1.

## 1.1. Testbench file:

```
module dynamic_array_tb;
    // Declare dynamic arrays
    int dyn_arr1[];
    int dyn_arr2[];

    initial begin
        // initialize dyn_arr2 array elements with (9,1,8,3,4,4)
        dyn_arr2 = '{9,1,8,3,4,4};

        // Allocate 6 elements in dyn_arr1 using new
        dyn_arr1 = new[6];

        // Initialize dyn_arr1 with its index as value using foreach
        foreach (dyn_arr1[i]) begin
            dyn_arr1[i] = i;
        end

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

        // Delete dyn_arr1
        dyn_arr1.delete();

        // Display dyn_arr2 before any operations
        // $display("Original dyn_arr2: %p", dyn_arr2);

        // Reverse dyn_arr2
        dyn_arr2.reverse();
        $display("Reversed dyn_arr2: %p", dyn_arr2);

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

        // Reverse sort (descending)
        dyn_arr2.rsort();
        $display("Reverse Sorted dyn_arr2: %p", dyn_arr2);

        // Shuffle dyn_arr2
        dyn_arr2.shuffle();
        $display("Shuffled dyn_arr2: %p", dyn_arr2);
    end
endmodule
```

## 1.2. Transcript:

```
dyn_arr1: '{0, 1, 2, 3, 4, 5}
dyn_arr1 size: 6
Reversed dyn_arr2: '{4, 4, 3, 8, 1, 9}
Sorted dyn_arr2: '{1, 3, 4, 4, 8, 9}
Reverse Sorted dyn_arr2: '{9, 8, 4, 4, 3, 1}
Shuffled dyn_arr2: '{8, 1, 4, 3, 9, 4}
```

2.

## 2.1. Design file:

```
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
```

## 2.2. Verification plan:

| Label | Design Requirement Description | Stimulus Generation | Functional Coverage | Functionality Check |
|---|---|---|---|---|
| COUNTER1 | When rst_n is asserted low, count_out should be 0 and zero should be 1, max_count = 0. | Directed at the start of simulation | - | Checker ensures count_out = 0, zero = 1, max_count = 0. |
| COUNTER2 | When load_n is low, data_load should be loaded into count_out. | Directed + random valid data_load | - | Checker verifies count_out == data_load when load_n == 0. |
| COUNTER3 | When ce = 1, up_down = 1, counter increments each clock cycle. | Directed increment loop | - | Checker validates correct counting sequence and max_count assertion at max. |
| COUNTER4 | When ce = 1, up_down = 0, counter decrements each clock cycle. | Directed decrement loop | - | Checker validates correct counting sequence and zero assertion at 0. |
| COUNTER5 | If ce = 0, counter must hold its value regardless of up_down. | Directed with ce = 0 | - | Checker ensures count_out is stable when ce == 0. |
| COUNTER6 | When incrementing reaches 2^WIDTH - 1, max_count should be 1. | Directed + edge condition testing | - | Checker asserts max_count == 1 only at count_out == max. |
| COUNTER7 | When decrementing reaches 0, zero should be 1. | Directed + edge condition testing | - | Checker asserts zero == 1 only at count_out == 0. |
| COUNTER8 | Verify counter behavior under random sequences of load_n, up_down, and ce. | Constrained random tests | - | Scoreboard checks behavior against expected model. |

| Label | Design Requirement Description | Stimulus Generation | Functional Coverage | Functionality Check |
|---|---|---|---|---|
| COUNTER9 | Ensure correct priority of load_n over ce. | Mixed scenario tests | - | Checker confirms load_n overrides counting when active. |

## 2.3. Package file:

```
package counter_pkg;

  // Parameter for data width
  parameter int WIDTH = 4;

  // Class to generate constrained stimulus for counter
  class counter_txn;

    // Signals (inputs to DUT)
    rand bit rst_n;
    rand bit load_n;
    rand bit ce;
    rand bit up_down;
    rand bit [WIDTH - 1:0] data_load;

    constraint c_rst_n {
        rst_n dist {0 := 5, 1 := 95}; // 5% active (0), 95% inactive (1)
    }

    constraint c_load_n {
        load_n dist {0 := 70, 1 := 30}; // 70% active (0), 30% inactive (1)
    }

    constraint c_ce {
        ce dist {0 := 30, 1 := 70}; // 30% active (0), 70% inactive (1)
    }

    constraint c_up_down {
      up_down dist {0 := 50, 1 := 50}; // 50% active (0), 50% inactive (1)
    }

    function void reset (rst_n = 1, load_n = 1, ce = 0, up_down = 0, data_load
= 0);
        this.rst_n    = rst_n;
        this.load_n   = load_n;
        this.ce       = ce;
        this.up_down  = up_down;
        this.data_load = data_load;
    endfunction

  endclass : counter_txn

endpackage : counter_pkg
```

## 2.4. Testbench file:

```
import counter_pkg::*;

module counter_tb;

    // TESTBENCH VARIABLES
    int local_error_count = 0;
    int error_count = 0;
    int pass_count  = 0;

    // DUT
    logic clk;
    logic rst_n;
    logic load_n;
    logic up_down;
    logic ce;
    logic [WIDTH - 1:0] data_load;
    logic [WIDTH - 1:0] count_out;
    logic max_count;
    logic zero;

    // GOLDEN MODEL
    logic [WIDTH - 1:0] golden_count_out;
    logic golden_max_count;
    logic golden_zero;

    counter #(.WIDTH(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)
    );

    counter_txn txn;

    always #5 clk = ~clk;

    task wait_cycles(input int num_cycles);
        repeat (num_cycles) @(negedge clk);
    endtask

    task assert_reset();
```

```
        rst_n = 0;
    endtask

    task deassert_reset();
        rst_n = 1;
    endtask


    task golden_model();
        if (!rst_n) begin
            golden_count_out = 0;
        end else if (!load_n) begin
            golden_count_out = data_load;
        end else if (ce) begin
            if (up_down) begin
                golden_count_out = golden_count_out + 1;
            end else begin
                golden_count_out = golden_count_out - 1;
            end
        end
        golden_max_count = (golden_count_out == {WIDTH{1'b1}});
        golden_zero = (golden_count_out == 0);
    endtask

    task check_result();
        local_error_count = 0;

        if (count_out !== golden_count_out) begin
            $error("[ERROR] count_out mismatch. Expected: %0d, Got: %0d",
golden_count_out, count_out);
            local_error_count++;
        end

        if (max_count !== golden_max_count) begin
            $error("[ERROR] max_count mismatch. Expected: %b, Got: %b",
golden_max_count, max_count);
            local_error_count++;
        end

        if (zero !== golden_zero) begin
            $error("[ERROR] zero mismatch. Expected: %b, Got: %b",
golden_zero, zero);
            local_error_count++;
        end

        if (local_error_count == 0) begin
            pass_count++;
            $display("[PASS] Outputs match expected values.");
        end else begin
```

```
            error_count++;
            $display("[FAIL] Total mismatches in this check: %0d",
local_error_count);
        end
    endtask


    initial begin
        $display("Starting counter testbench...");
        clk = 0;
        rst_n = 1;
        txn = new();

        // COUNTER1
        $display("\nCOUNTER1\n");
        assert_reset();
        golden_model();
        wait_cycles(1);
        deassert_reset();
        check_result();

        // COUNTER2
        $display("\nCOUNTER2\n");
        repeat (10) begin
            assert (txn.randomize(load_n, data_load));
            drive_inputs();
            golden_model();
            wait_cycles(1);
            if (~load_n) begin
                check_result();
            end
        end
        txn.reset();

        // COUNTER3 & COUNTER4
        $display("\nCOUNTER3 & COUNTER4\n");
        repeat (10) begin
            assert (txn.randomize(ce, up_down));
            drive_inputs();
            golden_model();
            wait_cycles(1);
            if (ce) begin
                check_result();
            end
        end
        txn.reset();

        // COUNTER5
        $display("\nCOUNTER5\n");
```

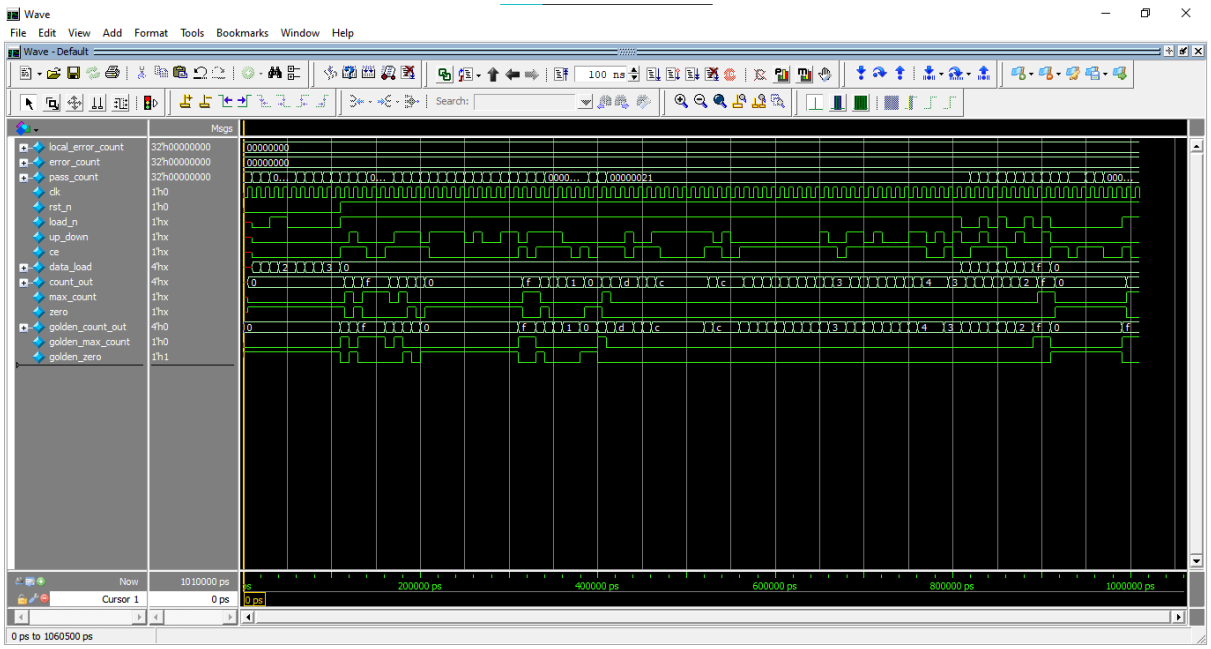
```
repeat (10) begin
    assert (txn.randomize(up_down));
    drive_inputs();
    golden_model();
    wait_cycles(1);
    check_result();
end
txn.reset();

// COUNTER6 & COUNTER7
$display("\nCOUNTER6 & COUNTER7\n");
repeat (50) begin
    assert (txn.randomize(ce, up_down));
    drive_inputs();
    golden_model();
    wait_cycles(1);
    if (max_count) begin
        check_result();
    end
    if (zero) begin
        check_result();
    end
end
txn.reset();

// COUNTER8
$display("\nCOUNTER8\n");
repeat (10) begin
    assert (txn.randomize());
    drive_inputs();
    golden_model();
    wait_cycles(1);
    check_result();
end
txn.reset();

// COUNTER9
$display("\nCOUNTER9\n");
repeat (10) begin
    assert (txn.randomize(load_n, ce));
    drive_inputs();
    golden_model();
    wait_cycles(1);
    if (~load_n & ce) begin
        check_result();
    end
end
```

```
        // Display completion message
        $display("Simulation Completed: %0d test cases executed.", pass_count
+ error_count);
        $display("Test Summary: Passed = %0d, Failed = %0d", pass_count,
error_count);
        $stop;
    end

    task drive_inputs();
        rst_n     = txn.rst_n;
        load_n    = txn.load_n;
        ce        = txn.ce;
        up_down   = txn.up_down;
        data_load = txn.data_load;
    endtask

endmodule
```

## 2.5. Do file:

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

## 2.6. Waveform snippet:




## 2.7. Transcript:

```
COUNTER1
[PASS] Outputs match expected values.
.
.
.
COUNTER9
[PASS] Outputs match expected values.

Simulation Completed: 48 test cases executed.
Test Summary: Passed = 48, Failed = 0
```

## 2.8. Bugs:

```
No bugs detected.
```

## 2.9. Coverage report:

```
> vcover report counter_tb.ucdb -details -annotate -all -output
coverage_rpt.txt

================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        10        10         0   100.00%

================================================================================
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       7         7         0   100.00%

================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         30        30         0   100.00%
```

3.

## 3.1. Design file:

```
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 [1:0] 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
  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)
          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 <= 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
```

## 3.2. Verification plan:

| Label | Design Requirement Description | Stimulus Generation | Functional Coverage | Functionality Check |
|---|---|---|---|---|
| ALSU1 | When rst is asserted high, out should be 0 and leds should be 0. | Directed at the start of simulation then randomized with constraint (most of the time low). | - | Checker ensures out = 0, leds = 0. |
| ALSU2 | when opcode is ADD, then out should perform addition on ports A and B taking cin. | Randomized under constraints on the A and B | - | Checker ensures out = A + B + cin |
| ALSU3 | when opcode is MUL, then out should perform the multiplication on ports A and B | Randomized under constraints on the A and B | - | Checker ensures out = A * B |
| ALSU4 | when opcode is OR, then out should perform the OR operation on ports A and B if reduction_A, reduction B are low | Randomized without constraints on A or B. | - | Checker ensures out = A \| B |
| ALSU5 | when opcode is OR & any of the inputs reduction_A or reduction_B is high. | Randomized under constraints on the A and B | - | Checker ensures out = (INPUT_PRIORITY == ‘A’) ? \|A : \|B |
| ALSU6 | when opcode is XOR, then out should perform the XOR operation on ports A and B if reduction_A, reduction B are low | Randomized without constraints on A or B. | - | Checker ensures out = A ^ B |
| ALSU7 | when opcode is XOR & any of the inputs reduction_A or reduction_B is high. | Randomized under constraints on the A and B | - | Checker ensures out = (INPUT_PRIORITY == ‘A’) ? ^A : ^B |

| Label | Design Requirement Description | Stimulus Generation | Functional Coverage | Functionality Check |
|---|---|---|---|---|
| ALSU8 | when opcode is SHIFT then the output will shift right or left based on the direction input | Randomized without constraints on A or B. | - | Checker ensures out = (Direction == ‘left’) ? out << : out >> |
| ALSU9 | when opcode is ROTATE then the output will rotate right or left based on the direction input | Randomized without constraints on A or B. | - | Checker ensures out = (Direction == ‘left’) ? out << : out >> |
| ALSU10 | When invalid cases exist without bypass, out should be low and leds should blink | Randomized under constraints (valid cases happen frequently) | - | Checker ensures out = 0, leds blinking |
| ALSU11 | If the bypass inputs are high, then the output will bypass port A or B | Randomized under constraints | - | Checker ensures out = A or B |

## 3.3. Package file:

```
package alsu_pkg;

    typedef enum logic [2:0] {OR, XOR, ADD, MUL, SHIFT, ROTATE, INVALID_6,
INVALID_7} opcode_e;

    typedef enum bit [2:0] {
        MAXNEG = 3'b100,
        ZERO = 3'b000,
        MAXPOS = 3'b011
    } input_val_e;

    class alsu_inputs;
        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 bit signed [2:0] A;
        rand bit signed [2:0] B;


        // c_rst: Reset should be asserted with a low probability
        constraint c_rst {
            rst dist {0 := 95, 1 := 5};
        }

        // c_ADD_MUL: Adder and Multiplier inputs favor MAXPOS, ZERO, MAXNEG
        constraint c_ADD_MUL {
            if (opcode == ADD || opcode == MUL) {
                A dist {MAXNEG := 30, ZERO := 30, MAXPOS := 30, [-3:-1] := 5,
[1:2] := 5};
                B dist {MAXNEG := 30, ZERO := 30, MAXPOS := 30, [-3:-1] := 5,
[1:2] := 5};
            }
        }

        // c_OR_XOR_A: OR/XOR with red_op_A => A one-hot, B zero
        constraint c_OR_XOR_A {
            if ((opcode inside {OR, XOR}) && red_op_A && !red_op_B) {
                A dist {3'b001 := 30, 3'b010 := 30, 3'b100 := 30,
[3'b000:3'b111] := 10};
                B == 3'b000;
```

```
            }
        }

        // c_OR_XOR_B: OR/XOR with red_op_B => B one-hot, A zero
        constraint c_OR_XOR_B {
            if ((opcode inside {OR, XOR}) && red_op_B && !red_op_A) {
                B dist {3'b001 := 30, 3'b010 := 30, 3'b100 := 30,
[3'b000:3'b111] := 10};
                A == 3'b000;
            }
        }

        // c_opcode: Invalid cases should occur less frequent than the valid
cases
        constraint c_opcode {
            opcode dist {[OR:ROTATE] := 80, [INVALID_6:INVALID_7] := 20}
        }

        // c_bypass: Bypass signals should be disabled most of the time
        constraint c_bypass {
            bypass_A dist {0 := 95, 1 := 5};
            bypass_B dist {0 := 95, 1 := 5};
        }

        // No constraints on A/B for shift/rotate handled implicitly
    endclass

endpackage
```

## 3.4. Testbench file:

```
import alsu_pkg::*;

module alsu_tb;
    // TESTBENCH VARIABLES
    int NUM_TEST_CASES = 100;
    int local_error_count = 0;
    int error_count = 0;
    int pass_count = 0;

    // DUT
    parameter INPUT_PRIORITY = "A";
    parameter FULL_ADDER = "ON";
    bit clk;
    bit rst;
    bit cin;
    bit red_op_A;
    bit red_op_B;
    bit bypass_A;
    bit bypass_B;
    bit direction;
    bit serial_in;
    opcode_e opcode;
    bit signed [2:0] A;
    bit signed [2:0] B;
    logic [15:0] leds;
    logic signed [5:0] out;

    // GOLDEN MODEL
    logic signed [5:0] golden_out;
    logic [15:0] golden_leds;

    ALSU #(
        .INPUT_PRIORITY(INPUT_PRIORITY),
        .FULL_ADDER(FULL_ADDER)
    ) 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),
        .opcode(opcode),
        .A(A),
```

```
    .B(B),
    .leds(leds),
    .out(out)
);

ALSU_golden #(
    .INPUT_PRIORITY(INPUT_PRIORITY),
    .FULL_ADDER(FULL_ADDER)
) GOLDEN (
    .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),
    .opcode(opcode),
    .A(A),
    .B(B),
    .leds(golden_leds),
    .out(golden_out)
);

alsu_inputs alsu_obj;

initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

initial begin
    $display("Starting ALSU testbench...");
    alsu_obj = new();

    assert_reset();
    check_result();

    repeat(NUM_TEST_CASES) begin
        wait_cycles(1);
        assert(alsu_obj.randomize());
        rst = alsu_obj.rst;
        cin = alsu_obj.cin;
        red_op_A = alsu_obj.red_op_A;
        red_op_B = alsu_obj.red_op_B;
        bypass_A = alsu_obj.bypass_A;
        bypass_B = alsu_obj.bypass_B;
```

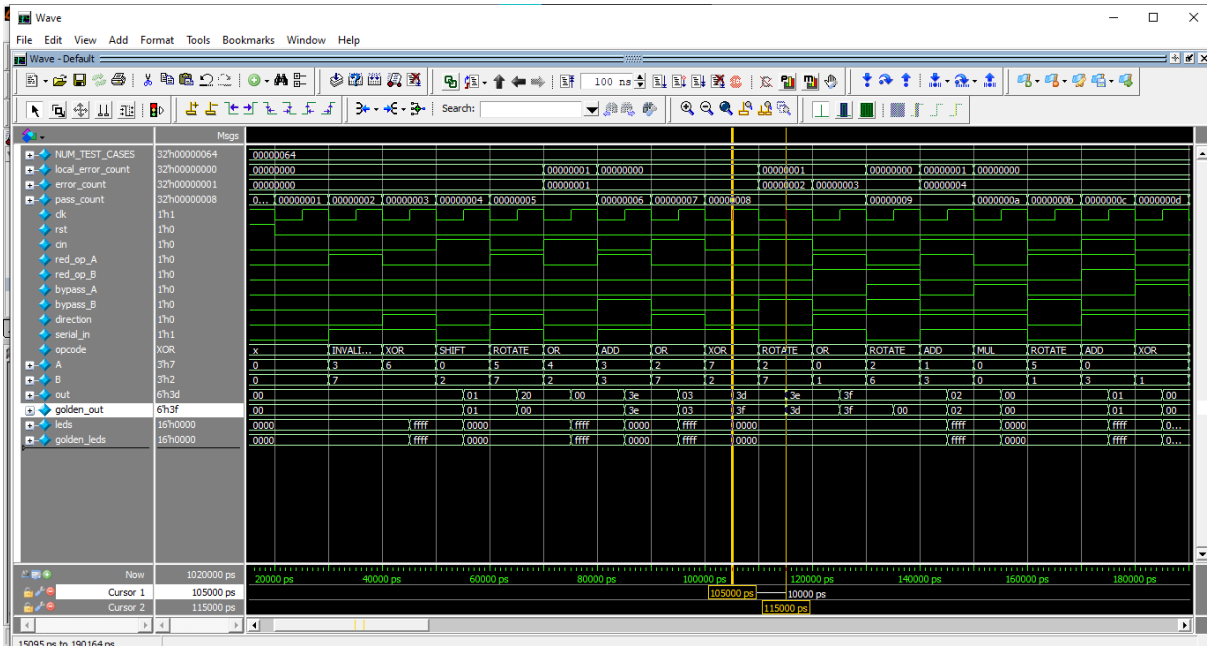
```
            direction = alsu_obj.direction;
            serial_in = alsu_obj.serial_in;
            opcode = alsu_obj.opcode;
            A = alsu_obj.A;
            B = alsu_obj.B;
            check_result();
        end

        // Display completion message
        $display("Simulation Completed: %0d test cases executed.",
NUM_TEST_CASES);
        $display("Test Summary: Passed = %0d, Failed = %0d", pass_count,
error_count);
    $stop;

    end

    task wait_cycles(input int num_cycles);
        repeat (num_cycles) @(negedge clk);
    endtask

    task assert_reset();
        wait_cycles(1);
        rst = 1;
        wait_cycles(1);
        rst = 0;
    endtask

    task check_result();
        local_error_count = 0;

        if (out !== golden_out) begin
            $error("[ERROR] out mismatch. Expected: %0d, Got: %0d",
golden_out, out);
            local_error_count++;
        end

        if (leds !== golden_leds) begin
            $error("[ERROR] leds mismatch. Expected: %b, Got: %b",
golden_leds, leds);
            local_error_count++;
        end

        if (local_error_count == 0) begin
            pass_count++;
            $display("[PASS] Outputs match expected values.");
        end else begin
            error_count++;
```

```
            $display("[FAIL] Total mismatches in this check: %0d",
local_error_count);
        end
    endtask

endmodule
```

## 3.5. Do file:

```
vlog ALSU_pkg.sv
vlog ALSU_golden.v
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.ALSU_tb -cover
add wave *
coverage save ALSU_tb.ucdb -onexit
run -all
# quit -sim
# vcover report ALSU_tb.ucdb -details -annotate -all -output
coverage_rpt.txt
```

## 3.6. Waveform snippet:



## 3.7. Transcript:

```
# Starting ALSU testbench...
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# ** Error: [ERROR] out mismatch. Expected: 0, Got: -32
#    Time: 70 ns  Scope: alsu_tb.check_result File: ALSU_tb.sv Line: 123
# [FAIL] Total mismatches in this check: 1
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# ** Error: [ERROR] out mismatch. Expected: -1, Got: -3
#    Time: 110 ns  Scope: alsu_tb.check_result File: ALSU_tb.sv Line: 123
# [FAIL] Total mismatches in this check: 1
# ** Error: [ERROR] out mismatch. Expected: -3, Got: -2
#    Time: 120 ns  Scope: alsu_tb.check_result File: ALSU_tb.sv Line: 123
# [FAIL] Total mismatches in this check: 1
# [PASS] Outputs match expected values.
# ** Error: [ERROR] out mismatch. Expected: 0, Got: -1
#    Time: 140 ns  Scope: alsu_tb.check_result File: ALSU_tb.sv Line: 123
# [FAIL] Total mismatches in this check: 1
...
# Simulation Completed: 1000 test cases executed.
# Test Summary: Passed = 776, Failed = 225
# ** Note: $stop    : ALSU_tb.sv(108)
```

## 3.8. Bugs:

1. when opcode is ADD and cin = 1
2. case statement uses opcode instead of opcode_reg

```
# Simulation Completed: 1000 test cases executed.
# Test Summary: Passed = 1001, Failed = 0
# ** Note: $stop    : ALSU_tb.sv(108)
```

## 3.9. Coverage report:

```
> vcover report counter_tb.ucdb -details -annotate -all -output
coverage_rpt.txt

================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        32        32         0   100.00%

================================================================================
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      48        48         0   100.00%

================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        120       120         0   100.00%
```