Assignment extra 2

Q1

CODE :

array_extra.sv

```
1   module array_2 ();
2
3       bit [12] my_array [4];
4       int j = 0;
5
6       initial begin
7           my_array [0] = 12'h012;
8           my_array [1] = 12'h345;
9           my_array [2] = 12'h678;
10          my_array [3] = 12'h9ab;
11          $display(" my_arrar = %p", my_array);
12
13          for (int k= 0; k<4 ; k++) begin
14           for (int i=4 ; i<6 ; i++) begin
15              $display(" bits [4:5] = %b", my_array [j][i]);
16          end
17          j++;
18          end
19
20      end
21  endmodule
```

DISPLAY :

```
VSIM 6> run -all
#  my_arrar = '{18, 837, 1656, 2475}
#  bits [4:5] = 0
#  bits [4:5] = 0
#  bits [4:5] = 0
#  bits [4:5] = 1
#  bits [4:5] = 0
#  bits [4:5] = 1
#  bits [4:5] = 1
#  bits [4:5] = 0
```

Q2

CODE PKG :

assignment2_extra > alu_pkg.sv

```
1   package alu_pkg;
2
3       typedef enum bit [1:0] { Add,Sub,Not_A,ReductionOR_B } opcode_e;
4
5       class ALU_class ;
6
7       rand bit  reset;
8       rand opcode_e opcode;
9       rand reg signed [3:0] A;
10      rand reg signed [3:0] B;
11
12      constraint ALU_reset {
13          reset dist { 0:=99 , 1:=1};
14      }
15          endclass
16
17  endpackage
```

CODE TB :

assignment2_extra > ALU_tb.sv

```
1   import alu_pkg ::*;
2   module ALU_tb ;
3       reg  clk;
4       reg  reset;
5       opcode_e opcode;    // The opcode
6       reg  signed [3:0] A;    // Input data A in 2's complement
7       reg  signed [3:0] B;    // Input data B in 2's complement
8       wire signed [4:0] C; // ALU output in 2's complement
9
10      wire signed [4:0] C_ex;
11
12      ALU_4_bit t (clk,reset,opcode,A,B,C);
13      ALU_4_bit k (clk,reset,opcode,A,B,C_ex);
14
15      ALU_class my_ALU_class = new();
16
17      int error_count = 0;
18      int correct_count =0;
19
20      initial begin
21          clk = 0;
22          forever #1 clk = ~clk;
23      end
24
25      task cheack_result ;
26          @(negedge clk);
27          if (C != C_ex) begin
28              $display ("error");
29              error_count++;
30          end
31          else begin
32              $display ("correct");
33              correct_count++;
34          end
35          endtask
36
```

```
37      task cheack_reset ;
38          reset = 0;
39          @(negedge clk);
40          cheack_result ;
41          reset = 1;
42
43      endtask
44
45      initial begin
46          //1
47          cheack_reset;
48
49          //2
50          repeat (1000) begin
51              assert (my_ALU_class.randomize());
52              reset = my_ALU_class.reset;
53              A = my_ALU_class.A;
54              B = my_ALU_class.B;
55              opcode = my_ALU_class.opcode;
56              cheack_result;
57          end
58
59          //3
60          cheack_reset;
61
62          //4
63          repeat (90000) begin
64              assert (my_ALU_class.randomize());
65              cheack_result ;
66          end
67
68          //5
69          cheack_reset;
70
71          $display("error = %d correct = %d", error_count, correct_count);
72          $stop;
73      end
74
```

DO FILE :

assignment2_extra > ALUexstra.do

```
1   vlib work
2   vlog ALU.v ALU_tb.sv  alu_pkg.sv +cover -covercells
3   vsim -voptargs=+acc work.ALU_tb -cover
4   add wave *
5   coverage save ALU_tb.ucdb -onexit
6   run -all
```

Chat (CTRL + I) / Edit (CTRL + L)

DISPLAY :

```
# correct
# error =           0 correct =       91003
# ** Note: $stop    : ALU_tb.sv(72)
#    Time: 182012 ns  Iteration: 1  Instance: /ALU_tb
# Break in Module ALU_tb at ALU_tb.sv line 72
```

COVERAGE :

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         44        44         0   100.00%

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

Toggle Coverage for instance /ALU_tb/t --

                                          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)
```

VERIFICATION PLAN :

	label	description	stimulus generation	function functionality cheack
2	ALU_1	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is corre
3	ALU_2	verifying randomize and constraint input	directed at the start of the simulation	cheacker in the testbench to make sure the output is corre
4	ALU_3	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is corre
5	ALU_4	verifying randomize and constraint input	directed at the start of the simulation	cheacker in the testbench to make sure the output is corre
6	ALU_5	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is corre

Q3
CODE TB :

```
import fsm_pkg ::*;
module fsm_tb ();
    logic clk, rst, x;
    logic y;
    logic [9:0] users_count;

    fsm_transaction my_fsm1 = new;

    int error_count = 0;
    int correct_count =0;

    state_e cs = IDLE;
    state_e ns;

     FSM_010 l (.*);

   initial begin
    clk = 0;
    forever #1 clk = ~clk;
   end

   task cheack_result (input fsm_transaction my_fsm2);
    golden_model (my_fsm2);
    if (my_fsm2.rst !== 1) begin
    @(negedge clk);
    if ( (my_fsm2.y_exp !== y) || (my_fsm2.users_count_exp !== users_count) )begin
        $display ("error in out");
        error_count++;
    end
    else begin
        //$display ("correct");
        correct_count++;
    end
    end
    else begin
    cheack_result_rst;
    end
    endtask

    task golden_model (input fsm_transaction my_fsm3) ;
        case (cs)
            IDLE: begin
                if (my_fsm3.x)
                    ns = IDLE;
                else
                    ns = ZERO;
            end
```

```
47              end
48              ZERO: begin
49                  if (my_fsm3.x)
50                      ns = ONE;
51                  else
52                      ns = ZERO;
53              end
54              ONE: begin
55                  if (my_fsm3.x)
56                      ns = IDLE;
57                  else
58                      ns = STORE;
59              end
60              STORE: begin
61                  if (my_fsm3.x)
62                      ns = IDLE;
63                  else
64                      ns = ZERO;
65              end
66          endcase
67          @(posedge clk) ;
68          if(my_fsm3.rst) begin
69              cs <= IDLE;
70              my_fsm3.users_count_exp <= 0;
71          end
72          else begin
73              if (cs == STORE)
74          my_fsm3.users_count_exp++;
75          cs = ns;
76          end
77           my_fsm3.y_exp = (cs == STORE)? 1:0;
78
79      endtask
80
81      task cheack_reset ;
82          rst = 1;
83          cheack_result_rst;
84          rst = 0;
85
86      endtask
87
88      task cheack_result_rst ;
89          @(negedge clk);
```

```
87
88      task cheack_result_rst ;
89          @(negedge clk);
90          if ((y !== 0) && (users_count !== 0)) begin
91          $display ("error in rst");
92          error_count++;
93      end
94      else begin
95        //  $display ("correct");
96          correct_count++;
97      end
98
99      endtask
100
101     initial begin
102         x = 0;
103
104         cheack_reset;
105
106         repeat (9000) begin
107             assert (my_fsm1.randomize());
108             rst = my_fsm1.rst;
109             x = my_fsm1.x;
110             cheack_result(my_fsm1);
111         end
112
113         repeat (9000) begin
114             assert (my_fsm1.randomize());
115             x = my_fsm1.x;
116             rst = 0;
117             my_fsm1.rst=0;
118             cheack_result(my_fsm1);
119         end
120         $display("error = %d correct = %d", error_count, correct_count);
121         $stop;
122     end
123
124 endmodule
```

CODE PKG :

assignment2_extra > fsm_pkg.sv

```
1   package fsm_pkg;
2
3       typedef enum { STORE,IDLE,ZERO,ONE } state_e;
4
5       class fsm_transaction ;
6           rand bit x,rst;
7           bit y_exp ;
8           bit [9:0] users_count_exp;
9
10          constraint fsm_reset {
11              rst dist { 0:=99 , 1:=1};
12          }
13          constraint x_reset {
14              x dist { 0:=67 , 1:=33};
15          }
16      endclass
17
18  endpackage
19
```

DO FILE :

assignment2_extra > fsm.do

```
1   vlib work
2   vlog FSM_010.v fsm_pkg.sv  fsm_tb.sv +cover -covercells
3   vsim -voptargs=+acc work.fsm_tb -cover
4   add wave *
5   coverage save fsm_tb.ucdb -onexit
6   run -all
```

DISPLAY :

```
# Loading work.FSM_010(fast)
# error =           0 correct =       18001
# ** Note: $stop    : fsm_tb.sv(121)
#    Time: 36002 ns  Iteration: 1  Instance: /fsm_tb
# Break in Module fsm_tb at fsm_tb.sv line 121
```

COVERAGE :

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         36        36         0   100.00%
```

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        21        21         0   100.00%
```

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

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

```
        FSM Transitions              7         7         0   100.00%
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      17        17         0   100.00%
```

VERIFICATION PLAN :

7	FSM_1	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is correc
8	FSM_2	verifying randomize and constraint input	directed at the start of the simulation	cheacker in the testbench to make sure the output is
9				correct