P1)

- **Code**

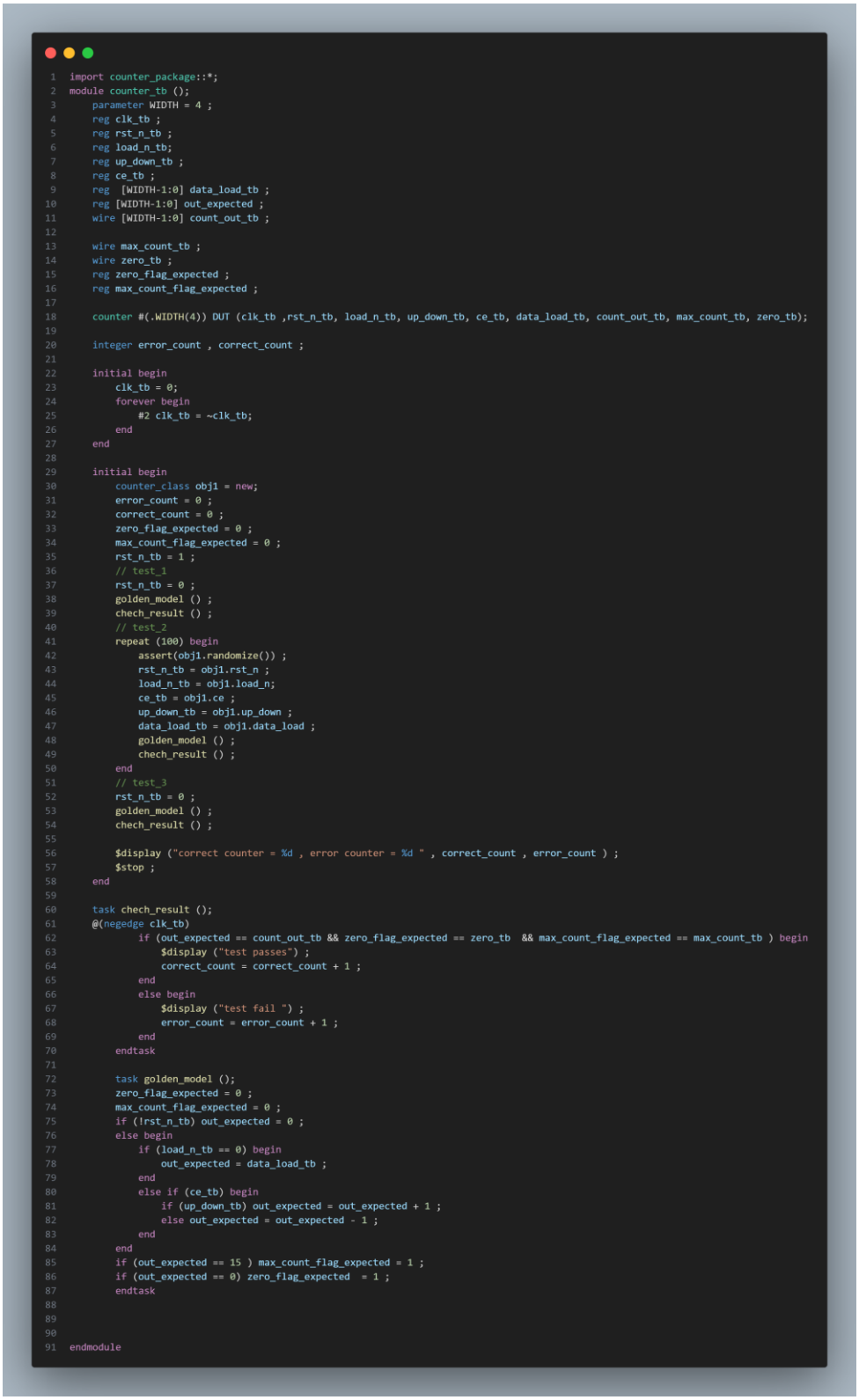
```
module test_dynamic_array ();
    int dyn_arr1 [] ,dyn_arr2 [] ;
    initial begin
        dyn_arr2 = '{9,1,8,3,4,4} ;
        dyn_arr1 = new[6] ;
        foreach (dyn_arr1[i]) begin
            dyn_arr1 [i] = i ;
        end
        $display ("elements of array 1 --> %p and it's size is %d" ,dyn_arr1 , dyn_arr1.size());
        dyn_arr1.delete() ;
        dyn_arr2.reverse ;
        $display ("the reversed array --> %p" , dyn_arr2);
        dyn_arr2.sort ;
        $display ("the sorted array --> %p" , dyn_arr2);
        dyn_arr2.rsort ;
        $display ("the reversed sorted array --> %p" , dyn_arr2);
        dyn_arr2.shuffle ;
        $display ("the shuffled array --> %p" , dyn_arr2);
    end
endmodule
```

- **Transcript**

```
VSIM 2> run -all
# elements of array 1 --> '{0, 1, 2, 3, 4, 5} and it's size is           6
# the reversed array --> '{4, 4, 3, 8, 1, 9}
# the sorted array --> '{1, 3, 4, 4, 8, 9}
# the reversed sorted array --> '{9, 8, 4, 4, 3, 1}
# the shuffled array --> '{8, 1, 4, 3, 9, 4}
```

P2)

- **Testbench code**



```
import counter_package::*;
module counter_tb ();
    parameter WIDTH = 4 ;
    reg clk_tb ;
    reg rst_n_tb ;
    reg load_n_tb;
    reg up_down_tb ;
    reg ce_tb ;
    reg  [WIDTH-1:0] data_load_tb ;
    reg [WIDTH-1:0] out_expected ;
    wire [WIDTH-1:0] count_out_tb ;

    wire max_count_tb ;
    wire zero_tb ;
    reg zero_flag_expected ;
    reg max_count_flag_expected ;

    counter #(.WIDTH(4)) DUT (clk_tb ,rst_n_tb, load_n_tb, up_down_tb, ce_tb, data_load_tb, count_out_tb, max_count_tb, zero_tb);

    integer error_count , correct_count ;

    initial begin
        clk_tb = 0;
        forever begin
            #2 clk_tb = ~clk_tb;
        end
    end

    initial begin
        counter_class obj1 = new;
        error_count = 0 ;
        correct_count = 0 ;
        zero_flag_expected = 0 ;
        max_count_flag_expected = 0 ;
        rst_n_tb = 1 ;
        // test_1
        rst_n_tb = 0 ;
        golden_model () ;
        chech_result () ;
        // test_2
        repeat (100) begin
            assert(obj1.randomize()) ;
            rst_n_tb = obj1.rst_n ;
            load_n_tb = obj1.load_n;
            ce_tb = obj1.ce ;
            up_down_tb = obj1.up_down ;
            data_load_tb = obj1.data_load ;
            golden_model () ;
            chech_result () ;
        end
        // test_3
        rst_n_tb = 0 ;
        golden_model () ;
        chech_result () ;

        $display ("correct counter = %d , error counter = %d " , correct_count , error_count ) ;
        $stop ;
    end

    task chech_result ();
    @(negedge clk_tb)
            if (out_expected == count_out_tb && zero_flag_expected == zero_tb  && max_count_flag_expected == max_count_tb ) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        endtask

        task golden_model ();
        zero_flag_expected = 0 ;
        max_count_flag_expected = 0 ;
        if (!rst_n_tb) out_expected = 0 ;
        else begin
            if (load_n_tb == 0) begin
                out_expected = data_load_tb ;
            end
            else if (ce_tb) begin
                if (up_down_tb) out_expected = out_expected + 1 ;
                else out_expected = out_expected - 1 ;
            end
        end
        if (out_expected == 15 ) max_count_flag_expected = 1 ;
        if (out_expected == 0) zero_flag_expected  = 1 ;
        endtask


endmodule
```

- **Package code**

```
1  package counter_package;
2      parameter WIDTH = 4 ;
3      class counter_class;
4          rand logic load_n;
5          rand logic rst_n ;
6          rand logic up_down;
7          rand logic ce ;
8          rand logic [3:0] data_load ;
9
10         constraint rst_n_dist {
11             rst_n dist {1:/95 , 0:/5} ;
12         }
13
14         constraint load_n_dist {
15             load_n dist {1:/70 , 0 :/30} ;
16         }
17
18         constraint ce_dist {
19             ce dist {1:/70 , 0 :/30} ;
20         }
21
22         constraint up_down_dist {
23             up_down dist {1:/70 , 0 :/30} ;
24         }
25     endclass
26 endpackage
```

- **Design Code**

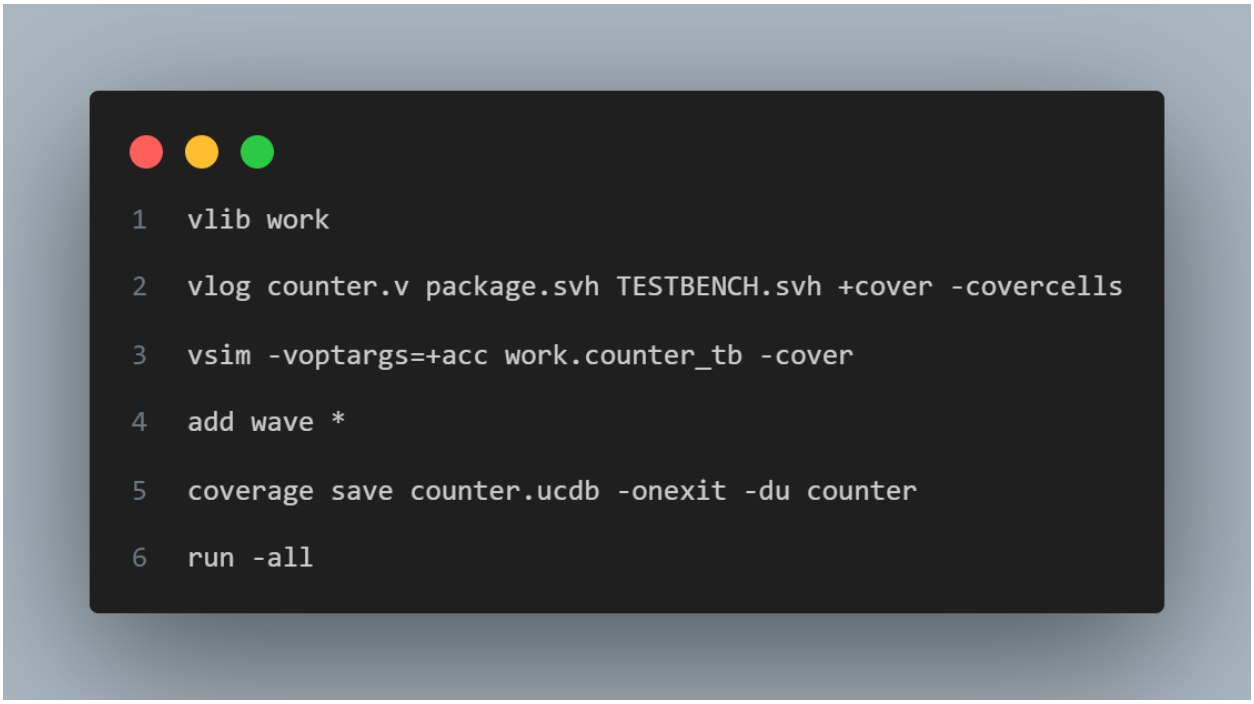
```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: Counter Design
//
////////////////////////////////////////////////////////////////////////////////
module counter (clk ,rst_n, load_n, up_down, ce, data_load, count_out, max_count, zero);
parameter WIDTH = 4;
input clk;
input rst_n;
input load_n;
input up_down;
input ce;
input [WIDTH-1:0] data_load;
output reg [WIDTH-1:0] count_out;
output max_count;
output zero;

always @(posedge clk) begin
    if (!rst_n)
        count_out <= 0;
    else if (!load_n)
        count_out <= data_load;
    else if (ce)
        if (up_down)
            count_out <= count_out + 1;
        else
            count_out <= count_out - 1;
end

assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
assign zero = (count_out == 0)? 1:0;

endmodule
```

There is no bugs found in the design

- **Verification plan**

A	B	C	D	E
Label	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
TEST_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation		A checker in the testbench to make sure the output is correct
TEST_2	Take 100 different input values from constraints , and chech the output with the golden model	Randomized		A checker in the testbench to make sure the output is correct
TEST_3	When the reset is asserted, the output value should be low	Directed at the end of the simulation		A checker in the testbench to make sure the output is correct

- **DO file**



```
vlib work
vlog counter.v package.svh TESTBENCH.svh +cover -covercells
vsim -voptargs=+acc work.counter_tb -cover
add wave *
coverage save counter.ucdb -onexit -du counter
run -all
```

- **Code coverage report**

```
# ================================================================================
# === Instance: /\counter_tb#DUT
# === Design Unit: work.counter
# ================================================================================
# Branch Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Branches                        10        10         0   100.00%
#
# ================================Branch Details================================
#
# Branch Coverage for instance /\counter_tb#DUT
#
#     Line         Item                      Count     Source
#     ----         ----                      -----     ------
#   File counter.v
# ------------------------------------IF Branch------------------------------------
#     21                                      102     Count coming in to IF
#     21              1                         5         if (!rst_n)
#
#     23              1                        32         else if (!load_n)
#
#     25              1                        47         else if (ce)
#
#                                              18     All False Count
# Branch totals: 4 hits of 4 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     26                                       47     Count coming in to IF
#     26              1                        38             if (up_down)
#
#     28              1                         9             else
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
```

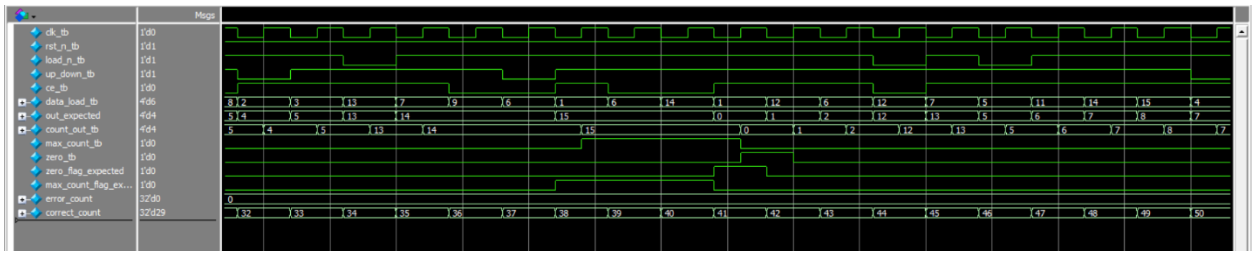
```
# ------------------------------------IF Branch------------------------------------
#     32                                      83     Count coming in to IF
#     32              1                        3     assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
#
#     32              2                       80     assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     33                                      83     Count coming in to IF
#     33              1                        8     assign zero = (count_out == 0)? 1:0;
#
#     33              2                       75     assign zero = (count_out == 0)? 1:0;
#
# 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 /\counter_tb#DUT  --
#
#   File counter.v
# ----------------Focused Condition View-------------------
# Line       32 Item    1  (count_out == {4{{1}}})
# Condition totals: 1 of 1 input term covered = 100.00%
#
#                Input Term   Covered  Reason for no coverage   Hint
#               -----------  --------  -----------------------  --------------
#   (count_out == {4{{1}}})         Y
#
#      Rows:       Hits  FEC Target                Non-masking condition(s)
#  ----------  ----------  ---------------------       --------------------------
#   Row   1:           1  (count_out == {4{{1}}})_0  -
#   Row   2:           1  (count_out == {4{{1}}})_1  -
```

```
# ----------------Focused Condition View-------------------
# Line       33 Item    1  (count_out == 0)
# Condition totals: 1 of 1 input term covered = 100.00%
#
#        Input Term   Covered  Reason for no coverage   Hint
#       -----------  --------  -----------------------  --------------
#   (count_out == 0)         Y
#
#      Rows:       Hits  FEC Target            Non-masking condition(s)
#  ---------  ---------  -------------------  -------------------------
#   Row   1:          1  (count_out == 0)_0    -
#   Row   2:          1  (count_out == 0)_1    -
#
#
# Statement Coverage:
#     Enabled Coverage             Bins      Hits    Misses  Coverage
#     ----------------             ----      ----    ------  --------
#     Statements                      7         7         0   100.00%
#
# ================================Statement Details================================
#
# Statement Coverage for instance /\counter_tb#DUT  --
#
#     Line         Item                    Count     Source
#     ----         ----                    -----     ------
#   File counter.v
#     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              1                     102     always @(posedge clk) begin
#
#     21                                              if (!rst_n)
#
#     22              1                       5             count_out <= 0;
#
#     23                                              else if (!load_n)
#
#     24              1                      32             count_out <= data_load;
#
#     25                                              else if (ce)
#
#     26                                                  if (up_down)
#
#     27              1                      38                 count_out <= count_out + 1;
#
#     28                                                  else
#
#     29              1                       9                 count_out <= count_out - 1;
#
#     30                                          end
#
#     31
#
#     32              1                      84     assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
#
#     33              1                      84     assign zero = (count_out == 0)? 1:0;
#
```

```
#
# Toggle Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Toggles                         30        30         0   100.00%
#
# ================================Toggle Details================================
#
# Toggle Coverage for instance /\counter_tb#DUT  --
#
#                                           Node      1H->0L      0L->1H  "Coverage"
#                                           ----------------------------------------
#                                             ce           1           1      100.00
#                                            clk           1           1      100.00
#                                  count_out[3-0]          1           1      100.00
#                                  data_load[0-3]          1           1      100.00
#                                         load_n           1           1      100.00
#                                      max_count           1           1      100.00
#                                          rst_n           1           1      100.00
#                                        up_down           1           1      100.00
#                                           zero           1           1      100.00
#
# Total Node Count     =         15
# Toggled Node Count   =         15
# Untoggled Node Count =          0
#
# Toggle Coverage      =     100.00% (30 of 30 bins)
#
#
# Total Coverage By Instance (filtered view): 100.00%
```

- **Questasim snippets**



- **Transcript**

```
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# correct counter =         102 , error counter =           0
# ** Note: $stop    : TESTBENCH.svh(59)
#    Time: 408 ns  Iteration: 1  Instance: /counter_tb
# Break in Module counter_tb at TESTBENCH.svh line 59
```

P3)

- **Testbench**

```
import ALSU_package::*;
module ALSU_tb ();
    reg clk_tb ;
    reg rst_tb ;
    reg signed [2:0] A_tb ;
    reg signed [2:0] B_tb ;
    reg signed [1:0] cin_tb ;
    reg serial_in_tb ;
    reg red_op_A_tb ;
    reg red_op_B_tb ;
    reg [2:0] opcode_tb;
    reg bypass_A_tb ;
    reg bypass_B_tb ;
    reg direction_tb;

    reg signed [5:0] out_expected ;


    wire [15:0] leds_tb ;
    wire signed [5:0] out_tb ;

    ALSU #(.INPUT_PRIORITY("A"),.FULL_ADDER("ON")) DUT (A_tb, B_tb, cin_tb,
                                                        serial_in_tb,
                                                        red_op_A_tb, red_op_B_tb,
                                                        opcode_tb, bypass_A_tb,
                                                        bypass_B_tb, clk_tb, rst_tb,
                                                        direction_tb, leds_tb, out_tb);


    integer error_count , correct_count ;

    initial begin
        clk_tb = 0;
        forever begin
            #2 clk_tb = ~clk_tb;
        end
    end

    initial begin
        ALSU_class obj1 = new ;
        rst_tb = 0 ;
        error_count = 0 ;
        correct_count = 0 ;
        out_expected = 0 ;
        assert_reset () ;

        repeat (2000) begin
            assert (obj1.randomize())
            rst_tb       = obj1.rst       ;
            A_tb         = obj1.A         ;
            B_tb         = obj1.B         ;
            cin_tb       = obj1.cin       ;
            serial_in_tb = obj1.serial_in ;
            red_op_A_tb  = obj1.red_op_A  ;
            red_op_B_tb  = obj1.red_op_B  ;
            bypass_A_tb  = obj1.bypass_A  ;
            bypass_B_tb  = obj1.bypass_B  ;
            direction_tb = obj1.direction ;
            opcode_tb    = obj1.opcode    ;
            golden_model () ;
            @(negedge clk_tb) ;
            @(negedge clk_tb) ;
            check_result () ;
        end
        opcode_tb = 3'bx ;
        golden_model () ;
        @(negedge clk_tb) ;
        check_result () ;

        assert_reset () ;

        $display ("correct counter = %d , error counter = %d " , correct_count , error_count ) ;
        $stop ;
    end

    task golden_model ();
        if (rst_tb) begin
            out_expected = 0 ;
        end
        else if (bypass_A_tb) out_expected = A_tb;
        else if (bypass_B_tb) out_expected = B_tb;
        else if ((opcode_tb == 3'b110) ||
                 (opcode_tb == 3'b111) ||
                 ((red_op_A_tb || red_op_B_tb) &&
                  (opcode_tb != 3'b000) && (opcode_tb != 3'b001))) out_expected = 0;
        else if (opcode_tb == 0) begin
            if (red_op_A_tb) out_expected = |A_tb ;
            else if (red_op_B_tb) out_expected = |B_tb ;
            else out_expected = A_tb | B_tb ;
        end
        else if (opcode_tb == 1) begin
            if (red_op_A_tb) out_expected = ^A_tb ;
            else if (red_op_B_tb) out_expected = ^B_tb ;
            else out_expected = A_tb ^ B_tb ;
        end
        else if (opcode_tb == 2) out_expected = A_tb + B_tb + cin_tb ;
        else if (opcode_tb == 3) out_expected = A_tb * B_tb ;
        else if (opcode_tb == 4) begin
            if (direction_tb) out_expected = {out_expected[4:0] , serial_in_tb } ;
            else              out_expected = {serial_in_tb , out_expected[5:1] } ;
        end
        else if (opcode_tb == 5) begin
            if (direction_tb) out_expected = {out_expected[4:0] , out_expected[5] } ;
            else              out_expected = {out_expected[0] , out_expected[5:1] } ;
        end
    endtask

    task check_result ();
        if (out_expected == out_tb) begin
            $display ("test passed") ;
            correct_count = correct_count + 1 ;
        end
        else begin
            $display ("test fail ") ;
            error_count = error_count + 1 ;
        end
    endtask

    task assert_reset ();
        rst_tb = 1 ;
        out_expected = 0 ;
        @(negedge clk_tb) ;
        check_result () ;
        rst_tb = 0 ;
    endtask
endmodule
```

- **Package code**

```
package ALSU_package;
    typedef enum logic [2:0] {
        OR, XOR, ADD, MULT, SHIFT, ROTATE, INVALID_6, INVALID_7
    } opcode_t ;
    class ALSU_class;
        rand logic rst ;
        rand logic signed [2:0] A ;
        rand logic signed [2:0] B ;
        rand logic signed [1:0] cin ;
        rand logic serial_in ;
        rand logic red_op_A ;
        rand logic red_op_B ;
        rand logic bypass_A ;
        rand logic bypass_B ;
        rand logic direction;
        rand opcode_t opcode ;
        bit [2:0] last_opcode;

        constraint Arethmatic {
            if (opcode == ADD || opcode == MULT) {
                A dist {-4:/25, 0:/25, 3:/25, -3:/5, -2:/5, -1:/5, 1:/5, 2:/5};
                B dist {-4:/25, 0:/25, 3:/25, -3:/5, -2:/5, -1:/5, 1:/5, 2:/5};
            }
        }

        constraint reduction_A {
            if ( (opcode == OR || opcode == XOR) && red_op_A ) {
                A dist {3'b001:/25, 3'b010:/25, 3'b100:/25,
                        3'b000:/5, 3'b011:/5, 3'b101:/5, 3'b110:/5, 3'b111:/5};
                B dist {3'b000:/100} ;
            }
        }

        constraint reduction_B {
            if ( (opcode == OR || opcode == XOR) && red_op_B ) {
                B dist {3'b001:/25, 3'b010:/25, 3'b100:/25,
                        3'b000:/5, 3'b011:/5, 3'b101:/5, 3'b110:/5, 3'b111:/5};
                A dist {3'b000:/100} ;
            }
        }




        constraint OPCODE {
            opcode dist { [0:5] :/ 95,  [6:7] :/ 5 };
            if (last_opcode inside {4,5})
                !(opcode inside {4,5}); // block consecutive 4/5
        }

        constraint BYPASS {
            bypass_A dist {0:/95 , 1:/5} ;
            bypass_B dist {0:/95 , 1:/5} ;
        }

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

        constraint RED_OP {
            if (opcode == OR || opcode == XOR ) {
                red_op_A dist {0:/80 , 1:/20} ;
                red_op_B dist {0:/70 , 1:/30} ;
            }
            else {
                red_op_A dist {0:/95 , 1:/5} ;
                red_op_B dist {0:/95 , 1:/5} ;
            }
        }

        function void post_randomize();
            last_opcode = opcode; // remember for next iteration
        endfunction
    endclass
endpackage
```

- **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, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input signed [1:0] cin ;
input [2:0] opcode;
input signed [2:0] A, B;
output reg [15:0] leds;
output reg signed [5:0] out;

reg red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg signed [1:0] cin_reg;
reg [2:0] opcode_reg;
reg signed [2:0] A_reg, B_reg;

wire invalid_red_op , 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;
  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

//ALSU output processing
always @(posedge clk or posedge rst) begin
  if(rst) begin
    out <= 0;
  end
  else begin
    if (bypass_A_reg && bypass_B_reg)
      out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
    else if (bypass_A_reg)
      out <= A_reg;
    else if (bypass_B_reg)
      out <= B_reg;
    else if (invalid)
       out <= 0;
    else begin
        case (opcode_reg)
          3'h0: begin
            if (red_op_A_reg && red_op_B_reg)
              out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
            else if (red_op_A_reg)
              out <= |A_reg;
            else if (red_op_B_reg)
              out <= |B_reg;
            else
              out <= A_reg | B_reg;
          end
          3'h1: begin
            if (red_op_A_reg && red_op_B_reg)
              out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
            else if (red_op_A_reg)
              out <= ^A_reg;
            else if (red_op_B_reg)
              out <= ^B_reg;
            else
              out <= A_reg ^ B_reg;
          end
          3'h2: out <= (FULL_ADDER == "ON")? A_reg + B_reg + cin_reg : A_reg + B_reg;
          3'h3: out <= A_reg * B_reg  ;
          3'h4: begin
            if (direction_reg)
              out <= {out[4:0], serial_in_reg};
            else
              out <= {serial_in_reg, out[5:1]};
          end
          3'h5: begin
            if (direction_reg)
              out <= {out[4:0], out[5]};
            else
              out <= {out[0], out[5:1]};
          end

        endcase
    end
  end
end

endmodule
```

Bugs found in design

- Change case to depend on **opcode_reg** value instead of **opcode**
- Add if condition in case of addition to be depending on the value of parameter FULL_ADDER (“ON” take in consider CIN , “OFF” ignore CIN)

- **Verification plan**

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
TEST_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation		A checker in the testbench to make sure the output is correct
TEST_2	Take 2000 different input values from constraints , and chech the output with the golden model	Randomized		A checker in the testbench to make sure the output is correct
TEST_3	When the reset is asserted, the output value should be low	Directed at the end of the simulation		A checker in the testbench to make sure the output is correct

- **DO file**

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

- **Code coverage**

```
#
# ================================================================================
# === Instance: /\ALSU_tb#DUT
# === Design Unit: work.ALSU
# ================================================================================
# Branch Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Branches                        31        31         0   100.00%
#
# ================================Branch Details================================
#
# Branch Coverage for instance /\ALSU_tb#DUT
#
#     Line         Item                      Count     Source
#     ----         ----                      -----     ------
#   File ALSU.v
# ------------------------------------IF Branch------------------------------------
#     25                                     4001     Count coming in to IF
#     25              1                        34       if(rst) begin
#
#     36              1                      3967       end else begin
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     52                                     4020     Count coming in to IF
#     52              1                        49       if(rst) begin
#
#     54              1                      3971       end else begin
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     55                                     3971     Count coming in to IF
#     55              1                       412            if (invalid)
#
#     57              1                      3559            else
#
# Branch totals: 2 hits of 2 branches = 100.00%
```

```
#
# ------------------------------------IF Branch------------------------------------
#     55                                     3971     Count coming in to IF
#     55              1                       412            if (invalid)
#
#     57              1                      3559            else
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     64                                     3687     Count coming in to IF
#     64              1                        34       if(rst) begin
#
#     67              1                      3653       else begin
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     68                                     3653     Count coming in to IF
#     68              1                         7         if (bypass_A_reg && bypass_B_reg)
#
#     70              1                       166         else if (bypass_A_reg)
#
#     72              1                       196         else if (bypass_B_reg)
#
#     74              1                       309         else if (invalid)
#
#     76              1                      2975         else begin
#
# Branch totals: 5 hits of 5 branches = 100.00%
#
# ------------------------------------CASE Branch------------------------------------
#     77                                     2975     Count coming in to CASE
#     78              1                       561              3'h0: begin
#
#     88              1                       606              3'h1: begin
#
#     98              1                       614              3'h2: out <= (FULL_ADDER == "ON")? A_reg + B_reg + cin_reg : A_reg + B_reg;
#
#     99              1                       537              3'h3: out <= A_reg * B_reg  ;
#
```

```
#
#     100             1                       358              3'h4: begin
#
#     106             1                       299              3'h5: begin
#
# Branch totals: 6 hits of 6 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     79                                      561     Count coming in to IF
#     79              1                        12                 if (red_op_A_reg && red_op_B_reg)
#
#     81              1                        81                 else if (red_op_A_reg)
#
#     83              1                       131                 else if (red_op_B_reg)
#
#     85              1                       337                 else
#
# Branch totals: 4 hits of 4 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     89                                      606     Count coming in to IF
#     89              1                         7                 if (red_op_A_reg && red_op_B_reg)
#
#     91              1                        93                 else if (red_op_A_reg)
#
#     93              1                       170                 else if (red_op_B_reg)
#
#     95              1                       336                 else
#
# Branch totals: 4 hits of 4 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     101                                     358     Count coming in to IF
#     101             1                       165                 if (direction_reg)
#
#     103             1                       193                 else
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
```

```
#
# -------------------------------------IF Branch-------------------------------------
#     107                                      299     Count coming in to IF
#     107             1                        137                 if (direction_reg)
#
#     109             1                        162                 else
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
#
# Condition Coverage:
#     Enabled Coverage              Bins   Covered    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Conditions                       6         6         0   100.00%
#
# ================================Condition Details================================
#
# Condition Coverage for instance /\ALSU_tb#DUT  --
#
#   File ALSU.v
# ----------------Focused Condition View-------------------
# Line       68 Item    1  (bypass_A_reg && bypass_B_reg)
# Condition totals: 2 of 2 input terms covered = 100.00%
#
#     Input Term   Covered  Reason for no coverage   Hint
#    -----------  --------  ----------------------  --------------
#   bypass_A_reg         Y
#   bypass_B_reg         Y
#
#      Rows:       Hits  FEC Target            Non-masking condition(s)
#  ---------  ----------  --------------------  -------------------------
#   Row   1:           1  bypass_A_reg_0        -
#   Row   2:           1  bypass_A_reg_1        bypass_B_reg
#   Row   3:           1  bypass_B_reg_0        bypass_A_reg
#   Row   4:           1  bypass_B_reg_1        bypass_A_reg
```

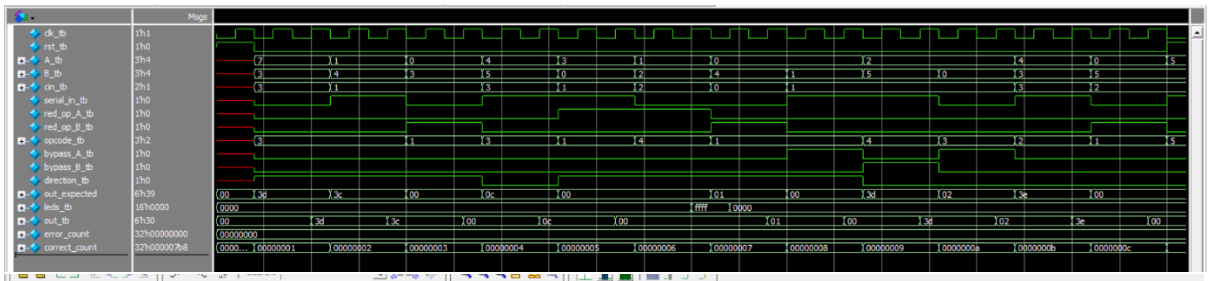
```
# ----------------Focused Condition View-------------------
# Line       79 Item    1  (red_op_A_reg && red_op_B_reg)
# Condition totals: 2 of 2 input terms covered = 100.00%
#
#     Input Term   Covered  Reason for no coverage   Hint
#    -----------  --------  ----------------------  --------------
#   red_op_A_reg         Y
#   red_op_B_reg         Y
#
#      Rows:       Hits  FEC Target            Non-masking condition(s)
#  ---------  ---------  --------------------  --------------------------
#   Row   1:          1  red_op_A_reg_0        -
#   Row   2:          1  red_op_A_reg_1        red_op_B_reg
#   Row   3:          1  red_op_B_reg_0        red_op_A_reg
#   Row   4:          1  red_op_B_reg_1        red_op_A_reg
#
# ----------------Focused Condition View-------------------
# Line       89 Item    1  (red_op_A_reg && red_op_B_reg)
# Condition totals: 2 of 2 input terms covered = 100.00%
#
#     Input Term   Covered  Reason for no coverage   Hint
#    -----------  --------  ----------------------  --------------
#   red_op_A_reg         Y
#   red_op_B_reg         Y
#
#      Rows:       Hits  FEC Target            Non-masking condition(s)
#  ---------  ---------  --------------------  --------------------------
#   Row   1:          1  red_op_A_reg_0        -
#   Row   2:          1  red_op_A_reg_1        red_op_B_reg
#   Row   3:          1  red_op_B_reg_0        red_op_A_reg
#   Row   4:          1  red_op_B_reg_1        red_op_A_reg
#
#
# Expression Coverage:
#     Enabled Coverage              Bins    Covered    Misses  Coverage
#     ----------------              ----    -------    ------  --------
#     Expressions                      8          8         0   100.00%
#
```

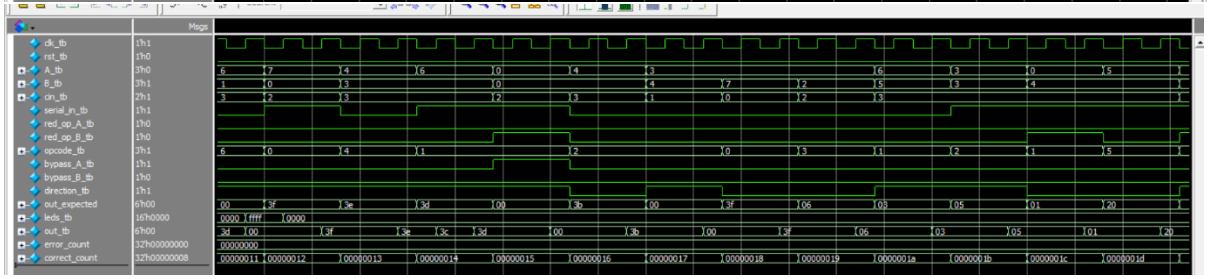
```
#
# ================================Expression Details================================
#
# Expression Coverage for instance /\ALSU_tb#DUT  --
#
#   File ALSU.v
# ----------------Focused Expression View-----------------
# Line       19 Item    1  ((red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]))
# Expression totals: 4 of 4 input terms covered = 100.00%
#
#     Input Term   Covered  Reason for no coverage   Hint
#    -----------  --------  ----------------------  --------------
#   red_op_A_reg         Y
#   red_op_B_reg         Y
#   opcode_reg[1]        Y
#   opcode_reg[2]        Y
#
#      Rows:       Hits  FEC Target            Non-masking condition(s)
#  ---------  ---------  --------------------  ------------------------
#   Row   1:          1  red_op_A_reg_0        ((opcode_reg[1] | opcode_reg[2]) && ~red_op_B_reg)
#   Row   2:          1  red_op_A_reg_1        ((opcode_reg[1] | opcode_reg[2]) && ~red_op_B_reg)
#   Row   3:          1  red_op_B_reg_0        ((opcode_reg[1] | opcode_reg[2]) && ~red_op_A_reg)
#   Row   4:          1  red_op_B_reg_1        ((opcode_reg[1] | opcode_reg[2]) && ~red_op_A_reg)
#   Row   5:          1  opcode_reg[1]_0       ((red_op_A_reg | red_op_B_reg) && ~opcode_reg[2])
#   Row   6:          1  opcode_reg[1]_1       ((red_op_A_reg | red_op_B_reg) && ~opcode_reg[2])
#   Row   7:          1  opcode_reg[2]_0       ((red_op_A_reg | red_op_B_reg) && ~opcode_reg[1])
#   Row   8:          1  opcode_reg[2]_1       ((red_op_A_reg | red_op_B_reg) && ~opcode_reg[1])
#
# ----------------Focused Expression View-----------------
# Line       20 Item    1  (opcode_reg[1] & opcode_reg[2])
# Expression totals: 2 of 2 input terms covered = 100.00%
#
#     Input Term   Covered  Reason for no coverage   Hint
#    -----------  --------  ----------------------  --------------
#   opcode_reg[1]        Y
#   opcode_reg[2]        Y
#
```

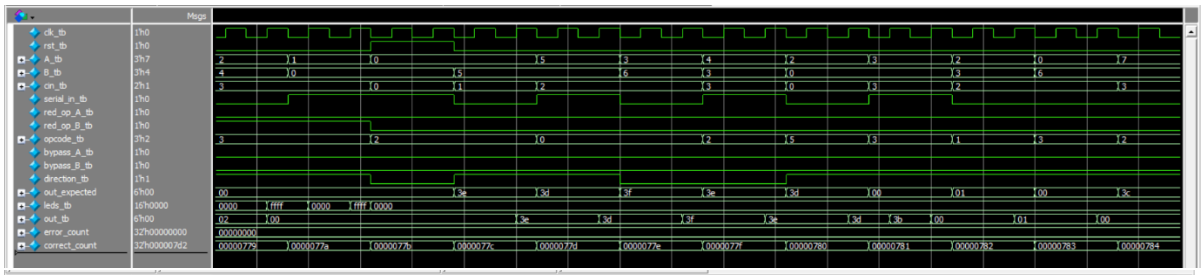
```
#
# ----------------Focused Expression View-----------------
# Line       20 Item    1  (opcode_reg[1] & opcode_reg[2])
# Expression totals: 2 of 2 input terms covered = 100.00%
#
#      Input Term   Covered  Reason for no coverage   Hint
#     -----------  --------  ----------------------  --------------
#   opcode_reg[1]         Y
#   opcode_reg[2]         Y
#
#      Rows:       Hits  FEC Target            Non-masking condition(s)
#  ---------  ---------  --------------------  -------------------------
#   Row   1:          1  opcode_reg[1]_0       opcode_reg[2]
#   Row   2:          1  opcode_reg[1]_1       opcode_reg[2]
#   Row   3:          1  opcode_reg[2]_0       opcode_reg[1]
#   Row   4:          1  opcode_reg[2]_1       opcode_reg[1]
#
# ----------------Focused Expression View-----------------
# Line       21 Item    1  (invalid_red_op | invalid_opcode)
# Expression totals: 2 of 2 input terms covered = 100.00%
#
#       Input Term   Covered  Reason for no coverage   Hint
#      -----------  --------  ----------------------  --------------
#   invalid_red_op         Y
#   invalid_opcode         Y
#
#      Rows:       Hits  FEC Target            Non-masking condition(s)
#  ---------  ---------  --------------------  -------------------------
#   Row   1:          1  invalid_red_op_0      ~invalid_opcode
#   Row   2:          1  invalid_red_op_1      ~invalid_opcode
#   Row   3:          1  invalid_opcode_0      ~invalid_red_op
#   Row   4:          1  invalid_opcode_1      ~invalid_red_op
#
#
# Statement Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Statements                      48        48         0   100.00%
#
```

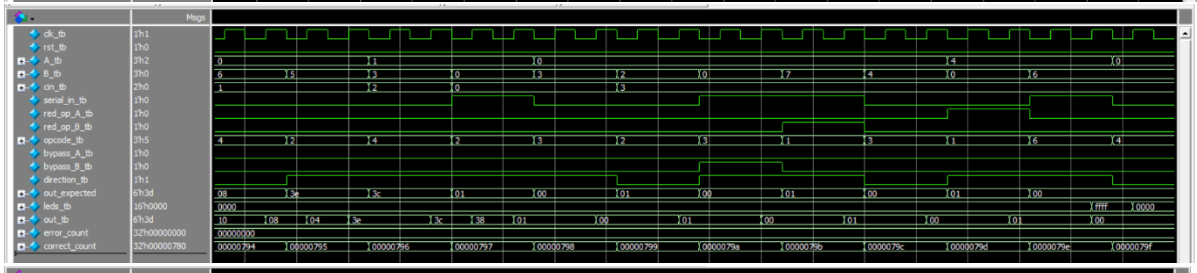
```
# ================================Toggle Details================================
#
# Toggle Coverage for instance /\ALSU_tb#DUT  --
#
#                                             Node      1H->0L      0L->1H  "Coverage"
#                                             ----------------------------------------
#                                           A[0-2]           1           1      100.00
#                                        A_reg[2-0]          1           1      100.00
#                                           B[0-2]           1           1      100.00
#                                        B_reg[2-0]          1           1      100.00
#                                          bypass_A          1           1      100.00
#                                       bypass_A_reg         1           1      100.00
#                                          bypass_B          1           1      100.00
#                                       bypass_B_reg         1           1      100.00
#                                          cin[0-1]          1           1      100.00
#                                       cin_reg[1-0]         1           1      100.00
#                                              clk           1           1      100.00
#                                        direction           1           1      100.00
#                                     direction_reg          1           1      100.00
#                                          invalid           1           1      100.00
#                                    invalid_opcode          1           1      100.00
#                                    invalid_red_op          1           1      100.00
#                                       leds[15-0]           1           1      100.00
#                                      opcode[0-2]           1           1      100.00
#                                   opcode_reg[2-0]          1           1      100.00
#                                           out[5-0]         1           1      100.00
#                                          red_op_A          1           1      100.00
#                                       red_op_A_reg         1           1      100.00
#                                          red_op_B          1           1      100.00
#                                       red_op_B_reg         1           1      100.00
#                                                rst         1           1      100.00
#                                          serial_in         1           1      100.00
#                                       serial_in_reg        1           1      100.00
#
# Total Node Count     =         61
# Toggled Node Count   =         61
# Untoggled Node Count =          0
#
# Toggle Coverage      =     100.00% (122 of 122 bins)
#
#
```

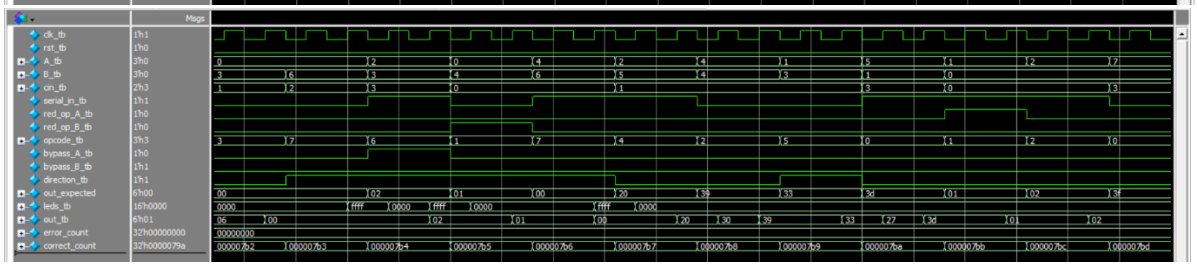
- **Questasim snippets**

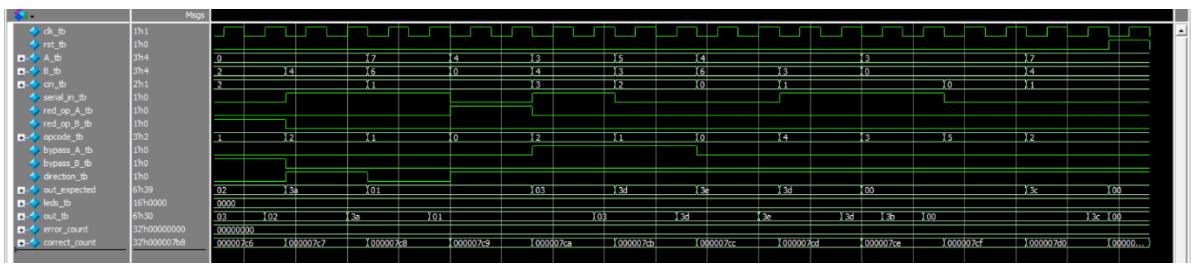












- **Transcript**

```
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# correct counter =        2002 , error counter =           0
# ** Note: $stop    : TESTBENCH.svh(68)
#    Time: 16008 ns  Iteration: 1  Instance: /ALSU_tb
# Break in Module ALSU_tb at TESTBENCH.svh line 68
```