Assignment 2

Digital Design Verification

Contents

1 Q1: test dynamic array

1.1 1. Testbench

```
module dyn_array_tb;

//=============================================================
// dynamic array declaration (with initialization for dyn_arr2)
//=============================================================
int dyn_arr1 [] ;
int dyn_arr2 [] = '{9,1,8,3,4,4};


initial begin

    //=======================================
    // allocate six elements in dyn_arr1
    //=======================================
    dyn_arr1 = new[6];

    //=======================================
    // initialize array dyn_arr1 with index
    //=======================================
    foreach (dyn_arr1[i]) begin
        dyn_arr1[i] = i;
    end

    //=======================================
    // display dyn_arr1 elements and size
    //=======================================
    $display("dyn_arr1 elements: %p , size : %0d",dyn_arr1 ,dyn_arr1.size());

    //=======================================
    // delete array dyn_arr1
    //=======================================
    dyn_arr1.delete();
    $display("After deleting dyn_arr1 , Size: %0d", dyn_arr1.size());

    //=======================================
    // Reverse sort dyn_arr2
    //=======================================
    dyn_arr2.reverse();
    $display("After reverse dyn_arr2: %p", dyn_arr2);

    //=======================================
    // Sort dyn_arr2
    //=======================================
    dyn_arr2.sort();
    $display("After sort dyn_arr2: %p", dyn_arr2);

    //=======================================
    // Reverse sort dyn_arr2
    //=======================================
    dyn_arr2.rsort();
    $display("After reverse sort dyn_arr2: %p", dyn_arr2);

    //=======================================
    // Shuffle dyn_arr2
    //=======================================
    dyn_arr2.shuffle();
    $display("After shuffle dyn_arr2: %p", dyn_arr2);

end
endmodule
```

```
# dyn_arr1 elements: '{0, 1, 2, 3, 4, 5} , size : 6
# After deleting dyn_arr1, Size: 0
# After reverse dyn_arr2: '{4, 4, 3, 8, 1, 9}
# After sort dyn_arr2: '{1, 3, 4, 4, 8, 9}
# After reverse sort dyn_arr2: '{9, 8, 4, 4, 3, 1}
# After shuffle dyn_arr2: '{8, 1, 4, 3, 9, 4}
```

Figure 1: Transcript : all test cases passed

2 Q2: Counter

2.1 1. Testbench code

```
`timescale 1ns/1ps

//=============================================================
// Import the counter package containing parameters and classes
//=============================================================
import counter_pkg::*;  // Bring in our parameter + class

module counter_tb;

  //=============================================================
  // Local parameters: Override the default WIDTH if needed
  //=============================================================
  localparam int TB_WIDTH = WIDTH; // or you can set TB_WIDTH = 4, 6, or 8.

  //=============================================================
  // Testbench signals: Declare all signals used to interface with DUT
```

```
  //================================================================
  bit clk;       // Clock signal
  bit rst_n;     // Active-low reset
  bit load_n;    // Load enable (active-low)
  bit up_down;   // Direction control: 1 for up, 0 for down
  bit ce;        // Count enable
  logic [TB_WIDTH-1:0] data_load; // Data to load into counter
  wire [TB_WIDTH-1:0] count_out;  // Counter output
  wire max_count; // Flag indicating max count reached
  wire zero;      // Flag indicating count is zero

  //================================================================
  // Instantiate the DUT (Device Under Test) with overridden WIDTH
  //================================================================
  counter #(.WIDTH(TB_WIDTH)) dut (
    .clk      (clk),
    .rst_n    (rst_n),
    .load_n   (load_n),
    .up_down  (up_down),
    .ce       (ce),
    .data_load(data_load),
    .count_out(count_out),
    .max_count(max_count),
    .zero     (zero)
  );

  //================================================================
  // Create an instance of the random config class
  //================================================================
  counter_cfg cfg;

  //================================================================
  // Reference model state (golden model for verification)
  //================================================================
  logic [TB_WIDTH-1:0] golden_count; // Reference count value
  wire golden_max; // Reference max count flag
  wire golden_zero; // Reference zero flag

  //================================================================
  // Clock generator: Generates a 10 ns period clock signal
  //================================================================
  initial begin
    clk = 0;
    forever begin
        #5 clk = ~clk; // Toggle clock every 5ns
        cfg.clk = clk;  // Update clock in config
    end
  end

  //================================================================
  // Task: synchronous reset assertion
  //================================================================
  task assert_reset();
    begin
      rst_n = 0;   // Activate reset (active-low)
      @(posedge clk); // wait 1 clock cycle
      @(posedge clk); // wait another clock cycle
      rst_n = 1;   // Deactivate reset
    end
  endtask

  //================================================================
  // Golden model logic: Simulate expected counter behavior
  //================================================================
  always @(posedge clk) begin
    if (!rst_n) begin
      golden_count <= '0; // Reset golden count to zero
    end else if (!load_n) begin
      golden_count <= data_load; // Load value into golden count
    end else if (ce) begin
      if (up_down)
        golden_count <= golden_count + 1; // Increment
      else
        golden_count <= golden_count - 1; // Decrement
    end
  end

  // Calculate golden model outputs
  assign golden_max  = (golden_count == {TB_WIDTH{1'b1}});
  assign golden_zero = (golden_count == 0);

  //================================================================
  // Task: Compare DUT outputs with golden model
  //================================================================
  task check_result();
    if (count_out !== golden_count) begin
      $error("[CHECK_RESULT] Mismatch! DUT=%0d, GOLDEN=%0d", count_out, golden_count);
    end
    if (max_count !== golden_max) begin
      $error("[CHECK_RESULT] max_count mismatch! DUT=%b, GOLDEN=%b", max_count, golden_max);
    end
    if (zero !== golden_zero) begin
      $error("[CHECK_RESULT] zero mismatch! DUT=%b, GOLDEN=%b", zero, golden_zero);
    end
  endtask

  //================================================================
  // Main verification process: Drives the testbench flow
  //================================================================
  initial begin
```

```
    // Create config object for randomization
    cfg = new();

    // 1) Assert reset
    assert_reset();

    // 2) Run random tests
    for (int i = 0; i < 40; i++) begin
      if (!cfg.randomize()) begin
        $error("Randomization failed!");
        $finish;
      end

      // Drive signals from random config
      rst_n    = cfg.rst_n;
      load_n   = cfg.load_n;
      up_down  = cfg.up_down;
      ce       = cfg.ce;
      data_load= cfg.data_load;

      @(posedge clk); // Wait for clock edge

      // Compare DUT with golden model
      check_result();
    end

    $display("All done. End of simulation.");
    $finish; // End simulation
  end

endmodule
```

2.2 2. Package code

```
package counter_pkg;

    //============================
    // 1) Declare the parameter
    //============================
    parameter int WIDTH = 4;

    //=====================================================
    // 2) Create a class for constrained-random stimulus
    //=====================================================
    class counter_cfg;

        //====================================
        // DUT signals we want to randomize
        //====================================
        bit             clk;            // clock signal
        rand bit        rst_n;          // Active-low reset
        rand bit        load_n;         // Active-low load
        rand bit        up_down;        // 1 => increment, 0 => decrement
        rand bit        ce;             // clock enable
        rand logic [WIDTH-1:0] data_load;

        //====================
        // Coverage group
        //====================
        covergroup cg @(posedge clk);
          cp1: coverpoint rst_n;
          cp2: coverpoint load_n;
          cp3: coverpoint up_down;
          cp4: coverpoint ce;
          cp5: coverpoint data_load;
        endgroup

        //====================
        // Constructor
        //====================
        function new();
          cg = new();
        endfunction

        //================================================
        // 3) Constraints to meet the 70%/30% guidelines
        // use distribution for probability
        //================================================
        constraint reset_deactivated_most {
          // Reset low 30% of time, high 70%
          rst_n dist { 1 := 70, 0 := 30 };
        }

        constraint load_active_70 {
          // load_n=0 is "active" => 70% of time
          load_n dist { 0 := 70, 1 := 30 };
        }

        constraint enable_active_70 {
          // ce=1 => 70% of time
          ce dist { 1 := 70, 0 := 30 };
        }

        constraint up_down_dist {
          // Distribution constraint: 50% chance of 0, 50% chance of 1
          up_down dist { 0 := 50, 1 := 50 };
        }

        constraint up_down_data_load_c {
              if (up_down) {
```

```
            // up_down=1 => pick data_load mostly in lower half
            data_load dist {
              [0 : (1<<(WIDTH-1))-1] := 80,
              [(1<<(WIDTH-1)) : (1<<WIDTH)-1] := 20
            };
              } else {
            // up_down=0 => pick data_load mostly in upper half
            data_load dist {
              [0 : (1<<(WIDTH-1))-1] := 20,
              [(1<<(WIDTH-1)) : (1<<WIDTH)-1] := 80
            };
              }
        }

    endclass

endpackage
```

2.3 3. 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) begin
        if (up_down)
            count_out <= count_out + 1;
        else
            count_out <= count_out - 1;
    end
end

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

endmodule
```

2.4 4. Bug Fixes

missing (begin and end) for "else if (ce)"

2.5 5. Verification Plan

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
COUNTER_1	When reset (`rst_n`) is asserted, the output `count_out` should be zero.	Directed at the start of the simulation, followed by randomization with a constraint to keep reset active for 30% of the time.	Coverage point to track how many times reset is asserted and confirm `count_out == 0`.	A checker in the testbench ensures `count_out` is 0 when `rst_n == 0`.
COUNTER_2	When load (`load_n`) is asserted, `count_out` should take the value of `data_load`.	Randomization of `load_n` with a 70% probability of being active (low) and randomized `data_load` values.	Coverage point to track how many times `load_n` is asserted and the range of `data_load` values.	A checker in the testbench verifies `count_out == data_load` when `load_n == 0`.
COUNTER_3	When `ce` is enabled, the counter should increment or decrement based on `up_down`.	Randomization with 70% chance for `ce` being high, and a 50-50 distribution for `up_down`.	Coverage point to capture the toggling of `up_down` and transitions of `count_out`.	A checker compares `count_out` with the expected increment or decrement, verified against the golden model.
COUNTER_4	`max_count` should be asserted when `count_out` reaches its maximum value.	Random tests allowing the counter to reach its maximum possible value (`{WIDTH{1'b1}}`).	Coverage point to check how often `max_count` is triggered.	A checker validates `max_count == 1` when `count_out == max value`.
COUNTER_5	`zero` should be asserted when `count_out` is zero.	Directed reset tests and decrement tests pushing `count_out` to zero.	Coverage point for how often `zero` is asserted.	A checker verifies `zero == 1` only when `count_out == 0`.

Table 1: Verification Plan for Counter Design

2.6 6. Do File

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

# to run do file
#-- do run.txt
# to execute coverage report (one for code coverage and other fuctional coverage)
#-- vcover report counter_tb.ucdb -details -annotate -all -output coverage_rpt.txt -du=counter
#-- vcover report -details -cvg -output counter_coverage_report.txt counter_tb.ucdb
```

2.7 7. Coverage Report

```
Coverage Report by DU with details

=================================================================================
=== Design Unit: work.counter
=================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        10        10         0   100.00%

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

Branch Coverage for Design Unit work.counter

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File counter.v
-----------------------------------IF Branch-----------------------------------
    21                                        42     Count coming in to IF
    21              1                         18         if (!rst_n)
    23              1                         15         else if (!load_n)
    25              1                          5         else if (ce) begin
                                               4     All False Count
Branch totals: 4 hits of 4 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    26                                         5     Count coming in to IF
    26              1                          2             if (up_down)
    28              1                          3             else
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    33                                        25     Count coming in to IF
    33              1                          4     assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
    33              2                         21     assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    34                                        25     Count coming in to IF
    34              1                          8     assign zero = (count_out == 0)? 1:0;
    34              2                         17     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 Design Unit work.counter --

  File counter.v
----------------Focused Condition View-------------------
Line       33 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       34 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 Design Unit work.counter —

Line	Item	Count	Source
File counter.v			
8			module counter (clk ,rst_n, load_n, up_down, ce, data_load, count_out, max_count, z
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	42	always @(posedge clk) begin
21			if (!rst_n)
22	1	18	count_out <= 0;
23			else if (!load_n)
24	1	15	count_out <= data_load;
25			else if (ce) begin
26			if (up_down)
27	1	2	count_out <= count_out + 1;
28			else
29	1	3	count_out <= count_out – 1;
30			end
31			end
32			
33	1	26	assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
34	1	26	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 Design Unit work.counter

Node	1H–>0L	0L–>1H	"Coverage"
ce	1	1	100.00
clk	1	1	100.00
count_out[0–3]	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 Design Unit (filtered view): 100.00%

Coverage Report by instance with details

==========

=== Instance: /counter_pkg
=== Design Unit: work.counter_pkg

==========

Covergroup Coverage:

Covergroups	1	na	na	100.00%
Coverpoints/Crosses	5	na	na	na
Covergroup Bins	24	24	0	100.00%

Covergroup	Metric	Goal	Bins	Status

TYPE /counter_pkg/counter_cfg/cg	100.00%	100	–	Covered
covered/total bins:	24	24	–	
missing/total bins:	0	24	–	
% Hit:	100.00%	100	–	
Coverpoint cp1	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	18	1	–	Covered
bin auto[1]	24	1	–	Covered
Coverpoint cp2	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	28	1	–	Covered
bin auto[1]	14	1	–	Covered
Coverpoint cp3	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	26	1	–	Covered
bin auto[1]	16	1	–	Covered
Coverpoint cp4	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	16	1	–	Covered
bin auto[1]	26	1	–	Covered
Coverpoint cp5	100.00%	100	–	Covered
covered/total bins:	16	16	–	
missing/total bins:	0	16	–	
% Hit:	100.00%	100	–	
bin auto[0]	2	1	–	Covered
bin auto[1]	2	1	–	Covered
bin auto[2]	1	1	–	Covered
bin auto[3]	1	1	–	Covered
bin auto[4]	2	1	–	Covered
bin auto[5]	3	1	–	Covered
bin auto[6]	2	1	–	Covered
bin auto[7]	1	1	–	Covered
bin auto[8]	3	1	–	Covered
bin auto[9]	4	1	–	Covered
bin auto[10]	2	1	–	Covered
bin auto[11]	5	1	–	Covered
bin auto[12]	4	1	–	Covered
bin auto[13]	4	1	–	Covered
bin auto[14]	1	1	–	Covered
bin auto[15]	3	1	–	Covered

COVERGROUP COVERAGE:

Covergroup	Metric	Goal	Bins	Status
TYPE /counter_pkg/counter_cfg/cg	100.00%	100	–	Covered
covered/total bins:	24	24	–	
missing/total bins:	0	24	–	
% Hit:	100.00%	100	–	
Coverpoint cp1	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	18	1	–	Covered
bin auto[1]	24	1	–	Covered
Coverpoint cp2	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	28	1	–	Covered
bin auto[1]	14	1	–	Covered
Coverpoint cp3	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	26	1	–	Covered
bin auto[1]	16	1	–	Covered
Coverpoint cp4	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	16	1	–	Covered
bin auto[1]	26	1	–	Covered
Coverpoint cp5	100.00%	100	–	Covered
covered/total bins:	16	16	–	
missing/total bins:	0	16	–	
% Hit:	100.00%	100	–	
bin auto[0]	2	1	–	Covered
bin auto[1]	2	1	–	Covered
bin auto[2]	1	1	–	Covered
bin auto[3]	1	1	–	Covered
bin auto[4]	2	1	–	Covered
bin auto[5]	3	1	–	Covered

bin auto[6]	2	1	–	Covered
bin auto[7]	1	1	–	Covered
bin auto[8]	3	1	–	Covered
bin auto[9]	4	1	–	Covered
bin auto[10]	2	1	–	Covered
bin auto[11]	5	1	–	Covered
bin auto[12]	4	1	–	Covered
bin auto[13]	4	1	–	Covered
bin auto[14]	1	1	–	Covered
bin auto[15]	3	1	–	Covered

TOTAL COVERGROUP COVERAGE: 100.00% COVERGROUP TYPES: 1

Total Coverage By Instance (filtered view): 100.00%

2.8 8.Waveform

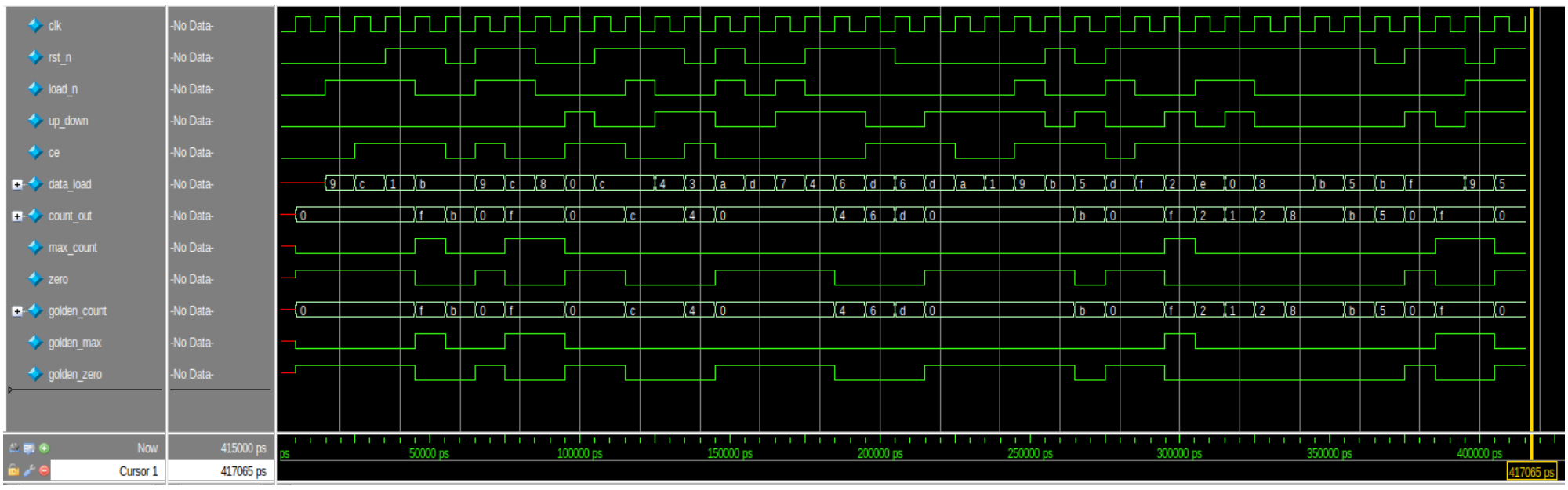

Figure 2: simulation waveform

```
# vsim -voptargs="+acc" work.counter_tb -coverage
# Start time: 17:01:59 on Mar 14,2025
# ** Note: (vsim-8009) Loading existing optimized design _opt1
# Loading sv_std.std
# Loading work.counter_pkg(fast)
# Loading work.counter_tb_sv_unit(fast)
# Loading work.counter_tb(fast)
# Loading work.counter(fast)
# All done. End of simulation.
# ** Note: $finish    : counter_tb.sv(144)
#    Time: 415 ns  Iteration: 1  Instance: /counter_tb
# 1
# Break in Module counter_tb at counter_tb.sv line 144
```

Figure 3: Transcript : all test cases passed

3 Q3: ALU

3.1 1. Testbench code

```
`timescale 1ns/1ps

import ALSU_pkg::*;

module ALSU_tb;

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

  // -----------------------------------
  // DUT instantiation
  // -----------------------------------
  ALSU #(
    .INPUT_PRIORITY("A"),
    .FULL_ADDER("ON")
  ) dut (
    .clk        (clk),
    .rst        (rst),
```

```
    .cin        (cin),
    .red_op_A   (red_op_A),
    .red_op_B   (red_op_B),
    .bypass_A   (bypass_A),
    .bypass_B   (bypass_B),
    .direction  (direction),
    .serial_in  (serial_in),
    .A          (A),
    .B          (B),
    .opcode     (opcode),
    .leds       (leds),
    .out        (out)
);

// ---------------------------------------
// Create an object for random stimulus
// ---------------------------------------
alsu_rand_class stim;

// -----------------------------------
// Clock & reset generation
// -----------------------------------
initial begin
      clk = 0;
      forever begin
         #5 clk = ~clk;
         stim.clk = clk;
      end
 end
// ------------
// reset task
// ------------
task do_reset();
    rst = 1;
    #10;
    // Check result against a golden model
      golden_model(
        rst, A, B, cin, serial_in, red_op_A, red_op_B, bypass_A, bypass_B,
        direction, opcode
      );
    #10;
    rst = 0;
  endtask

  // -----------------------------------
  // Golden model for reference
  // -----------------------------------
task golden_model(
    input logic rst,
    input logic signed [2:0] A, B,
    input logic cin, serial_in, red_op_A, red_op_B,
    input logic bypass_A, bypass_B, direction,
    input logic [2:0] opcode
);

    logic signed [5:0] expected_out;
    logic [15:0] expected_leds;

    logic invalid_red_op, invalid_opcode, invalid;

    // Invalid condition handling
    invalid_red_op = (red_op_A | red_op_B) & (opcode[1] | opcode[2]);
    invalid_opcode = opcode[1] & opcode[2];
    invalid = invalid_red_op | invalid_opcode;

    if(rst) begin
       expected_out = 0;
       expected_leds = 0;
    end else begin
      if (invalid)
        expected_leds = ~expected_leds;
      else
        expected_leds = 0;
    end


    if (bypass_A && bypass_B)
    expected_out = ("A" == "A") ? A : B;  // INPUT_PRIORITY is "A"
    else if (bypass_A)
    expected_out = A;
    else if (bypass_B)
    expected_out = B;
    else if (invalid)
    expected_out = 0;
    else begin
    case (opcode)
        3'h0: begin // OR or Reduction OR
            if (red_op_A && red_op_B)
                expected_out = ("A" == "A") ? |A : |B;
            else if (red_op_A)
                expected_out = |A;
            else if (red_op_B)
                expected_out = |B;
            else
                expected_out = A | B;
        end
        3'h1: begin // XOR or Reduction XOR
            if (red_op_A && red_op_B)
                expected_out = ("A" == "A") ? ^A : ^B;
            else if (red_op_A)
```

```
                expected_out = ^A;
            else if (red_op_B)
                expected_out = ^B;
            else
                expected_out = A ^ B;
        end
        3'h2: expected_out = A + B; // ADD
        3'h3: expected_out = A * B; // MUL
        3'h4: begin // SHIFT
            if (direction)
                expected_out = {expected_out[4:0], serial_in};
            else
                expected_out = {serial_in, expected_out[5:1]};
        end
        3'h5: begin // ROTATE
            if (direction)
                expected_out = {expected_out[4:0], expected_out[5]};
            else
                expected_out = {expected_out[0], expected_out[5:1]};
        end
        default: expected_out = 0;
    endcase
    end

    // Wait another clock so the output is stable
    @(posedge clk);
    #1;

    if ( (out != expected_out) && (leds != expected_leds)) begin
        $error("[ALSU] Mismatch with golden model: opcode=%0b.  out=%0d, expected_out=%0d",
                opcode, out, expected_out);
    end
endtask


  // ------------------------------------
  // Test/Stimulus
  // ------------------------------------
  initial begin

    stim = new();

    do_reset();  // start in reset


    for (int i = 0; i < 200; i++) begin

      // Randomize with constraints
      if(!(stim.randomize())) begin
         $error("Randomization failed!");
         $finish;
      end

      // Drive signals
      rst        = stim.rst;
      cin        = stim.cin;
      red_op_A   = stim.red_op_A;
      red_op_B   = stim.red_op_B;
      bypass_A   = stim.bypass_A;
      bypass_B   = stim.bypass_B;
      direction  = stim.direction;
      serial_in  = stim.serial_in;
      opcode     = stim.opcode;
      A          = stim.A;
      B          = stim.B;

      // Wait a clock for inputs to be sampled
      @(posedge clk);

      // Check result against a golden model
      golden_model(
        rst, A, B, cin, serial_in, red_op_A, red_op_B, bypass_A, bypass_B,
        direction, opcode
      );

    end

    $display("ALSU test completed");
    $finish;
  end

endmodule
```

3.2 2. Package code

```
package ALSU_pkg;
  // ----------------------------------------
  // 1) Define the opcodes (including invalids)
  // ----------------------------------------
  typedef enum logic [2:0] {
    OR_0       = 3'h0,  // 000
    XOR_1      = 3'h1,  // 001
    ADD_2      = 3'h2,  // 010
    MUL_3      = 3'h3,  // 011
    SHIFT_4    = 3'h4,  // 100
    ROTATE_5   = 3'h5,  // 101
    INVALID_6  = 3'h6,  // 110
    INVALID_7  = 3'h7   // 111
  } opcode_e;

```

```
// ----------------------------------------
// 2) 3-bit signed range is -4 .. +3
// ----------------------------------------
localparam logic signed [2:0] MAXNEG = -4;  // 3'b100
localparam logic signed [2:0] ZERO   =  0;
localparam logic signed [2:0] MAXPOS =  3;  // 3'b011

  class alsu_rand_class;
    // -------------------------------
    // Randomizable DUT inputs
    // -------------------------------
    bit                     clk;
    rand bit                rst;
    rand bit                cin;
    rand bit                red_op_A;
    rand bit                red_op_B;
    rand bit                bypass_A;
    rand bit                bypass_B;
    rand bit                direction;
    rand bit                serial_in;
    rand opcode_e           opcode;
    rand logic signed [2:0] A;
    rand logic signed [2:0] B;


    // -------------------------------
    // Constraints from specification
    // -------------------------------

    // (a) Make RESET happen with a low probability
    constraint c_reset_low_prob {
      rst dist { 0 := 95, 1 := 5 };
    }

    // (b) For ADD or MUL, pick corner values of A,B more often
    //     (MAXNEG, ZERO, MAXPOS) than the other possibilities.
    //     Weighted distribution is used here.
    constraint c_adder_mult_corner {
       if (opcode inside {ADD_2, MUL_3}) {
           A dist { MAXNEG := 5, ZERO := 5, MAXPOS := 5, [-3:-1] := 1, [1:2] := 1 };
           B dist { MAXNEG := 5, ZERO := 5, MAXPOS := 5, [-3:-1] := 1, [1:2] := 1 };
       }
    }


    // (c) If opcode=OR or XOR and red_op_A=1, then A has exactly one bit set
    //     and B is 0 .
    constraint c_red_opA_onebit {
      if ((opcode==OR_0 || opcode==XOR_1) && red_op_A==1) {
        // Force B to be 0 or near 0
        B == 0;
        // A has exactly 1 bit set in its 3 bits:
        A dist {1:=5, 2:=5, MAXNEG:=5 , MAXPOS := 1 , [-3:0] := 1};
      }
    }

    // (d) Similarly, if opcode=OR or XOR and red_op_B=1, then B has exactly one bit set
    //     and A is 0.
    constraint c_red_opB_onebit {
      if ((opcode==OR_0 || opcode==XOR_1) && red_op_B==1) {
        A == 0;
        B dist {1:=5, 2:=5, MAXNEG:=5 , MAXPOS := 1 , [-3:0]:= 1 };
      }
    }

    // (e) Invalid cases (opcode=6 or 7, or red_op_X=1 for non-OR/XOR)
    //     should occur *less* frequently.
    //     Weighted distribution on opcode:
    constraint c_opcode_distribution {
      opcode dist {
          INVALID_6  := 1,
          INVALID_7  := 1,
          OR_0       := 5,
          XOR_1      := 5,
          ADD_2      := 5,
          MUL_3      := 5,
          SHIFT_4    := 5,
          ROTATE_5   := 5
      };
    }

    // (f) For red_op_A/B, require them to be 0 if opcode in {ADD_2, MUL_3, SHIFT_4, ROTATE_5}
    // except for a small chance to produce the invalid scenario:
    constraint c_red_op_non_orxor {
      if (opcode inside {ADD_2, MUL_3, SHIFT_4, ROTATE_5}) {
        (red_op_A == 0) dist {0:=95, 1:=5};
        (red_op_B == 0) dist {0:=95, 1:=5};
      }
    }

    constraint c_red_op_orxor {
      if (opcode inside {XOR_1, OR_0}) {
            {red_op_A, red_op_B} dist {2'b00 :/ 5, 2'b01 :/ 10, 2'b10 :/ 10 ,2'b11 :/ 75};
      }
    }

    // (g) bypass_A and bypass_B should be disabled most of the time
    constraint c_bypass_dist {
        bypass_A dist {0:=3,1:=1};
        bypass_B dist {0:=3,1:=1};
```

```
    }

    // (h) If SHIFT or ROTATE, do not constrain A,B.
    //     (No explicit constraint needed => they can be anything.)




    // -------------------------------------
    // Coverage points
    // -------------------------------------
    covergroup cg @(posedge clk);
      coverpoint rst;
      coverpoint cin;
      coverpoint red_op_A;
      coverpoint red_op_B;
      coverpoint bypass_A;
      coverpoint bypass_B;
      coverpoint opcode;
          coverpoint direction;
          coverpoint serial_in;
          coverpoint A;
          coverpoint B;
    endgroup

        // constructor
    function new();
      cg = new();
    endfunction


endclass

endpackage
```

3.3 3. Design code

```
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, cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
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 cin_reg;
reg [2:0] opcode_reg;
reg signed [2:0] 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;
  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
```

```
63    if(rst) begin
64      out <= 0;
65    end
66    else begin
67      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 if (invalid)
74          out <= 0;
75      else begin
76          case (opcode)
77            3'h0: begin
78              if (red_op_A_reg && red_op_B_reg)
79                out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
80              else if (red_op_A_reg)
81                out <= |A_reg;
82              else if (red_op_B_reg)
83                out <= |B_reg;
84              else
85                out <= A_reg | B_reg;
86            end
87            3'h1: begin
88              if (red_op_A_reg && red_op_B_reg)
89                out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
90              else if (red_op_A_reg)
91                out <= ^A_reg;
92              else if (red_op_B_reg)
93                out <= ^B_reg;
94              else
95                out <= A_reg ^ B_reg;
96            end
97            3'h2: out <= A_reg + B_reg;
98            3'h3: out <= A_reg * B_reg;
99            3'h4: begin
100             if (direction_reg)
101               out <= {out[4:0], serial_in_reg};
102             else
103               out <= {serial_in_reg, out[5:1]};
104           end
105           3'h5: begin
106             if (direction_reg)
107               out <= {out[4:0], out[5]};
108             else
109               out <= {out[0], out[5:1]};
110           end
111         endcase
112     end
113   end
114 end
115
116 endmodule
```

3.4 4. Bug Fixes

no bugs except cin_reg is one bit not two bits

3.5 5. Verification Plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
ALSU_1	When the reset is asserted, the outputs should be low.	- Directed reset at the start of simulation via do_reset().- Randomized afterward using constraint c_reset_low_prob (5% chance of reset=1).	- Covergroup in alsu_rand_class tracks rst transitions. - Ensures we observe enough reset assertions for coverage.	- The testbench checks that when rst is high, out is driven to 0 and leds is 0 (or blinking is suppressed). - Invokes golden_model() to confirm.
ALSU_2	In the absence of invalid cases, when opcode is ADD, the output should perform addition on ports A and B, taking cin if FULL_ADDER is on.	- Randomize opcode under c_opcode_distribution so that ADD (3'h2) appears frequently. - Corner-case inputs for A and B under c_adder_mult_corner (favor -4,0,3). - red_op_A and red_op_B are mostly 0 to avoid invalid conditions.	- The coverage group tracks opcode, A, B, and cin to ensure corner cases (MAXNEG, ZERO, MAXPOS) are exercised. - Also measures how often ADD vs. other operations appear.	- After each random transaction, the testbench calls golden_model() to confirm out == A + B (plus cin if relevant). - Errors are flagged if out differs from the expected sum.

Table 2: Verification Plan

3.6 6. Do File

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

# to run do file
-- do run.txt
to execute coverage report
-- vcover report ALSU_tb.ucdb -details -annotate -all -output coverage_rpt.txt -du=ALSU
-- vcover report -details -cvg -output ALSU_coverage_report.txt ALSU_tb.ucdb
```

3.7 7. Coverage Report

```
Coverage Report by DU with details

================================================================================
=== Design Unit: work.ALSU
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        32        32         0   100.00%

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

Branch Coverage for Design Unit work.ALSU

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File ALSU.v
----------------------------------IF Branch------------------------------------
    24                                       402     Count coming in to IF
    24              1                         28       if(rst) begin
    35              1                        374       end else begin
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------IF Branch------------------------------------
    51                                       417     Count coming in to IF
    51              1                         43       if(rst) begin
    53              1                        374       end else begin
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------IF Branch------------------------------------
    54                                       374     Count coming in to IF
    54              1                        279           if (invalid)
    56              1                         95           else
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------IF Branch------------------------------------
    63                                       384     Count coming in to IF
    63              1                         28       if(rst) begin
    66              1                        356       else begin
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------IF Branch------------------------------------
    67                                       356     Count coming in to IF
    67              1                         15         if (bypass_A_reg && bypass_B_reg)
    69              1                         77         else if (bypass_A_reg)
    71              1                         46         else if (bypass_B_reg)
    73              1                        156         else if (invalid)
    75              1                         62         else begin
Branch totals: 5 hits of 5 branches = 100.00%

---------------------------------CASE Branch-----------------------------------
    76                                        62     Count coming in to CASE
    77              1                         17              3'h0: begin
    87              1                         18              3'h1: begin
    97              1                          6              3'h2: out <= A_reg + B_reg;
    98              1                          7              3'h3: out <= A_reg * B_reg;
    99              1                          5              3'h4: begin
    105             1                          8              3'h5: begin
                                               1     All False Count
Branch totals: 7 hits of 7 branches = 100.00%

----------------------------------IF Branch------------------------------------
    78                                        17     Count coming in to IF
    78              1                          3               if (red_op_A_reg && red_op_B_reg)
    80              1                          5               else if (red_op_A_reg)
    82              1                          2               else if (red_op_B_reg)
    84              1                          7               else
Branch totals: 4 hits of 4 branches = 100.00%

----------------------------------IF Branch------------------------------------
    88                                        18     Count coming in to IF
    88              1                          1               if (red_op_A_reg && red_op_B_reg)
    90              1                          5               else if (red_op_A_reg)
    92              1                          5               else if (red_op_B_reg)
    94              1                          7               else
```

```
Branch totals: 4 hits of 4 branches = 100.00%

------------------------------------IF Branch------------------------------------
    100                                     5     Count coming in to IF
    100             1                       1             if (direction_reg)
    102             1                       4             else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    106                                     8     Count coming in to IF
    106             1                       2             if (direction_reg)
    108             1                       6             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 Design Unit work.ALSU —

  File ALSU.v
----------------Focused Condition View----------------
Line       67 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       78 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       88 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 Design Unit work.ALSU —

  File ALSU.v
----------------Focused Expression View----------------
Line       18 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 19 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 20 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%

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

Statement Coverage for Design Unit work.ALSU —

Line	Item	Count	Source
File ALSU.v			
1			module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, c
2			parameter INPUT_PRIORITY = "A";
3			parameter FULL_ADDER = "ON";
4			input clk, cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
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 red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_ir
11			reg signed cin_reg;
12			reg [2:0] opcode_reg;
13			reg signed [2:0] A_reg, B_reg;
14			
15			wire invalid_red_op, invalid_opcode, invalid;
16			
17			//Invalid handling
18	1	177	assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg
19	1	167	assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
20	1	89	assign invalid = invalid_red_op | invalid_opcode;
21			
22			//Registering input signals
23	1	402	always @(posedge clk or posedge rst) begin
24			if(rst) begin
25	1	28	cin_reg <= 0;
26	1	28	red_op_B_reg <= 0;
27	1	28	red_op_A_reg <= 0;
28	1	28	bypass_B_reg <= 0;
29	1	28	bypass_A_reg <= 0;
30	1	28	direction_reg <= 0;
31	1	28	serial_in_reg <= 0;
32	1	28	opcode_reg <= 0;
33	1	28	A_reg <= 0;

```
34          1                28          B_reg <= 0;
35                                    end else begin
36          1               374          cin_reg <= cin;
37          1               374          red_op_B_reg <= red_op_B;
38          1               374          red_op_A_reg <= red_op_A;
39          1               374          bypass_B_reg <= bypass_B;
40          1               374          bypass_A_reg <= bypass_A;
41          1               374          direction_reg <= direction;
42          1               374          serial_in_reg <= serial_in;
43          1               374          opcode_reg <= opcode;
44          1               374          A_reg <= A;
45          1               374          B_reg <= B;
46                                    end
47                                  end
48
49                                  //leds output blinking
50          1               417     always @(posedge clk or posedge rst) begin
51                                    if(rst) begin
52          1                43          leds <= 0;
53                                    end else begin
54                                        if (invalid)
55          1               279           leds <= ~leds;
56                                        else
57          1                95           leds <= 0;
58                                    end
59                                  end
60
61                                  //ALSU output processing
62          1               384     always @(posedge clk or posedge rst) begin
63                                    if(rst) begin
64          1                28        out <= 0;
65                                    end
66                                    else begin
67                                      if (bypass_A_reg && bypass_B_reg)
68          1                15          out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
69                                      else if (bypass_A_reg)
70          1                77          out <= A_reg;
71                                      else if (bypass_B_reg)
72          1                46          out <= B_reg;
73                                      else if (invalid)
74          1               156            out <= 0;
75                                      else begin
76                                          case (opcode)
77                                            3'h0: begin
78                                              if (red_op_A_reg && red_op_B_reg)
79          1                 3               out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
80                                              else if (red_op_A_reg)
81          1                 5               out <= |A_reg;
82                                              else if (red_op_B_reg)
83          1                 2               out <= |B_reg;
84                                              else
85          1                 7               out <= A_reg | B_reg;
86                                            end
87                                            3'h1: begin
88                                              if (red_op_A_reg && red_op_B_reg)
89          1                 1               out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
90                                              else if (red_op_A_reg)
91          1                 5               out <= ^A_reg;
92                                              else if (red_op_B_reg)
93          1                 5               out <= ^B_reg;
94                                              else
95          1                 7               out <= A_reg ^ B_reg;
96                                            end
97          1                 6            3'h2: out <= A_reg + B_reg;
98          1                 7            3'h3: out <= A_reg * B_reg;
99                                            3'h4: begin
100                                             if (direction_reg)
101         1                 1               out <= {out[4:0], serial_in_reg};
102                                             else
103         1                 4               out <= {serial_in_reg, out[5:1]};
104                                           end
105                                           3'h5: begin
106                                             if (direction_reg)
107         1                 2               out <= {out[4:0], out[5]};
108                                             else
109         1                 6               out <= {out[0], out[5:1]};
```

Toggle Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
Toggles	118	118	0	100.00%

Toggle Details

Toggle Coverage for Design Unit work.ALSU

Node	1H–>0L	0L–>1H	"Coverage"
A[0–2]	1	1	100.00
A_reg[0–2]	1	1	100.00
B[0–2]	1	1	100.00
B_reg[0–2]	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	1	1	100.00
cin_reg	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[0-15]	1	1	100.00
opcode[0-2]	1	1	100.00
opcode_reg[0-2]	1	1	100.00
out[0-5]	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 = 59
Toggled Node Count = 59
Untoggled Node Count = 0

Toggle Coverage = 100.00% (118 of 118 bins)

Total Coverage By Design Unit (filtered view): 100.00%

Coverage Report by instance with details

==========

=== Instance: /ALSU_pkg
=== Design Unit: work.ALSU_pkg

==========

Covergroup Coverage:

Covergroups	1	na	na	100.00%
Coverpoints/Crosses	11	na	na	na
Covergroup Bins	40	40	0	100.00%

Covergroup	Metric	Goal	Bins	Status
TYPE /ALSU_pkg/alsu_rand_class/cg	100.00%	100	–	Covered
covered/total bins:	40	40	–	
missing/total bins:	0	40	–	
% Hit:	100.00%	100	–	
Coverpoint rst	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	377	1	–	Covered
bin auto[1]	26	1	–	Covered
Coverpoint cin	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	207	1	–	Covered
bin auto[1]	196	1	–	Covered
Coverpoint red_op_A	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	81	1	–	Covered
bin auto[1]	322	1	–	Covered
Coverpoint red_op_B	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	71	1	–	Covered
bin auto[1]	332	1	–	Covered
Coverpoint bypass_A	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	297	1	–	Covered
bin auto[1]	106	1	–	Covered
Coverpoint bypass_B	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	325	1	–	Covered
bin auto[1]	78	1	–	Covered
Coverpoint opcode	100.00%	100	–	Covered
covered/total bins:	8	8	–	
missing/total bins:	0	8	–	
% Hit:	100.00%	100	–	

bin auto[OR_0]	50	1	–	Covered
bin auto[XOR_1]	40	1	–	Covered
bin auto[ADD_2]	66	1	–	Covered
bin auto[MUL_3]	92	1	–	Covered
bin auto[SHIFT_4]	62	1	–	Covered
bin auto[ROTATE_5]	70	1	–	Covered
bin auto[INVALID_6]	10	1	–	Covered
bin auto[INVALID_7]	10	1	–	Covered
Coverpoint direction	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	227	1	–	Covered
bin auto[1]	176	1	–	Covered
Coverpoint serial_in	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	201	1	–	Covered
bin auto[1]	202	1	–	Covered
Coverpoint A	100.00%	100	–	Covered
covered/total bins:	8	8	–	
missing/total bins:	0	8	–	
% Hit:	100.00%	100	–	
bin auto[−4]	74	1	–	Covered
bin auto[−3]	16	1	–	Covered
bin auto[−2]	30	1	–	Covered
bin auto[−1]	30	1	–	Covered
bin auto[0]	110	1	–	Covered
bin auto[1]	36	1	–	Covered
bin auto[2]	38	1	–	Covered
bin auto[3]	66	1	–	Covered
Coverpoint B	100.00%	100	–	Covered
covered/total bins:	8	8	–	
missing/total bins:	0	8	–	
% Hit:	100.00%	100	–	
bin auto[−4]	68	1	–	Covered
bin auto[−3]	26	1	–	Covered
bin auto[−2]	32	1	–	Covered
bin auto[−1]	24	1	–	Covered
bin auto[0]	102	1	–	Covered
bin auto[1]	48	1	–	Covered
bin auto[2]	34	1	–	Covered
bin auto[3]	66	1	–	Covered

COVERGROUP COVERAGE:

Covergroup	Metric	Goal	Bins	Status
TYPE /ALSU_pkg/alsu_rand_class/cg	100.00%	100	–	Covered
covered/total bins:	40	40	–	
missing/total bins:	0	40	–	
% Hit:	100.00%	100	–	
Coverpoint rst	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	377	1	–	Covered
bin auto[1]	26	1	–	Covered
Coverpoint cin	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	207	1	–	Covered
bin auto[1]	196	1	–	Covered
Coverpoint red_op_A	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	81	1	–	Covered
bin auto[1]	322	1	–	Covered
Coverpoint red_op_B	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	71	1	–	Covered
bin auto[1]	332	1	–	Covered
Coverpoint bypass_A	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	297	1	–	Covered
bin auto[1]	106	1	–	Covered
Coverpoint bypass_B	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	325	1	–	Covered
bin auto[1]	78	1	–	Covered
Coverpoint opcode	100.00%	100	–	Covered
covered/total bins:	8	8	–	

missing/total bins:	0	8	–	
% Hit:	100.00%	100	–	
bin auto[OR_0]	50	1	–	Covered
bin auto[XOR_1]	40	1	–	Covered
bin auto[ADD_2]	66	1	–	Covered
bin auto[MUL_3]	92	1	–	Covered
bin auto[SHIFT_4]	62	1	–	Covered
bin auto[ROTATE_5]	70	1	–	Covered
bin auto[INVALID_6]	10	1	–	Covered
bin auto[INVALID_7]	10	1	–	Covered
Coverpoint direction	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	227	1	–	Covered
bin auto[1]	176	1	–	Covered
Coverpoint serial_in	100.00%	100	–	Covered
covered/total bins:	2	2	–	
missing/total bins:	0	2	–	
% Hit:	100.00%	100	–	
bin auto[0]	201	1	–	Covered
bin auto[1]	202	1	–	Covered
Coverpoint A	100.00%	100	–	Covered
covered/total bins:	8	8	–	
missing/total bins:	0	8	–	
% Hit:	100.00%	100	–	
bin auto[−4]	74	1	–	Covered
bin auto[−3]	16	1	–	Covered
bin auto[−2]	30	1	–	Covered
bin auto[−1]	30	1	–	Covered
bin auto[0]	110	1	–	Covered
bin auto[1]	36	1	–	Covered
bin auto[2]	38	1	–	Covered
bin auto[3]	66	1	–	Covered
Coverpoint B	100.00%	100	–	Covered
covered/total bins:	8	8	–	
missing/total bins:	0	8	–	
% Hit:	100.00%	100	–	
bin auto[−4]	68	1	–	Covered
bin auto[−3]	26	1	–	Covered
bin auto[−2]	32	1	–	Covered
bin auto[−1]	24	1	–	Covered
bin auto[0]	102	1	–	Covered
bin auto[1]	48	1	–	Covered
bin auto[2]	34	1	–	Covered
bin auto[3]	66	1	–	Covered

TOTAL COVERGROUP COVERAGE: 100.00% COVERGROUP TYPES: 1

Total Coverage By Instance (filtered view): 100.00%

3.8 8.Waveform

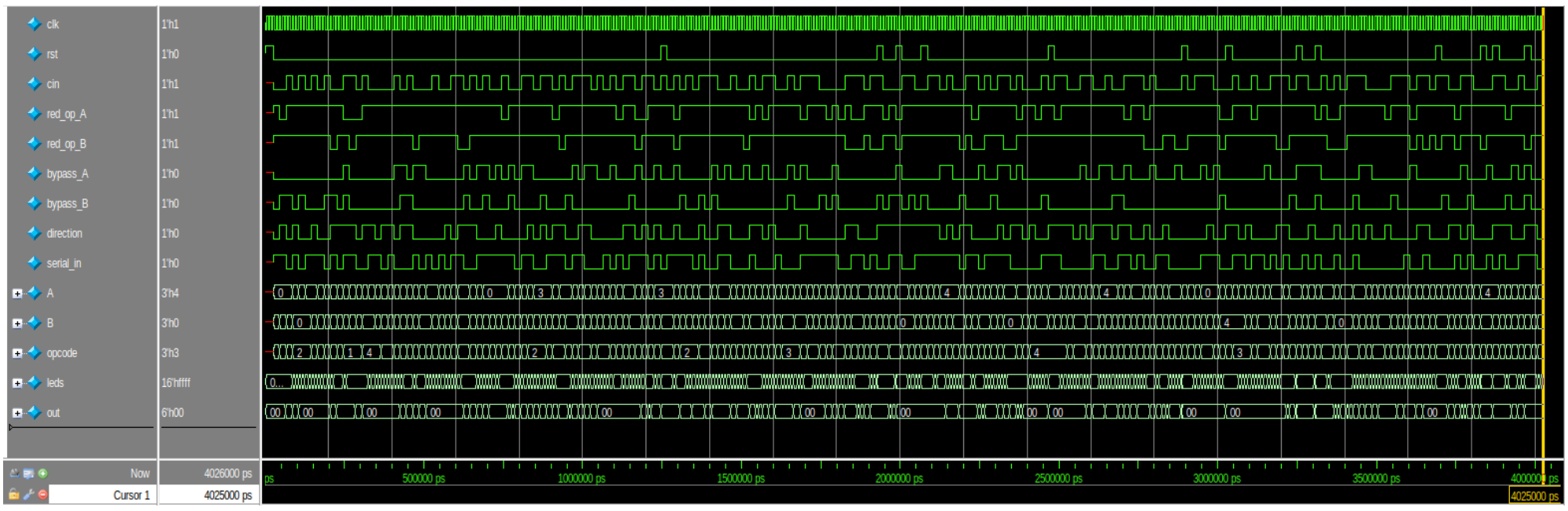

Figure 4: simulation waveform

```
# Top level modules:
#       ALSU_tb
# End time: 17:34:45 on Mar 15,2025, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# vsim -voptargs="+acc" work.ALSU_tb -coverage
# Start time: 17:34:45 on Mar 15,2025
# ** Note: (vsim-8009) Loading existing optimized design _opt
# Loading sv_std.std
# Loading work.ALSU_pkg(fast)
# Loading work.ALSU_tb_sv_unit(fast)
# Loading work.ALSU_tb(fast)
# Loading work.ALSU(fast)
# ALSU test completed
# ** Note: $finish    : ALSU_tb.sv(212)
#    Time: 4026 ns  Iteration: 0  Instance: /ALSU_tb
# 1
# Break in Module ALSU_tb at ALSU_tb.sv line 212
```

Figure 5: Transcript : all test cases passed