

IX. Comment

Since the total number of possibilities for the users_count counter is \(2^{10} = 1024\), achieving 100% toggle coverage across all states is not feasible. However, by excluding bits [4-9], the number of states to be covered is reduced, which allows us to achieve 100% toggle coverage for the remaining bits.