Data Types & Constrained Random

Assignment2

By: Mohamed Sayed Mohamed Soliman

Question 1

- Write a module to test dynamic array data type and its predefined methods.

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

• initialize array dyn_arr1 with index as its value

• display dyn_arr1 and its size

o 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

1. Code of the design

```
module test_dynamic_array ;

int  dyn_arr1 [] ;
int  dyn_arr2 [] ;

initial begin
    dyn_arr1 = '{9,1,8,3,4,4} ;
    dyn_arr2 = new[6] ;

    foreach(dyn_arr2[j])
    begin
        dyn_arr2[j] = j ;
    end

    foreach(dyn_arr2[j])
    begin
        $display("dyn_arr2[%0d] = %0d" , j , dyn_arr2[j] );
    end
    $display("dyn_arr2 = %p"  , dyn_arr2 );

    dyn_arr1.reverse ;
    $display("reverse of dyn_arr1 = %p"  , dyn_arr1 );

    dyn_arr1.sort ;
    $display("sort of dyn_arr1 = %p"  , dyn_arr1 );

    dyn_arr1.rsort ;
    $display("reverse sort of dyn_arr1 = %p"  , dyn_arr1 );

    dyn_arr1.shuffle ;
    $display("shuffle of dyn_arr1 = %p"  , dyn_arr1 );

end

endmodule
```

2. Result of the simulation

```
# dyn_arr2[0] = 0
# dyn_arr2[1] = 1
# dyn_arr2[2] = 2
# dyn_arr2[3] = 3
# dyn_arr2[4] = 4
# dyn_arr2[5] = 5
# dyn_arr2 = '{0, 1, 2, 3, 4, 5}
# reverse of dyn_arr1 = '{4, 4, 3, 8, 1, 9}
# sort of dyn_arr1 = '{1, 3, 4, 4, 8, 9}
# reverse sort of dyn_arr1 = '{9, 8, 4, 4, 3, 1}
# shuffle of dyn_arr1 = '{8, 4, 9, 1, 3, 4}
```

Question 2

➢ Counter

Parameters:

I. 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:

I. 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

II. Create a package that has a class with the following constraints
   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

III. Create a testbench that randomize the data in a repeat block or for loop using the class object to use the above constraints. Make the testbench self-checking.

1. Code Design

```
8   module counter (clk ,rst_n, load_n, up_down, ce, data_load, count_out, max_count, zero);
9   parameter WIDTH = 4;
10  input clk;
11  input rst_n;
12  input load_n;
13  input up_down;
14  input ce;
15  input [WIDTH-1:0] data_load;
16  output reg [WIDTH-1:0] count_out;
17  output max_count;
18  output zero;
19
20  always @(posedge clk) begin
21      if (!rst_n)
22          count_out <= 0;
23      else if (!load_n)
24          count_out <= data_load;
25      else if (ce)
26          if (up_down)
27              count_out <= count_out + 1;
28          else
29              count_out <= count_out - 1;
30  end
31
32  assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
33  assign zero = (count_out == 0)? 1:0;
34
35  endmodule
```

2. Verification plan

	A	B	C	D	E
1	Label	Description	Stimulus Generation	Functional	Functionality Check
2	COUNTER_1	When the reset is asserted, the output counter value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
3	COUNTER_2	When the load is asserted, the output count_out should take the value of the load_data input	Randomization	-	A checker in the testbench to make sure the output is correct
4	COUNTER_3	When the load is deasserted, ce is asserted and up_down is asserted the output count_out should take the old value +1	Randomization	-	A checker in the testbench to make sure the output is correct
5	COUNTER_4	When the load is deasserted, ce is asserted and up_down is deasserted the output count_out should take the old value -1	Randomization	-	A checker in the testbench to make sure the output is correct
6					

3. Counter Package

```
package pack_count;

    class cla_count ;
      parameter WIDTH = 8;
      rand  logic [WIDTH-1:0] data_load ;
      rand  logic             ce ;
      rand  logic             up_down ;
      rand  logic             load_n ;
      rand  logic             rst_n ;

      constraint enable {ce dist {1:/70 , 0:/30  } ; }
      constraint d {up_down dist {1:/60 , 0:/40  } ; }
      constraint load {load_n dist {1:/30 , 0:/70  } ; }
      constraint rst {rst_n dist {1:/99 , 0:/1  } ; }

        function void print ();
        $display("data_load = 0h%0h  , ce = 0h%0h , up_down = 0h%0h , load_n = 0h%0h , rst_n = 0h%0h  " , this.data_load , this.ce , this.up_down , this.load_n , this.rst
        endfunction


        endclass

endpackage
```

4. Counter Testbench

```
    import pack_count::*;

    module counter_tb();

      parameter WIDTH = 8;
      logic   clk ;
      logic   rst_n ;
      logic   load_n    ;
      logic   up_down    ;
      logic   ce   ;
      logic [WIDTH-1:0] data_load ;
      logic   max_count    ;
      logic   zero   ;
      logic [WIDTH-1:0] count_out ;

      integer i ;
      integer correct_counter = 0 ;
    integer error_counter = 0 ;
    logic [WIDTH:0] old_count_out ;

    counter #(.WIDTH(WIDTH)) dut (
                .clk(clk),
                .rst_n(rst_n),
                .load_n(load_n),
                .up_down(up_down),
                .ce(ce),
                .data_load(data_load),
                .max_count(max_count),
                .zero(zero),
                .count_out(count_out)
          );

          always #20 clk = ~clk ;
```

```
34
35  cla_count trans1 , trans2 ;
36
37  initial begin
38    clk = 0 ;
39    trans1 = new () ;
40    rest ();
41    for(i=0; i<1000 ;i=i+1) begin
42      trans1.randomize();
43      //trans1.print() ;
44      data_load = trans1.data_load ;
45      ce = trans1.ce ;
46      up_down = trans1.up_down ;
47      load_n = trans1.load_n ;
48      rst_n = trans1.rst_n ;
49
50      if (!trans1.rst_n)
51          check_result_rst(0);
52      else if (!trans1.load_n)
53          check_result_load(trans1.data_load);
54      else if(trans1.ce)
55      begin
56        if (trans1.up_down)
57          check_result_ce(old_count_out+1);
58          else
59          check_result_ce(old_count_out-1);
60      end
61      else
62      begin
63        check_result_ce(old_count_out);
64      end
```

```
68
69      $display (" testbench 1 "  );
70      $display ("error_counter = %0d " ,error_counter );
71      $display ("correct_counter = %0d " ,correct_counter );
72
73    $stop ;
74
75  end
76
77
78  task check_result_rst(input [WIDTH:0] expected_result );
79    @(negedge clk );
80    if(expected_result != count_out )
81        begin
82            $display (":error");
83            old_count_out = count_out ;
84            error_counter = error_counter +1 ;
85
86        end
87    else
88        begin
89            correct_counter = correct_counter + 1 ;
90            old_count_out = count_out ;
91        end
92  endtask
93
```

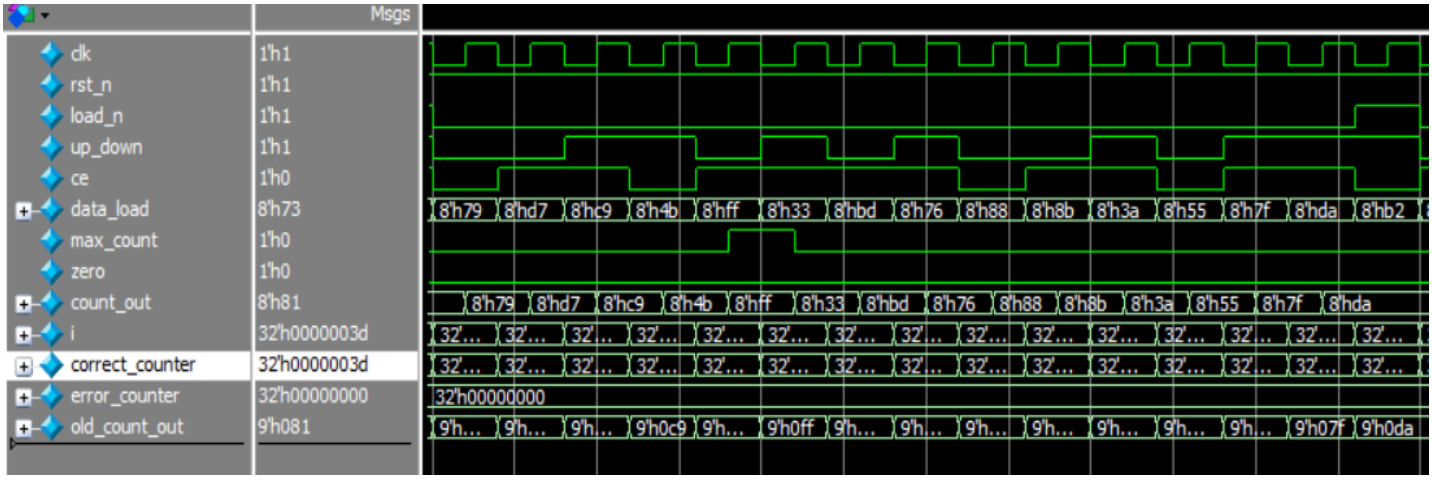
```
94
95          task check_result_load(input [WIDTH:0] expected_result );
96        @(negedge clk );
97        if(expected_result != count_out )
98            begin
99                $display (":error");
100               old_count_out = count_out ;
101               error_counter = error_counter +1 ;
102
103           end
104           else
105           begin
106               correct_counter = correct_counter + 1 ;
107               old_count_out = count_out ;
108           end
109   endtask
110
```

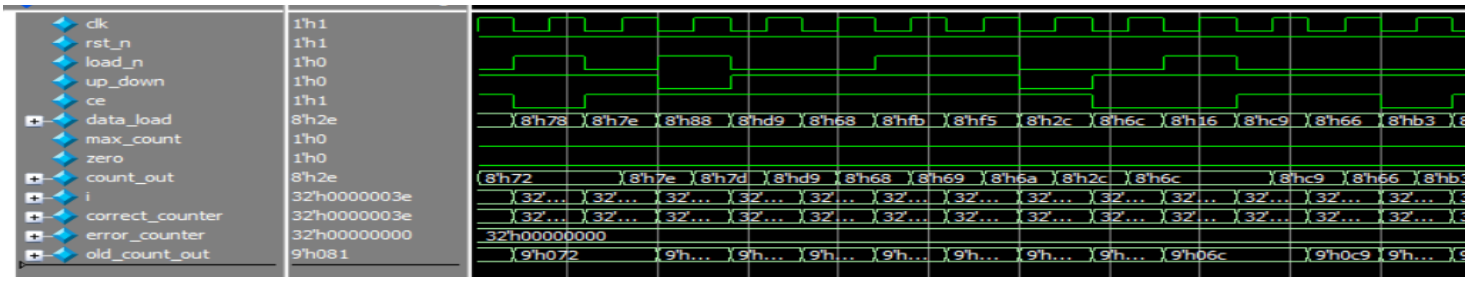
```
                    task check_result_ce(input [WIDTH-1:0] expected_result );
            @(negedge clk );
            if((expected_result)!= count_out )
                begin
                    $display (":error");
                    old_count_out = count_out ;
                    error_counter = error_counter +1 ;
                    $display("data_load = 0h%0h  , max_count = 0h%0h  , zero = 0h%0h , count_out = 0h%0h " , data_load , max_count , zero , count_out ) ;
                end
                else
                begin
                    correct_counter = correct_counter + 1 ;
                    old_count_out = count_out ;
                end
        endtask

  task rest ();
    rst_n = 1 ;
    #20
    rst_n = 0 ;

endtask

endmodule
```

5. Do File

```
1   vlib work
2   vlog counter.v counter_tb.sv +cover -covercells
3   vsim -voptargs=+acc work.counter_tb -cover
4   add wave *
5   coverage save counter_tb.ucdb -onexit
6   run -all
7
```

6. Waveform





7. Transcript

```
# error_counter = 0
# correct_counter = 1000
```

8. Code Coverage

```
================================================================================
=== File: counter.v
================================================================================
Statement Coverage:
    Enabled Coverage           Active      Hits    Misses % Covered
    ----------------           ------      ----    ------ ---------
    Stmts                           7         7         0     100.0

                               Statement Details

Statement Coverage for file counter.v --

    1                                      //////////////////////////////////////////////////////////////////////////////
    2                                      // Author: Kareem Waseem
    3                                      // Course: Digital Verification using SV & UVM
    4                                      //
    5                                      // Description: Counter Design
    6                                      //
    7                                      //////////////////////////////////////////////////////////////////////////////
    8                                      module counter (clk ,rst_n, load_n, up_down, ce, data_load, count_out, max_count, zero);
    9                                      parameter WIDTH = 4;
    10                                     input clk;
    11                                     input rst_n;
    12                                     input load_n;
    13                                     input up_down;
    14                                     input ce;
    15                                     input [WIDTH-1:0] data_load;
    16                                     output reg [WIDTH-1:0] count_out;
    17                                     output max_count;
    18                                     output zero;
    19
```

```
Branch Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Branches                        10        10         0     100.0

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

Branch Coverage for file counter.v --

------------------------------------IF Branch------------------------------------
    21                                      1000     Count coming in to IF
    21              1                          7         if (!rst_n)
    23              1                        707         else if (!load_n)
    25              1                        196         else if (ce)
                                              90     All False Count
Branch totals: 4 hits of 4 branches = 100.0%

------------------------------------IF Branch------------------------------------
    26                                      196     Count coming in to IF
    26              1                       116             if (up_down)
    28              1                        80             else
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    32                                      906     Count coming in to IF
    32              1                         5     assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
    32              2                       901     assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    33                                      906     Count coming in to IF
    33              1                        11     assign zero = (count_out == 0)? 1:0;
    33              2                       895     assign zero = (count_out == 0)? 1:0;
Branch totals: 2 hits of 2 branches = 100.0%
```

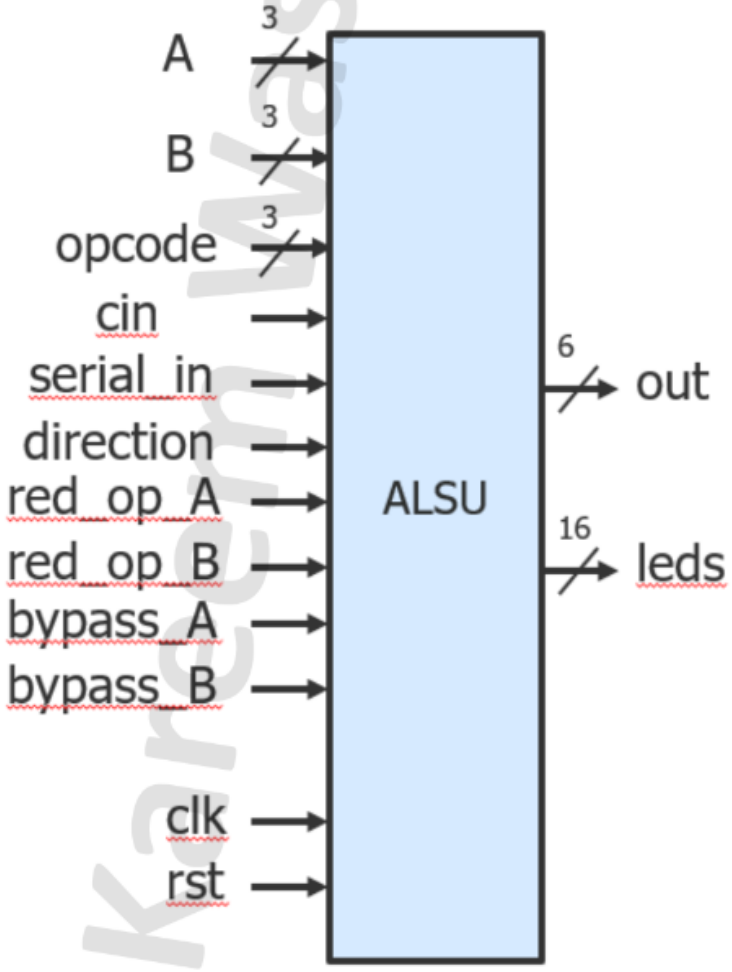
```
Toggle Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Toggle Bins                     46        46         0     100.0

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

Toggle Coverage for File counter.v --

        Line                                    Node       1H->0L       0L->1H  "Coverage"
--------------------------------------------------------------------------------------------
          10                                     clk            1            1       100.00
          11                                   rst_n            1            1       100.00
          12                                  load_n            1            1       100.00
          13                                 up_down            1            1       100.00
          14                                      ce            1            1       100.00
          15                           data_load[7]            1            1       100.00
          15                           data_load[6]            1            1       100.00
          15                           data_load[5]            1            1       100.00
          15                           data_load[4]            1            1       100.00
          15                           data_load[3]            1            1       100.00
          15                           data_load[2]            1            1       100.00
          15                           data_load[1]            1            1       100.00
          15                           data_load[0]            1            1       100.00
          16                           count_out[7]            1            1       100.00
          16                           count_out[6]            1            1       100.00
          16                           count_out[5]            1            1       100.00
          16                           count_out[4]            1            1       100.00
          16                           count_out[3]            1            1       100.00
          16                           count_out[2]            1            1       100.00
          16                           count_out[1]            1            1       100.00
          16                           count_out[0]            1            1       100.00
          17                              max_count            1            1       100.00
          18                                   zero            1            1       100.00
```

Question 3

➢ ALSU

2) 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.



1. Code Design

```
module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input [2:0] opcode;
input signed [2:0] A, B;    // first bug --> we must put it signed
output reg [15:0] leds;
output reg signed [5:0] out;  // second bug --> we must put it signed

reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg [2:0] opcode_reg, A_reg, B_reg;

wire invalid_red_op, invalid_opcode, invalid;

//Invalid handling
assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
assign invalid = invalid_red_op | invalid_opcode;

//Registering input signals
always @(posedge clk or posedge rst) begin
  if(rst) begin
    cin_reg <= 0;
    red_op_B_reg <= 0;
    red_op_A_reg <= 0;
    bypass_B_reg <= 0;
    bypass_A_reg <= 0;
    direction_reg <= 0;
    serial_in_reg <= 0;
    opcode_reg <= 0;
    A_reg <= 0;
    B_reg <= 0;
    B_reg <= 0;
  end else begin
    cin_reg <= cin;
    red_op_B_reg <= red_op_B;
    red_op_A_reg <= red_op_A;
    bypass_B_reg <= bypass_B;
    bypass_A_reg <= bypass_A;
    direction_reg <= direction;
    serial_in_reg <= serial_in;
    opcode_reg <= opcode;
    A_reg <= A;
    B_reg <= B;
  end
end

//leds output blinking
always @(posedge clk or posedge rst) begin
  if(rst) begin
    leds <= 0;
  end else begin
    if (invalid)
      leds <= ~leds;
    else
      leds <= 0;
  end
end
```

```
59  //ALSU output processing
60  always @(posedge clk or posedge rst) begin
61    if(rst) begin
62      out <= 0;
63    end
64    else begin
65      if (invalid)
66          out <= 0;
67      else if (bypass_A_reg && bypass_B_reg)
68        out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
69      else if (bypass_A_reg)
70        out <= A_reg;
71      else if (bypass_B_reg)
72        out <= B_reg;
73      else begin
74          case (opcode)
75            3'h0: begin
76              if (red_op_A_reg && red_op_B_reg)
77                out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;  // third bug is to replace AND with OR
78              else if (red_op_A_reg)
79                out <= |A_reg;
80              else if (red_op_B_reg)
81                out <= |B_reg;
82              else
83                out <= A_reg | B_reg;
84            end
85            3'h1: begin
86              if (red_op_A_reg && red_op_B_reg)
87                out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;  // fourth bug is to replace OR with XOR
88              else if (red_op_A_reg)
89                out <= ^A_reg;
90              else if (red_op_B_reg)
91                out <= ^B_reg;
92              else
93                out <= A_reg ^ B_reg;
94            end
95            3'h2:begin
96                  if(FULL_ADDER == "ON")  // fifth bug to add Cin operation in case of full adde
97                      out <= A_reg + B_reg + cin_reg ;
98                  else
99                      out <= A_reg + B_reg ;
100                 end
101           3'h3: out <= A_reg * B_reg;
102           3'h4: begin
103             if (direction_reg)
104               out <= {out[4:0], serial_in_reg};
105             else
106               out <= {serial_in_reg, out[5:1]};
107           end
```

```
            3'h5: begin
              if (direction_reg)
                out <= {out[4:0], out[5]};
              else
                out <= {out[0], out[5:1]};
            end
            default: out <= 0;
          endcase
      end
    end
end

endmodule
```

2. Verification plan

Label	Description	Stimulus Generation	Functiona	Functionality Check
ALSU_1	When the reset is asserted, the output value should be low	Directed at the start of the	-	A checker in the testbench to make sure the output is correct
ALSU_2	When the invalid is asserted , the output should be low	Randomization	-	A checker in the testbench to make sure the output is correct
ALSU_3	When the bypass_A is asserted, bypass_B is asserted and INPUT_PRIORITY = "A" the output count_out should take the A	Randomization	-	A checker in the testbench to make sure the output is correct
ALSU_4	When the bypass_A is asserted and bypass_B is deasserted the output count_out should take the A	Randomization	-	A checker in the testbench to make sure the output is correct
ALSU_5	When the bypass_A is deasserted and bypass_B is asserted the output count_out should take the B	Randomization		A checker in the testbench to make sure the output is correct
ALSU_6	test all the value of the opcode and the output for all case	Randomization		A checker in the testbench to make sure the output is correct

3. ALSU Package

```
package pack_ALSU;
typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
localparam MAXPOS = 3  ;
localparam MAXNEG = -4 ;
localparam zero = 0 ;

    class cla_ALSU ;
      rand  logic signed [2:0] A, B ;
      rand  reg_e opcode ;
      rand  logic       rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;



      constraint rst_n {rst dist {0:/99 , 1:/1  } ; }

      constraint input_A {
        if ( (opcode == ADD ) || (opcode == MULT ) )
        {
          A dist { MAXPOS:/25 ,  MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
        }
        else if (((opcode == XOR ) || (opcode == OR )) && (red_op_A == 1) && (red_op_B == 0)  )
        {
          B == 0 ;
          A dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
        }
        else
        {
          A inside { [MAXNEG : MAXPOS] } ;
        }
       }
        constraint input_B {
         if ( (opcode == ADD ) || (opcode == MULT ) )
         {
           B dist { MAXPOS:/25 ,  MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
         }
         else if (((opcode == XOR ) || (opcode == OR )) && (red_op_B == 1) && (red_op_A == 0) )
         {
           A == 0 ;
           B dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
         }
         else
         {
           B inside { [MAXNEG : MAXPOS]  } ;
         }
        }

        constraint input_opcode {opcode dist {[0:5]:/90 , [6:7]:/10  } ; }

        constraint input_bypass_A {bypass_A dist {1:=90 , 0:=10  } ; }
        constraint input_bypass_B {bypass_B dist {1:=90 , 0:=10  } ; }

      constraint input_red_op_A {red_op_A dist {1:/10 , 0:/90  } ; }
      constraint input_red_op_B {red_op_B dist {1:/10 , 0:/90  } ; }


      function void print ();
       $display("A = 0h%0h  , B = 0h%0h , opcode = 0h%0h , rst = 0h%0h , cin = 0h%0h  " , this.A , this.B , this.opcode , this.rst , this.cin  ) ;
       endfunction

       endclass

endpackage
```

4. ALSU Testbench

```
import pack_ALSU::*;

module ALSU_tb();

  parameter WIDTH = 3;
  parameter INPUT_PRIORITY = "A";
  parameter FULL_ADDER = "ON";

  logic signed [2:0] A, B ;
  logic [2:0] opcode ;
  logic       clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
  logic [15:0] leds;
  logic signed [5:0] out;

  typedef enum {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
  integer i ;
  integer correct_counter = 0 ;
  integer error_counter = 0 ;
  logic [5:0] old_count_out ;
  logic invalid ;
  logic test ;

  ALSU #(.INPUT_PRIORITY(INPUT_PRIORITY) , .FULL_ADDER(FULL_ADDER)  ) dut (.*);

  always #10 clk = ~clk ;

  cla_ALSU trans1 , trans2 ;

 initial begin
  clk       = 0 ;
  cin       = 0 ;
  red_op_A  = 0 ;
  red_op_B  = 0 ;
  bypass_A  = 0 ;
  bypass_B  = 0 ;
  direction = 0 ;
  serial_in = 0 ;
  A         = 0 ;
  B         = 0 ;
  opcode    = 0 ;
  trans1 = new() ;


  rest ();
  #10 ;

  repeat(1000000) begin
  assert(trans1.randomize());

  A         = trans1.A         ;
  B         = trans1.B         ;
  cin       = trans1.cin       ;
  red_op_A  = trans1.red_op_A  ;
  red_op_B  = trans1.red_op_B  ;
  bypass_A  = trans1.bypass_A  ;
  bypass_B  = trans1.bypass_B  ;
  direction = trans1.direction ;
  serial_in = trans1.serial_in ;
  opcode    = trans1.opcode    ;
  rst       = trans1.rst       ;

  golden_model ();

 end

  $display ("error_counter = %0d " ,error_counter );
  $display ("correct_counter = %0d " ,correct_counter );
  #100 ;

 $stop ;

end
```

```
76
77      task check_result (input logic signed [5:0] expected_result_out , input [15:0] expected_result_leds );
78        @(negedge clk );
79        @(negedge clk );
80        if( (expected_result_out != out) && (expected_result_leds != leds) )
81            begin
82                $display (":error");
83                old_count_out = out ;
84                error_counter = error_counter +1 ;
85                test = 1 ;
86
87            end
88        else
89            begin
90                correct_counter = correct_counter + 1 ;
91                old_count_out = out ;
92                test = 0 ;
93            end
94    endtask
```

```
96  task golden_model ();
97
98    invalid = (((red_op_A | red_op_B) & (opcode[1] | opcode[2])) | (opcode[1] & opcode[2]) ) ;
99    if (rst)
100   check_result(0,0);
101   else if (invalid)
102   check_result(0,'hffff);
103   else if(bypass_A && bypass_B)
104   begin
105     if (INPUT_PRIORITY == "A")
106     check_result(A,0);
107       else
108       check_result(B,0);
109   end
110   else if (bypass_A)
111   check_result(A,0);
112   else if (bypass_B)
113   check_result(B,0);
114   else begin
115     case (opcode)
116       3'h0: begin
117         if (red_op_A && red_op_B)
118         begin
119           if (INPUT_PRIORITY == "A")
120           check_result(|A,0);
121             else
122             check_result(|B,0);
123         end
```

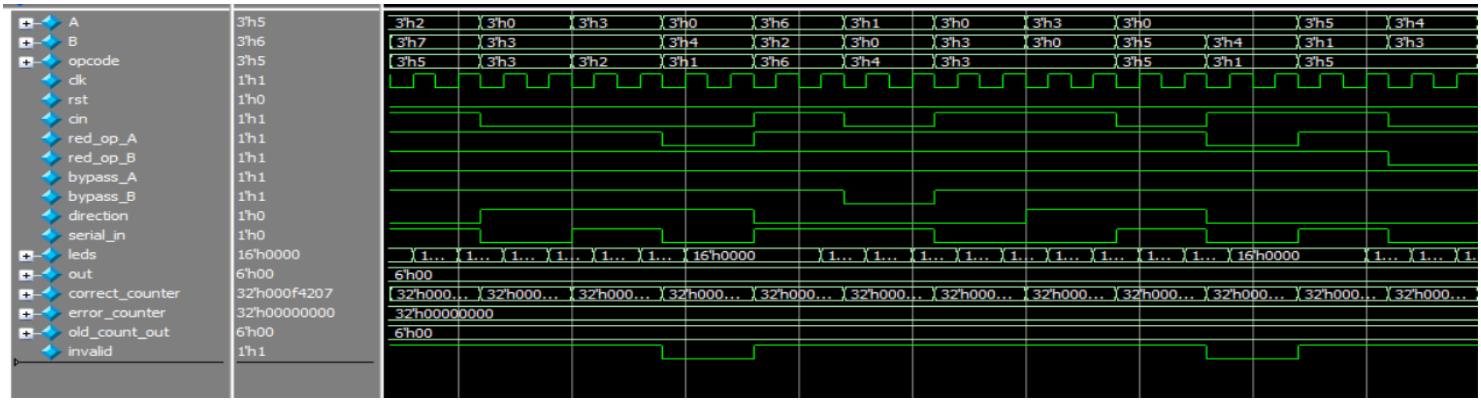

```
96  task golden_model ();
97
98    invalid = (((red_op_A | red_op_B) & (opcode[1] | opcode[2])) | (opcode[1] & opcode[2]) ) ;
99    if (rst)
100   check_result(0,0);
101   else if (invalid)
102   check_result(0,'hffff);
103   else if(bypass_A && bypass_B)
104   begin
105     if (INPUT_PRIORITY == "A")
106     check_result(A,0);
107       else
108       check_result(B,0);
109   end
110   else if (bypass_A)
111   check_result(A,0);
112   else if (bypass_B)
113   check_result(B,0);
114   else begin
115     case (opcode)
116       3'h0: begin
117         if (red_op_A && red_op_B)
118         begin
119           if (INPUT_PRIORITY == "A")
120           check_result(|A,0);
121             else
122             check_result(|B,0);
123         end
```

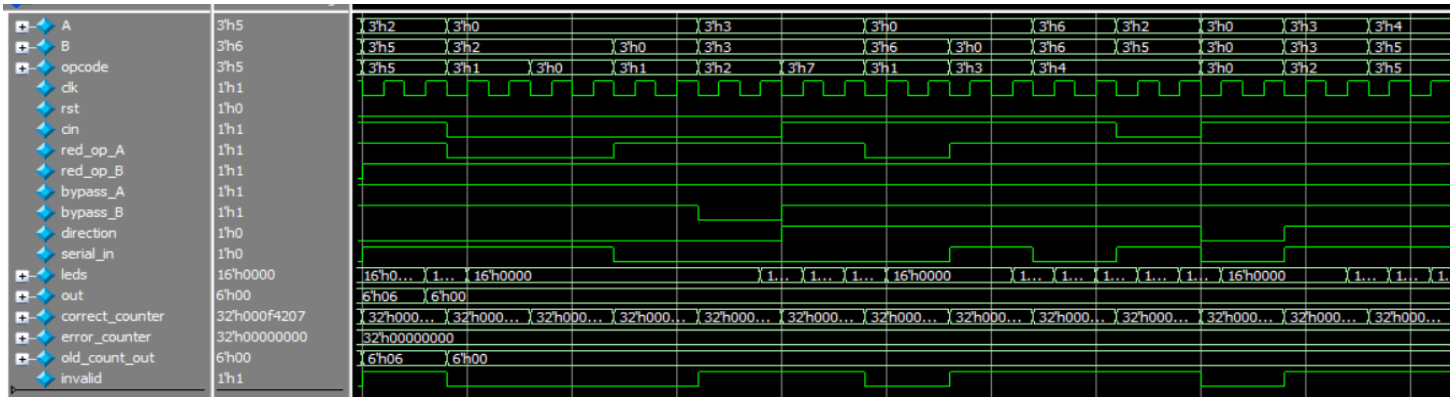
```
124                 else if (red_op_A)
125                 check_result(|A,0);
126                 else if (red_op_B)
127                 check_result(|B,0);
128                 else
129                 check_result(A|B,0);
130               end
131                3'h1: begin
132                 if (red_op_A && red_op_B)
133                 begin
134                   if (INPUT_PRIORITY == "A")
135                   check_result(^A,0);
136                     else
137                     check_result(^B,0);
138                 end
139                 else if (red_op_A)
140                 check_result(^A,0);
141                 else if (red_op_B)
142                 check_result(^B,0);
143                 else
144                 check_result(A^B,0);
145               end
146               3'h2: begin
147                   if(FULL_ADDER == "ON")
148                     check_result(A+B+cin,0);
149                   else
150                     check_result(A+B,0);
151                   end
152         3'h3: check_result(A*B,0);
153         3'h4: begin
154           if (direction)
155           check_result({old_count_out[4:0], serial_in},0);
156           else
157           check_result({serial_in, old_count_out[5:1]},0);
158         end
159         3'h5: begin
160           if (direction)
161           check_result({old_count_out[4:0], old_count_out[5]},0);
162           else
163           check_result({old_count_out[0], old_count_out[5:1]},0);
164         end
165       endcase
166   end
167   endtask
168
169
170     task rest ();
171       rst = 1 ;
172       #20
173       rst = 0 ;
174
175   endtask
176
177   endmodule
```

5. Do File

```
vlib work
vlog ALSU.v ALSU_tb.sv +cover -covercells
vsim -voptargs=+acc work.ALSU_tb -cover
add wave *
coverage save ALSU_tb.ucdb -onexit
run -all
```

6. Waveform





7. Transcript

```
#
# error_counter = 0
# correct_counter = 1000000
```

8. Code Coverage

```
================================================================================
=== File: ALSU.v
================================================================================
Statement Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           49        49         0     100.0

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

Statement Coverage for file ALSU.v --

    1                                           module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, by
    2                                           parameter INPUT_PRIORITY = "A";
    3                                           parameter FULL_ADDER = "ON";
    4                                           input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, dir
    5                                           input [2:0] opcode;
    6                                           input signed [2:0] A, B;
    7                                           output reg [15:0] leds;
    8                                           output reg signed [5:0] out;
    9
    10                                          reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_
    11                                          reg [2:0] opcode_reg, A_reg, B_reg;
    12
    13                                          wire invalid_red_op, invalid_opcode, invalid;
    14
    15                                          //Invalid handling
    16              1                  896366   assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_
    17              1                  856664   assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
    18              1                  494649   assign invalid = invalid_red_op | invalid_opcode;
    19
    20                                          //Registering input signals
    21              1                 1997369   always @(posedge clk or posedge rst) begin

Branch Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Branches                        30        30         0     100.0

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

Branch Coverage for file ALSU.v --

------------------------------------IF Branch------------------------------------
    22                                      1997369     Count coming in to IF
    22              1                         19911       if(rst) begin
    33              1                       1977458       end else begin
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    49                                      2009908     Count coming in to IF
    49              1                         29921       if(rst) begin
    52              1                       1513995           if (invalid)
    54              1                        465992           else
Branch totals: 3 hits of 3 branches = 100.0%
```

```
29
30  FSM Coverage:
31      Enabled Coverage            Active      Hits    Misses % Covered
32      ----------------            ------      ----    ------ ---------
33      FSMs                                                    100.0
34          States                       0         0         0     100.0
35          Transitions                  0         0         0     100.0
36  Toggle Coverage:
37      Enabled Coverage            Active      Hits    Misses % Covered
38      ----------------            ------      ----    ------ ---------
39      Toggle Bins                    118       118         0     100.0
40
41  ================================Toggle Details================================
42
43  Toggle Coverage for File ALSU.v --
44
45          Line                                  Node      1H->0L      0L->1H  "Coverage"
46  --------------------------------------------------------------------------------------
47              4                             serial_in           1           1      100.00
48              4                                   rst           1           1      100.00
```