1.

1.1. Design file:

```
import alu_pkg::*;

module alu_seq(operand1, operand2, clk, rst, opcode, out);
input byte operand1, operand2;
input clk, rst;
input opcode_e opcode;
output byte out;

always @(posedge clk) begin
    if (rst)
        out <= 0;
    else
        case (opcode)
            ADD: out <= operand1 + operand2;
            SUB: out <= operand1 - operand2;
            MULT:out <= operand1 * operand2;
            DIV: out <= operand1 / operand2;
            default: out <= 0;
        endcase
end
endmodule
```

1.2. Testbench file:

```
import alu_pkg::*;

module alu_tb();
    // TESTBENCH VARIABLES
    int NUM_TEST_CASES = 200;
    int error_count = 0;
    int pass_count = 0;

    // DUT
    byte operand1, operand2;
    bit clk, rst;
    opcode_e opcode;
    byte out;

    // GOLDEN MODEL
    byte expected_out;

    alu_seq DUT(operand1, operand2, clk, rst, opcode, out);

    Transaction tr = new();

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

    initial begin
        $display("Starting ALU testbench...");
        assert_reset();
        check_result(out, 0);

        repeat (NUM_TEST_CASES) begin
            assert(tr.randomize());
            operand1 = tr.operand1;
            operand2 = tr.operand2;
            opcode = tr.opcode;
            @(negedge clk);
            expected_out = golden_model(tr);
            check_result(out, expected_out);
        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(
        input byte out,
        input byte expected_out
    );

        if (out !== expected_out) begin
            error_count++;
            $display("[FAIL] mismatch. Expected: %0d, Got: %0d", expected_out,
out);
        end else begin
            pass_count++;
            $display("[PASS] Outputs match expected values.");
        end

    endtask

    function automatic byte golden_model(input Transaction tr);
        if (rst) begin
            return 0;
        end else begin
            return (tr.opcode == ADD)  ? tr.operand1 + tr.operand2 :
                (tr.opcode == SUB)  ? tr.operand1 - tr.operand2 :
                (tr.opcode == MULT) ? tr.operand1 * tr.operand2 :
                                       tr.operand1 / tr.operand2;
        end
    endfunction

endmodule
```

1.3. Package file:

```
package alu_pkg;

    typedef enum {ADD, SUB, MULT, DIV} opcode_e;
    parameter MAXPOS = 127;
    parameter ZERO = 0;
    parameter MAXNEG = -128;

    class Transaction;
        rand opcode_e opcode;
        rand byte operand1;
        rand byte operand2;
        bit clk;

        covergroup g1 @(posedge clk);
            opcode_cp: coverpoint opcode {
                bins opcode_add_or_sub = {ADD, SUB};
                bins opcode_add_to_sub = (ADD => SUB);
                illegal_bins opcode_not_DIV = {DIV};
            }

            operand1_cp: coverpoint operand1 {
                bins operand1_maxpos = {MAXPOS};
                bins operand1_zero = {ZERO};
                bins operand1_maxneg = {MAXNEG};
                bins misc = default;
            }
        endgroup

        function new();
            g1 = new();
        endfunction
    endclass

endpackage
```

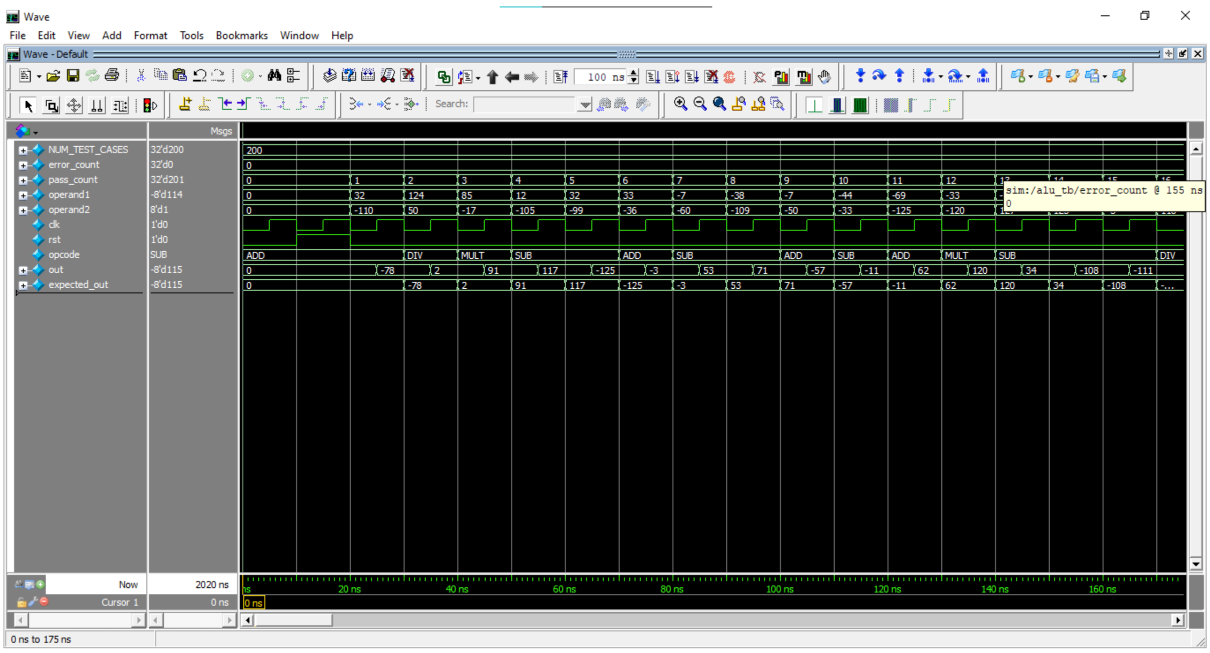
1.4. Bugs:

No bugs detected.

1.5. Do file:

```
vlib work
vlog alu_pkg.sv
vlog alu.sv alu_tb.sv +cover -covercells
vsim -c -voptargs=+acc work.alu_tb -cover
add wave *
coverage save alu_tb.ucdb -onexit
run -all
```

1.6. Waveform snippet:



```
NUM_TEST_CASES  32'd200
error_count     32'd0
pass_count      32'd201
operand1        -8'd114
operand2        8'd1
clk             1'd0
rst             1'd0
opcode          SUB
out             -8'd115
expected_out    -8'd115
```

```
sim:/alu_tb/error_count @ 155 ns
0
```

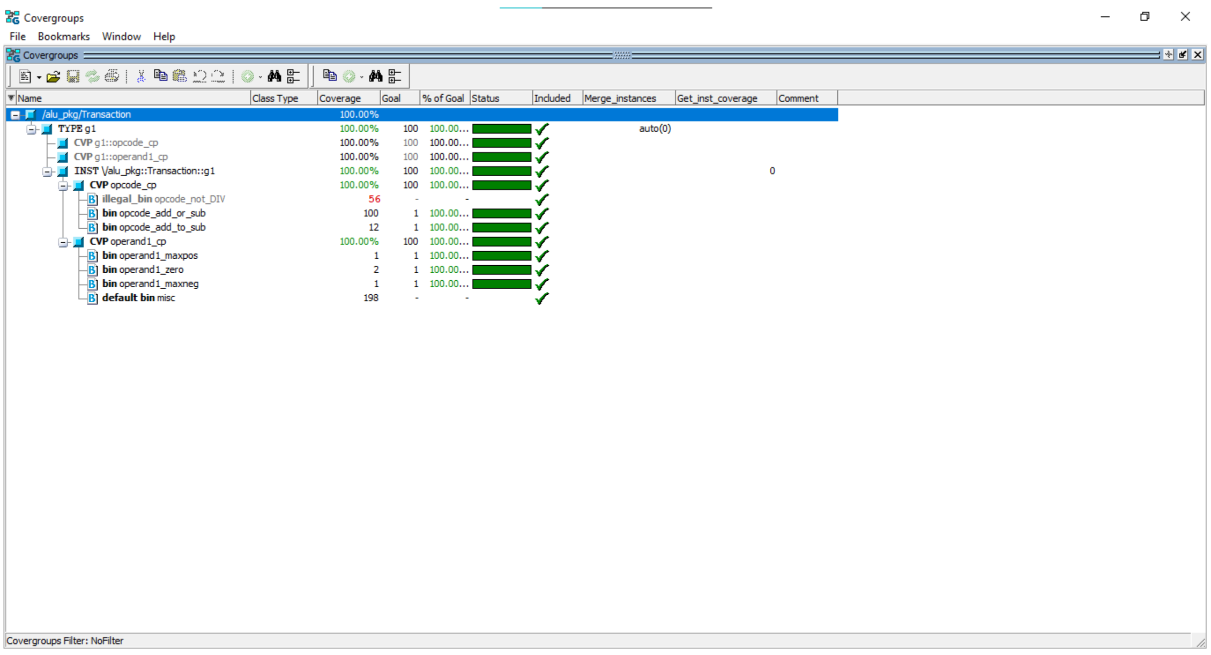
1.7. Transcript:

```
# Starting ALU testbench...
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# ** Error: (vsim-8565) Illegal state bin was hit at value=DIV. The bin
counter for the illegal bin '\/alu_pkg::Transaction::g1
.opcode_cp.opcode_not_DIV' is 1.
#    Time: 35 ns  Iteration: 0  Region: /alu_pkg::Transaction::#g1#
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# Simulation Completed: 200 test cases executed.
# Test Summary: Passed = 201, Failed = 0
# ** Note: $stop    : alu_tb.sv(48)
```

1.8. Code coverage:

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

=============================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         6         6         0   100.00%

=============================================================================
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       6         6         0   100.00%

=============================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         56        56         0   100.00%
```

1.9. Functional coverage:



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/alu_pkg/Transaction		100.00%							
TYPE g1		100.00%	100	100.00...	🟩	✓	auto(0)		
CVP g1::opcode_cp		100.00%	100	100.00...	🟩	✓			
CVP g1::operand1_cp		100.00%	100	100.00...	🟩	✓			
INST \/alu_pkg::Transaction::g1		100.00%	100	100.00...	🟩	✓		0	
CVP opcode_cp		100.00%	100	100.00...	🟩	✓			
illegal_bin opcode_not_DIV		56	-	-		✓			
bin opcode_add_or_sub		100	1	100.00...	🟩	✓			
bin opcode_add_to_sub		12	1	100.00...	🟩	✓			
CVP operand1_cp		100.00%	100	100.00...	🟩	✓			
bin operand1_maxpos		1	1	100.00...	🟩	✓			
bin operand1_zero		2	1	100.00...	🟩	✓			
bin operand1_maxneg		1	1	100.00...	🟩	✓			
default bin misc		198	-	-		✓			

Covergroups Filter: NoFilter

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
COUNTER0	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.
COUNTER1	When load_n is low, data_load should be loaded into count_out.	Directed + random valid data_load	Cover all values of data_load	Checker verifies count_out == data_load when load_n == 0.
COUNTER2	When ce = 1, up_down = 1, counter increments each clock cycle.	Directed increment loop	Cover all values of count_out, and transition bins from max to zero	Checker validates correct counting sequence and max_count assertion at max.
COUNTER3	When ce = 1, up_down = 0, counter decrements each clock cycle.	Directed decrement loop	Cover all values of count_out, and transition bins from zero to max	Checker validates correct counting sequence and zero assertion at 0.

2.3. Package file:

```
package counter_pkg;

    parameter int WIDTH = 4;

    class counter_txn;
        bit clk;
        rand bit rst_n;
        rand bit load_n;
        rand bit up_down;
        rand bit ce;
        randc bit [WIDTH - 1:0] data_load;
        bit [WIDTH - 1:0] count_out;
        bit max_count;
        bit zero;

        constraint c1 {
            rst_n dist {0 := 5, 1 := 95};
            load_n dist {0 := 70, 1 := 30};
            ce dist {0 := 30, 1 := 70};
            up_down dist {0 := 50, 1 := 50};
        }

        covergroup g1 @(posedge clk);
            data_load_cp: coverpoint data_load iff (rst_n && (!load_n));
            count_out_up_cp: coverpoint count_out iff (rst_n && load_n &&
up_down){
                bins all_values[] = {[{WIDTH{1'b0}}:{WIDTH{1'b1}}]};
                bins max_to_zero = ({WIDTH{1'b1}} => {WIDTH{1'b0}});
            }

            count_out_down_cp: coverpoint count_out iff (rst_n && load_n &&
(!up_down)){
                bins all_values[] = {[{WIDTH{1'b0}}:{WIDTH{1'b1}}]};
                bins zero_to_max = ({WIDTH{1'b0}} => {WIDTH{1'b1}});
            }
        endgroup

        function new();
            g1 = new();
        endfunction

    endclass : counter_txn

endpackage : counter_pkg
```

2.4. Testbench file:

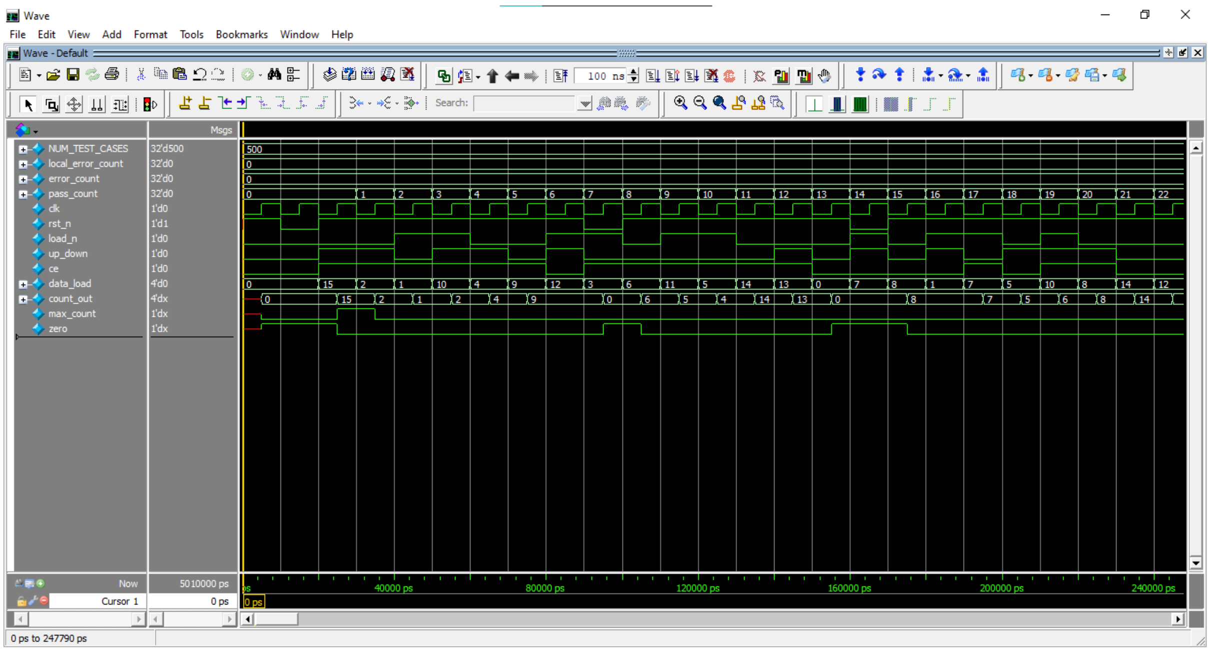
```
import counter_pkg::*;

module counter_tb;
    // TESTBENCH VARIABLES
    int NUM_TEST_CASES = 500;
    int local_error_count = 0;
    int error_count = 0;
    int pass_count  = 0;

    // DUT
    bit clk;
    bit rst_n;
    bit load_n;
    bit up_down;
    bit ce;
    bit [WIDTH - 1:0] data_load;
    logic [WIDTH - 1:0] count_out;
    logic max_count;
    logic 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 = new();

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

    initial begin
        $display("Starting Counter testbench...");
        assert_reset();
        repeat(NUM_TEST_CASES - 1) begin
            assert(txn.randomize());
            drive_inputs(txn);
            wait_cycles(1);
            golden_model(txn);
            check_result(txn);
        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();
        rst_n = 1;
        wait_cycles(1);
        rst_n = 0;
        wait_cycles(1);
        rst_n = 1;
    endtask

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

    task check_result(input counter_txn txn);
        local_error_count = 0;

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

        if (max_count !== txn.max_count) begin
            $error("[ERROR] max_count mismatch. Expected: %b, Got: %b",
txn.max_count, max_count);
```

```
            local_error_count++;
        end

        if (zero !== txn.zero) begin
            $error("[ERROR] zero mismatch. Expected: %b, Got: %b", txn.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

    task golden_model(input counter_txn txn);
        if (!txn.rst_n) begin
            txn.count_out = 0;
        end else if (!txn.load_n) begin
            txn.count_out = txn.data_load;
        end else if (txn.ce) begin
            if (txn.up_down) begin
                txn.count_out++;
            end else begin
                txn.count_out--;
            end
        end
        txn.max_count = (txn.count_out == {WIDTH{1'b1}});
        txn.zero = (txn.count_out == 0);
    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:

```
# [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.
# [PASS] Outputs match expected values.
# [PASS] Outputs match expected values.
# Simulation Completed: 500 test cases executed.
# Test Summary: Passed = 499, Failed = 0
```

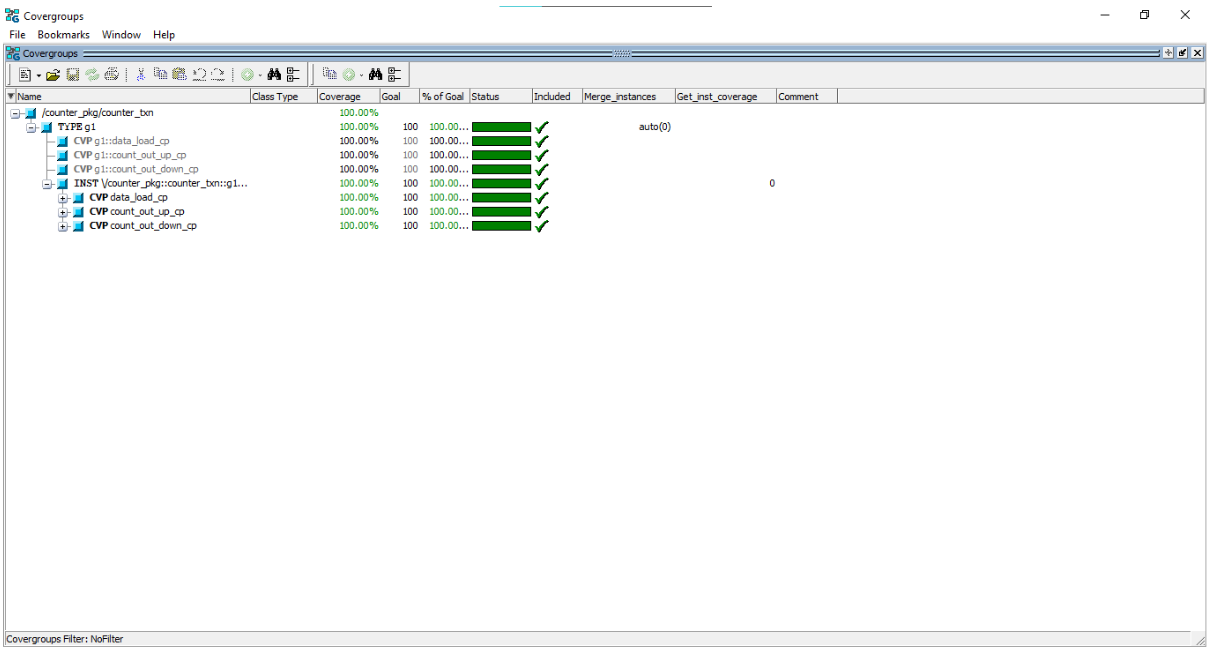
2.8. Bugs:

No bugs detected.

2.9. Code coverage:

```
> 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%
```

2.10. Functional coverage:



Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/counter_pkg/counter_txn		100.00%							
TYPE g1		100.00%	100	100.00...	🟩	✓	auto(0)		
CVP g1::data_load_cp		100.00%	100	100.00...	🟩	✓			
CVP g1::count_out_up_cp		100.00%	100	100.00...	🟩	✓			
CVP g1::count_out_down_cp		100.00%	100	100.00...	🟩	✓			
INST \/counter_pkg::counter_txn::g1...		100.00%	100	100.00...	🟩	✓		0	
CVP data_load_cp		100.00%	100	100.00...	🟩	✓			
CVP count_out_up_cp		100.00%	100	100.00...	🟩	✓			
CVP count_out_down_cp		100.00%	100	100.00...	🟩	✓			

Covergroups Filter: NoFilter

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	Cover MAXPOS, MAXNEG, ZERO as well as default for all other values	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	Cover MAXPOS, MAXNEG, ZERO as well as default for all other values	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	Cover the walking ones values for 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

Label	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
ALSU7	when opcode is XOR & any of the inputs reduction_A or reduction_B is high.	Randomized under constraints on the A and B	Cover the walking ones values for A and B	Checker ensures out = (INPUT_PRIORITY == ‘A’) ? ^A : ^B
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.	Cover the SHIFT opcode	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.	Cover the ROTATE opcode	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)	Cover the illegal opcode	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
ALSU12	In case of valid opcode values, the output should be evaluated based on the operation	Randomized under constraints	Cover the legal opcode as well as the transition to make sure output is evaluated at each opcode change with each transition	Checker ensures out = expected_out

3.3. Package file:

```
package alsu_pkg;

    typedef enum logic [2:0] {OR, XOR, ADD, MULT, 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;

    parameter NUM_VALID_OPCODES = 6;

    class alsu_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 bit signed [2:0] A;
        rand bit signed [2:0] B;
        bit [15:0] leds;
        bit signed [5:0] out;

        rand opcode_e opcode_seq [NUM_VALID_OPCODES];

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

        // c_ADD_MULT: Adder and Multiplier inputs favor MAXPOS, ZERO, MAXNEG
        constraint c_ADD_MULT {
            if (opcode == ADD || opcode == MULT) {
                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};
        }

        constraint c_red_op {
            red_op_A dist {0 := 50, 1 := 50};
            red_op_B dist {0 := 50, 1 := 50};
        }

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

        constraint opcode_seq_c {
            foreach (opcode_seq[i]) {
                foreach (opcode_seq[j]) {
                    if (i != j) {
                        opcode_seq[i] != opcode_seq[j];
                        opcode_seq[i] inside {[OR:ROTATE]};
                    }
                }
            }
        }
        covergroup cg;
            A_cp :coverpoint A{
                bins A_data_0 = {0};
                bins A_data_max = {MAXPOS};
                bins A_data_min = {MAXNEG};
                bins A_data_walkingones[] = {3'b001,3'b010,3'b100} iff
(red_op_A);
                bins A_data_default = default;
            }

            B_cp :coverpoint B{
                bins B_data_0 = {0};
                bins B_data_max = {MAXPOS};
                bins B_data_min = {MAXNEG};
                bins B_data_walkingones[] = {3'b001,3'b010,3'b100} iff
(red_op_B && (!red_op_A));
                bins B_data_default = default;
            }

            opcode_cp :coverpoint opcode{
                bins Bins_shift[] = {SHIFT, ROTATE};
                bins Bins_arith[] = {ADD, MULT};
                bins Bins_bitwise[] = {OR, XOR};
                illegal_bins Bins_invalid = {INVALID_6, INVALID_7};
                bins Bins_trans = (OR => XOR => ADD => MULT => SHIFT =>
ROTATE);
            }
        endgroup

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

        task sample_cg();
            if((!rst) || (!bypass_A) || (!bypass_B)) begin
                @(negedge clk) cg.sample();
            end
        endtask
    endclass

endpackage : alsu_pkg
```

3.4. Testbench file:

```
import alsu_pkg::*;

module alsu_tb;
    // TESTBENCH VARIABLES
    int NUM_TEST_CASES = 3000;
    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;

    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_inputs alsu_obj = new();

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

initial begin
    $display("Starting ALSU testbench...");
    assert_reset();
    alsu_obj.opcode_seq_c.constraint_mode(0);

    repeat(NUM_TEST_CASES/2) begin
        assert(alsu_obj.randomize());
        drive_inputs(alsu_obj);
        wait_cycles(1);
        golden_model(alsu_obj);
        wait_cycles(1);
        check_result(alsu_obj);
        alsu_obj.sample_cg();
    end

    alsu_obj.constraint_mode(0);
    alsu_obj.opcode_seq_c.constraint_mode(1);
    alsu_obj.red_op_A = 0;
    alsu_obj.red_op_B = 0;
    alsu_obj.bypass_A = 0;
    alsu_obj.bypass_B = 0;
    alsu_obj.rst = 0;

    repeat(NUM_TEST_CASES/2) begin
        assert(alsu_obj.randomize());
        foreach(alsu_obj.opcode_seq[i]) begin
            alsu_obj.opcode = alsu_obj.opcode_seq[i];
            drive_inputs(alsu_obj);
            wait_cycles(1);
            golden_model(alsu_obj);
            wait_cycles(1);
            check_result(alsu_obj);
            alsu_obj.sample_cg();
        end
    end

    // Display completion message
```

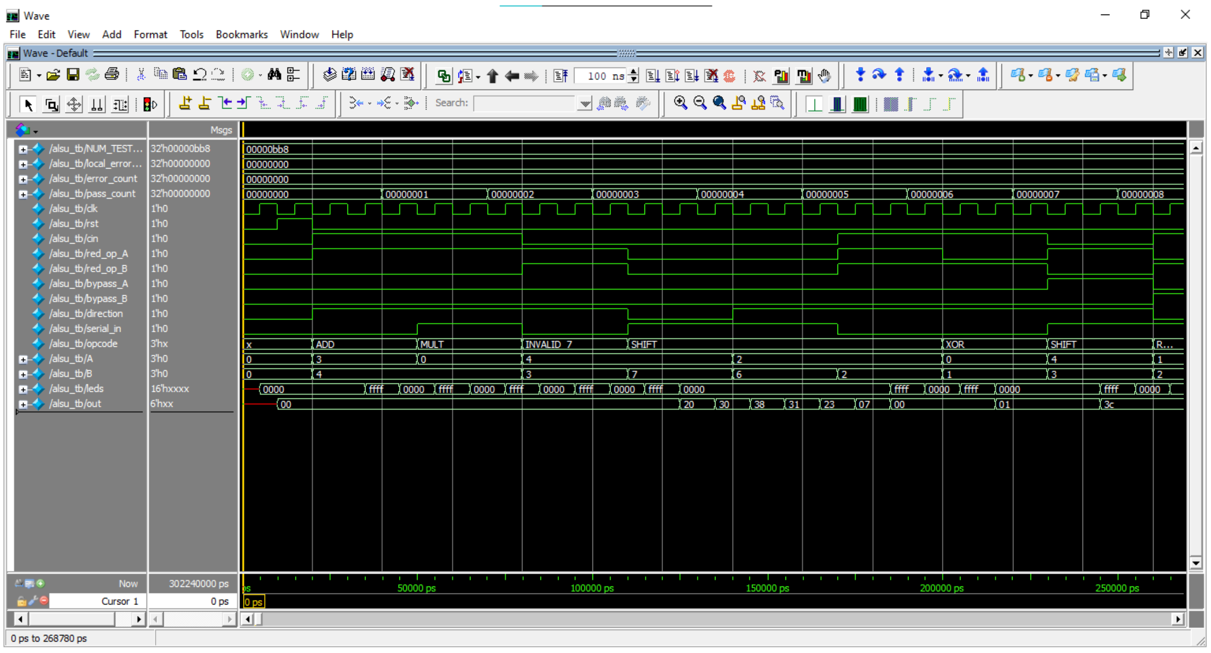
```
        $display("Simulation Completed: %0d test cases executed.",
(NUM_TEST_CASES / 2) + 6 * (NUM_TEST_CASES / 2));
        $display("Test Summary: Passed = %0d, Failed = %0d", pass_count,
error_count);
        $stop;
    end

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

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

    task drive_inputs(input alsu_inputs alsu_obj);
        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;
    endtask

    task check_result(input alsu_inputs alsu_obj);
        local_error_count = 0;

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

        if (leds !== alsu_obj.leds) begin
            $error("[ERROR] leds mismatch. Expected: %b, Got: %b",
alsu_obj.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

    function void golden_model(input alsu_inputs alsu_obj);
        bit invalid_red_op, invalid_opcode, invalid;

        // invalid checks
        invalid_red_op = (alsu_obj.red_op_A | alsu_obj.red_op_B) &
(alsu_obj.opcode[1] | alsu_obj.opcode[2]);
        invalid_opcode = alsu_obj.opcode[1] & alsu_obj.opcode[2];
        invalid        = invalid_red_op | invalid_opcode;

        // LEDs behavior
        if (alsu_obj.rst) begin
            alsu_obj.leds = 16'h0;
        end else begin
            if (invalid) begin
                alsu_obj.leds = ~alsu_obj.leds;   // toggle like RTL
            end else begin
                alsu_obj.leds = 16'h0;
            end
        end

        if (alsu_obj.rst) begin
            alsu_obj.out = 0;
        end else begin
            if (alsu_obj.bypass_A && alsu_obj.bypass_B) begin
                alsu_obj.out = ("A" == "A") ? alsu_obj.A : alsu_obj.B;
            end else if (alsu_obj.bypass_A) begin
                alsu_obj.out = alsu_obj.A;
            end else if (alsu_obj.bypass_B) begin
                alsu_obj.out = alsu_obj.B;
            end else if (invalid) begin
                alsu_obj.out = 0;
            end else begin
                case (alsu_obj.opcode)
                    OR: begin
                        if (alsu_obj.red_op_A && alsu_obj.red_op_B) begin
                            alsu_obj.out = ("A" == "A") ? (|alsu_obj.A) :
(|alsu_obj.B);
                        end else if (alsu_obj.red_op_A) begin
```

```
                                alsu_obj.out = |alsu_obj.A;
                            end else if (alsu_obj.red_op_B) begin
                                alsu_obj.out = |alsu_obj.B;
                            end else begin
                                alsu_obj.out = alsu_obj.A | alsu_obj.B;
                            end
                        end
                        XOR: begin
                            if (alsu_obj.red_op_A && alsu_obj.red_op_B) begin
                                alsu_obj.out = ("A" == "A") ? (^alsu_obj.A) :
(^alsu_obj.B);
                            end else if (alsu_obj.red_op_A) begin
                                alsu_obj.out = ^alsu_obj.A;
                            end else if (alsu_obj.red_op_B) begin
                                alsu_obj.out = ^alsu_obj.B;
                            end else begin
                                alsu_obj.out = alsu_obj.A ^ alsu_obj.B;
                            end
                        end
                        ADD: begin
                            if ("ON" == "ON") begin // FULL_ADDER check
                                alsu_obj.out = alsu_obj.A + alsu_obj.B +
alsu_obj.cin;
                            end else begin
                                alsu_obj.out = alsu_obj.A + alsu_obj.B;
                            end
                        end
                        MULT: alsu_obj.out = alsu_obj.A * alsu_obj.B;
                        SHIFT: begin
                            if (alsu_obj.direction) begin
                                alsu_obj.out = {out[4:0], serial_in};
                            end else begin
                                alsu_obj.out = {serial_in, out[5:1]};
                            end
                        end
                        ROTATE: begin
                            if (alsu_obj.direction) begin
                                alsu_obj.out = {out[4:0], out[5]};
                            end else begin
                                alsu_obj.out = {out[0], out[5:1]};
                            end
                        end
                        default: alsu_obj.out = 0;
                    endcase
                end
            end
        endfunction
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
```

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.
…
# Simulation Completed: 10500 test cases executed.
# Test Summary: Passed = 10500, Failed = 0
# ** Note: $stop    : ALSU_tb.sv(108)
```

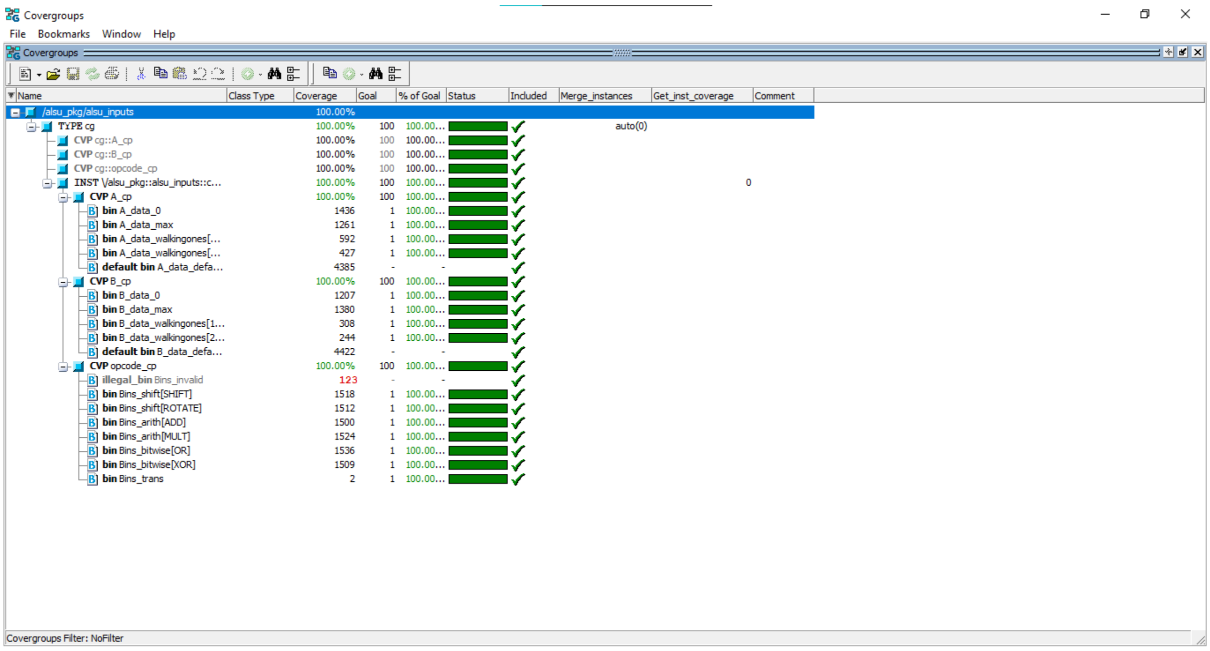
3.8. Bugs:

No bugs detected

3.9. Code coverage:

```
> vcover report ALSU_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%
```

3.10. Functional coverage:


Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/alsu_pkg/alsu_inputs		100.00%							
TYPE cg		100.00%	100	100.00...	🟩	✓	auto(0)		
CVP cg::A_cp		100.00%	100	100.00...	🟩	✓			
CVP cg::B_cp		100.00%	100	100.00...	🟩	✓			
CVP cg::opcode_cp		100.00%	100	100.00...	🟩	✓			
INST \/alsu_pkg::alsu_inputs::c...		100.00%	100	100.00...	🟩	✓		0	
CVP A_cp		100.00%	100	100.00...	🟩	✓			
bin A_data_0		1436	1	100.00...	🟩	✓			
bin A_data_max		1261	1	100.00...	🟩	✓			
bin A_data_walkingones[...		592	1	100.00...	🟩	✓			
bin A_data_walkingones[...		427	1	100.00...	🟩	✓			
default bin A_data_defa...		4385	-	-		✓			
CVP B_cp		100.00%	100	100.00...	🟩	✓			
bin B_data_0		1207	1	100.00...	🟩	✓			
bin B_data_max		1380	1	100.00...	🟩	✓			
bin B_data_walkingones[1...		308	1	100.00...	🟩	✓			
bin B_data_walkingones[2...		244	1	100.00...	🟩	✓			
default bin B_data_defa...		4422	-	-		✓			
CVP opcode_cp		100.00%	100	100.00...	🟩	✓			
illegal_bin Bins_invalid		123	-	-		✓			
bin Bins_shift[SHIFT]		1518	1	100.00...	🟩	✓			
bin Bins_shift[ROTATE]		1512	1	100.00...	🟩	✓			
bin Bins_arith[ADD]		1500	1	100.00...	🟩	✓			
bin Bins_arith[MULT]		1524	1	100.00...	🟩	✓			
bin Bins_bitwise[OR]		1536	1	100.00...	🟩	✓			
bin Bins_bitwise[XOR]		1509	1	100.00...	🟩	✓			
bin Bins_trans		2	1	100.00...	🟩	✓			

Covergroups Filter: NoFilter

4.

4.1. Design file:

```
module my_mem(
    input clk,
    input write,
    input read,
    input [7:0] data_in,
    input [15:0] address,
    output reg [7:0] data_out
);
    // Declare a 9-bit associative array using the logic data type & the key
of int datatype
    logic [8:0] mem_array [int];

    always @(posedge clk) begin
        if (write)
            mem_array[address] = {~^data_in, data_in};
        else if (read)
            data_out = mem_array[address];
    end
endmodule
```

4.2. Verification plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
RAM1	When write is asserted high, data_in will be written in ram in the address, even parity will be calculated and stored the MSB.	Write port is directed high, data_in and address are randomized without constraints	-	-
RAM2	When read is asserted high and write is asserted low, data_out will be the word written in ram in the address.	Read port is directed high, write port is directed low, address is randomized without constraints	-	Checker ensures data_out = mem[address]

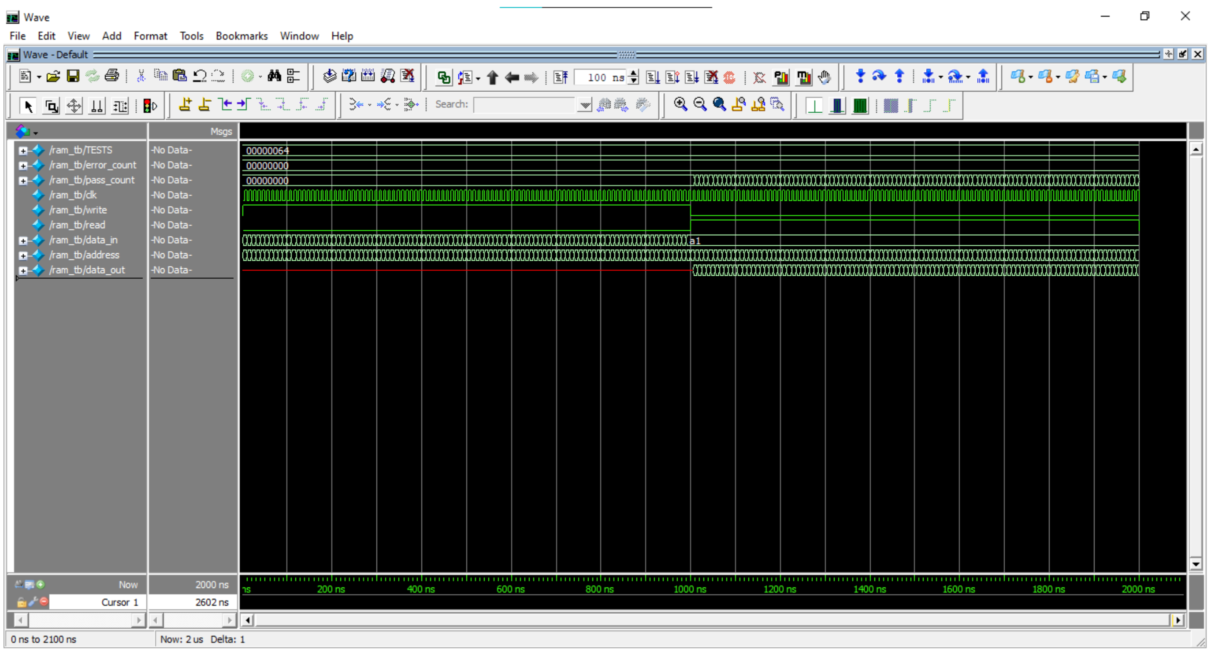
4.3. Testbench file:

```
module ram_tb ();
    // TESTBENCH VARIABLES
    int TESTS = 100;
    int error_count = 0;
    int pass_count = 0;

    // DUT
    bit clk;
    bit write;
    bit read;
    bit [7:0] data_in;
    bit [15:0] address;
    logic [8:0] data_out;

    // GOLDEN MODLE
    // stores random addresses
    bit [15:0] address_array [];

    // stores corresponding random data values
    bit [7:0] data_to_write_array [];

    // stores expected data, indexed by address
    bit [8:0] data_read_expect_assoc [int];

    // stores the actual data read from the RAM
    logic [7:0] data_read_queue [$];

    my_mem DUT (clk, write, read, data_in, address, data_out);

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

    initial begin
        write = 0;
        read = 0;
        address_array = new[TESTS];
        data_to_write_array = new[TESTS];

        // Data preparation
        stimulus_gen();
        golden_model();
        // Write Operations
        $display("Starting RAM testbench...");
        write = 1;
        read = 0;
        for (int i = 0; i < TESTS; i++) begin
            address = address_array[i];
            data_in = data_to_write_array[i];
            wait_cycles(1);
        end

        // Read and Self-Checking
        write = 0;
        read = 1;
        for (int i = 0; i < TESTS; i++) begin
            address = address_array[i];
            wait_cycles(1);
            check9Bits(address);
            data_read_queue.push_back(data_out);
        end

        // Test Completion and Reporting
        write = 0;
        read = 0;
        $display("Elements in data_read_queue are:");
        while (data_read_queue.size) begin
            $displayh(data_read_queue.pop_front());
        end

        // Display completion message
        $display("Simulation Completed: %0d test cases executed.", TESTS);
        $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 check9Bits(input [15:0] address);
        if (data_read_expect_assoc[address] !== data_out) begin
            error_count++;
            $error("[ERROR] data_out mismatch. Expected: 0x%0h, Got: 0x%0h",
data_read_expect_assoc[address], data_out);
        end else begin
            pass_count++;
        end
    endtask
    task stimulus_gen();
        for (int i = 0; i < TESTS; i++) begin
            address_array[i] = $random;
            data_to_write_array[i] = $random;
        end
    endtask

    task golden_model();
        for (int i = 0; i < TESTS; i++) begin
            data_read_expect_assoc[address_array[i]] =
{^data_to_write_array[i], data_to_write_array[i]};
        end
    endtask

endmodule
```

4.4. Do file:

```
vlog ram.sv ram_tb.sv +cover -covercells
vsim -voptargs=+acc work.ram_tb -cover
add wave *
coverage save ram_tb.ucdb -onexit
run -all
```

4.5. Waveform snippet:



4.6. Transcript:

```
# Starting RAM testbench...
# Elements in data_read_queue are:
# 81
# 63
# 8d
# 12
…
# Simulation Completed: 100 test cases executed.
# Test Summary: Passed = 100, Failed = 0
# ** Note: $stop    : ram_tb.sv(78)
```

4.7. Bugs:

```
1. data_out width should be 9 not 8.
2. Even parity is calculated instead of odd parity
```