1.

1.1. 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 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;
```

```verilog
      direction_reg <= direction;
      serial_in_reg <= serial_in;
      opcode_reg <= opcode;
      A_reg <= A;
      B_reg <= B;
   end
end

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

//ALSU output processing
always @(posedge clk or posedge rst) begin
   if(rst) begin
     out <= 0;
   end
   else begin
     if (bypass_A_reg && bypass_B_reg)
       out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
     else if (bypass_A_reg)
       out <= A_reg;
     else if (bypass_B_reg)
       out <= B_reg;
     else if (invalid)
         out <= 0;
     else begin
         case (opcode_reg)
           3'h0: begin
             if (red_op_A_reg && red_op_B_reg)
               out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
             else if (red_op_A_reg)
               out <= |A_reg;
             else if (red_op_B_reg)
               out <= |B_reg;
             else
               out <= A_reg | B_reg;
           end
           3'h1: begin
             if (red_op_A_reg && red_op_B_reg)
```

```
              out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
            else if (red_op_A_reg)
              out <= ^A_reg;
            else if (red_op_B_reg)
              out <= ^B_reg;
            else
              out <= A_reg ^ B_reg;
          end
          3'h2: out <= 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
```

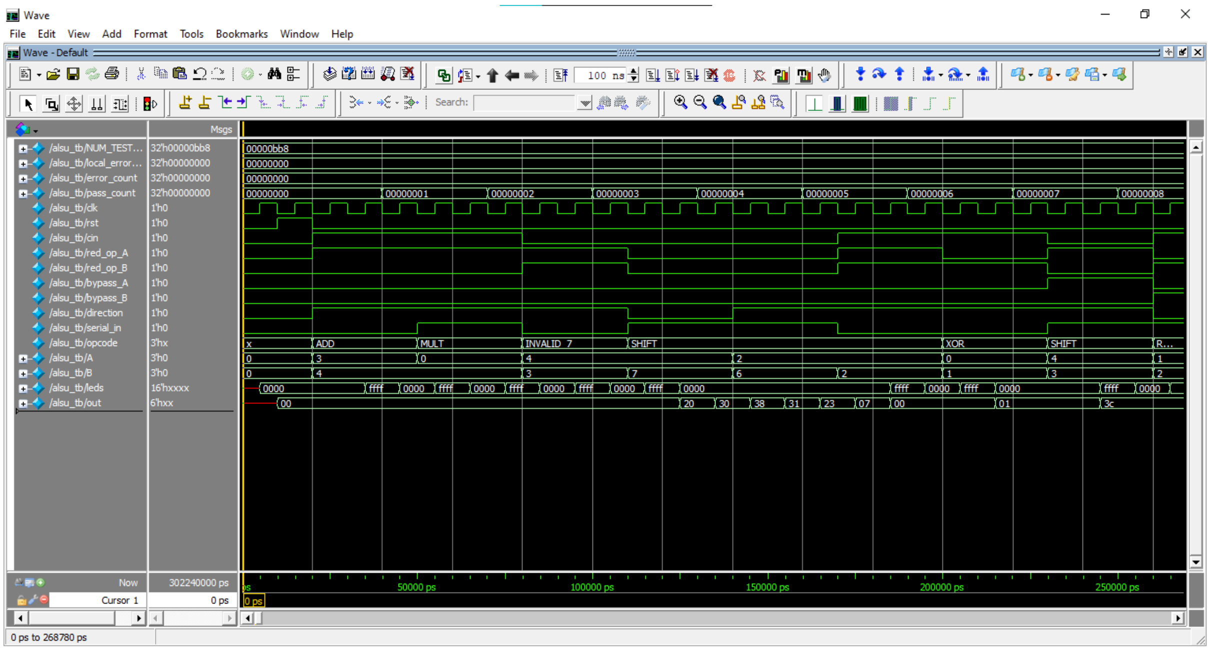
1.2. Package file:

```
package alsu_pkg;

    typedef enum {OR, XOR, ADD, MULT, SHIFT, ROTATE, INVALID_6, INVALID_7}
opcode_e;

    typedef enum {
        MAXNEG = -4,
        ZERO = 0,
        MAXPOS = 3
    } 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;
            ADD_MULT_A_cp: coverpoint A{
                bins A_data_0 = {0};
                bins A_data_max = {MAXPOS};
                bins A_data_min = {MAXNEG};
                bins A_data_default = default;
            }

            RED_A_cp: coverpoint A iff (red_op_A){
                bins A_data_walkingones[] = {3'sb001,3'sb010,3'sb100} iff
(red_op_A);
            }

            ADD_MULT_B_cp: coverpoint B{
                bins B_data_0 = {0};
                bins B_data_max = {MAXPOS};
                bins B_data_min = {MAXNEG};
                bins B_data_default = default;
            }

            RED_B_cp: coverpoint B iff (red_op_B && (!red_op_A)){
                bins B_data_walkingones[] = {3'sb001,3'sb010,3'sb100};
            }

            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);
            }

            opcode_shift_cp: coverpoint opcode{
                option.weight = 0;
                bins bins_shift[] = {SHIFT, ROTATE};
            }

            opcode_arith_cp: coverpoint opcode{
                option.weight = 0;
                bins bins_arith[] = {ADD, MULT};
            }

            opcode_bitwise_cp: coverpoint opcode{
                option.weight = 0;
                bins bins_arith[] = {OR, XOR};
            }
            opcode_not_bitwise_cp: coverpoint opcode{
                option.weight = 0;
                bins bins_not_bitwise[] = {[ADD:$]};
            }

            ADD_MULT_cross: cross ADD_MULT_A_cp, ADD_MULT_B_cp,
opcode_arith_cp;

            ADD_cin_cross: cross cin, opcode_arith_cp{
                option.cross_auto_bin_max = 0;
                bins cin_0 = binsof(opcode_arith_cp.bins_arith) intersect
{ADD} &&
                            binsof(cin) intersect {0};
                bins cin_1 = binsof(opcode_arith_cp.bins_arith) intersect
{ADD} &&
                            binsof(cin) intersect {1};
            }

            SHIFT_direction_cross: cross direction, opcode_shift_cp{
                option.cross_auto_bin_max = 0;
                bins direction_0 = binsof(opcode_shift_cp.bins_shift) &&
                            binsof(direction) intersect {0};
                bins direction_1 = binsof(opcode_shift_cp.bins_shift) &&
                            binsof(direction) intersect {1};
            }

            SHIFT_serial_in_cross: cross serial_in, opcode_shift_cp{
                option.cross_auto_bin_max = 0;
                bins serial_in_0 = binsof(opcode_shift_cp.bins_shift)
intersect {SHIFT} &&
                            binsof(serial_in) intersect {0};
                bins serial_in_1 = binsof(opcode_shift_cp.bins_shift)
intersect {SHIFT} &&
                            binsof(serial_in) intersect {1};
            }

            red_op_A_cross: cross RED_A_cp, ADD_MULT_B_cp, opcode_bitwise_cp
iff (red_op_A){
                ignore_bins B_data_max = binsof(ADD_MULT_B_cp.B_data_max);
                ignore_bins B_data_min = binsof(ADD_MULT_B_cp.B_data_min);
            }

            red_op_B_cross: cross RED_B_cp, ADD_MULT_A_cp, opcode_bitwise_cp
iff (red_op_B && (!red_op_A)){
                ignore_bins A_data_max = binsof(ADD_MULT_A_cp.A_data_max);
                ignore_bins A_data_min = binsof(ADD_MULT_A_cp.A_data_min);
            }
            invalid_red_op_A_cross: cross red_op_A, opcode_not_bitwise_cp{
                ignore_bins red_op_A_0 = binsof(red_op_A) intersect {0};
                illegal_bins red_op_A_1 = binsof(red_op_A) intersect {1};
            }

            invalid_red_op_B_cross: cross red_op_B, opcode_not_bitwise_cp{
                ignore_bins red_op_B_0 = binsof(red_op_B) intersect {0};
                illegal_bins red_op_B_1 = binsof(red_op_B) intersect {1};
            }
        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
```

1.3. Do file:

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

1.4. Waveform snippet:



1.5. 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)
```

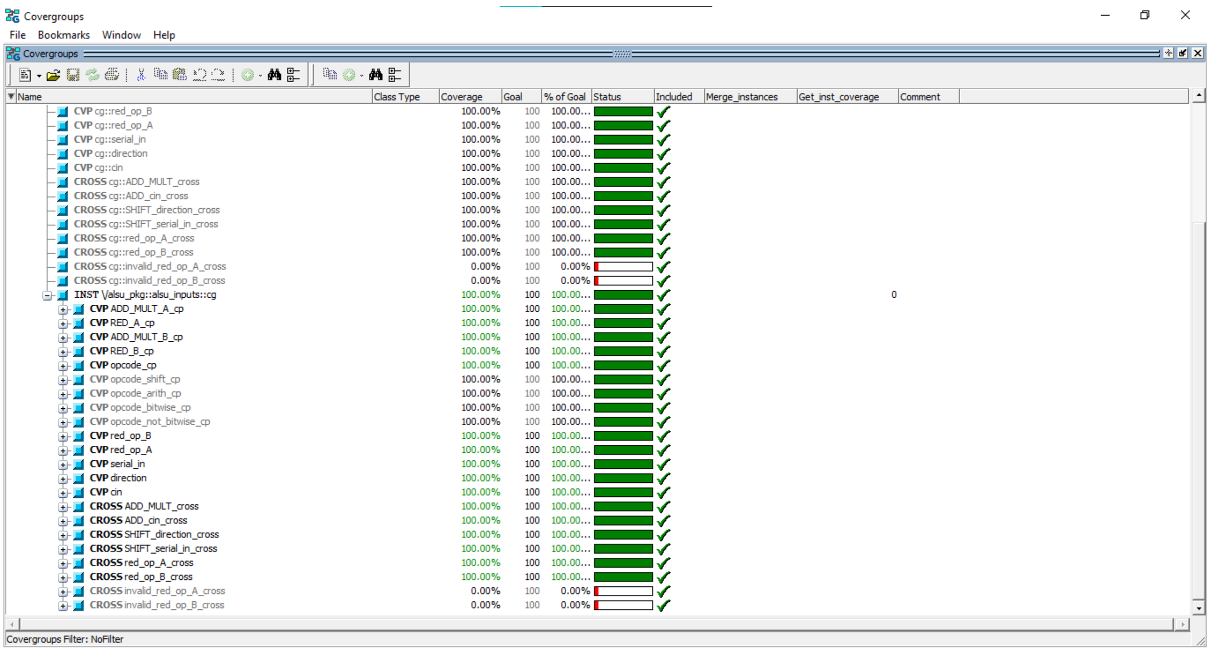
1.6. Bugs:

No bugs detected

1.7. 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%
```

1.8. Functional coverage:



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
CVP cg::red_op_B		100.00%	100	100.00...	🟩	✓			
CVP cg::red_op_A		100.00%	100	100.00...	🟩	✓			
CVP cg::serial_in		100.00%	100	100.00...	🟩	✓			
CVP cg::direction		100.00%	100	100.00...	🟩	✓			
CVP cg::cin		100.00%	100	100.00...	🟩	✓			
CROSS cg::ADD_MULT_cross		100.00%	100	100.00...	🟩	✓			
CROSS cg::ADD_cin_cross		100.00%	100	100.00...	🟩	✓			
CROSS cg::SHIFT_direction_cross		100.00%	100	100.00...	🟩	✓			
CROSS cg::SHIFT_serial_in_cross		100.00%	100	100.00...	🟩	✓			
CROSS cg::red_op_A_cross		100.00%	100	100.00...	🟩	✓			
CROSS cg::red_op_B_cross		100.00%	100	100.00...	🟩	✓			
CROSS cg::invalid_red_op_A_cross		0.00%	100	0.00%	🟥	✓			
CROSS cg::invalid_red_op_B_cross		0.00%	100	0.00%	🟥	✓			
INST \/alsu_pkg::alsu_inputs::cg		100.00%	100	100.00...	🟩	✓		0	
CVP ADD_MULT_A_cp		100.00%	100	100.00...	🟩	✓			
CVP RED_A_cp		100.00%	100	100.00...	🟩	✓			
CVP ADD_MULT_B_cp		100.00%	100	100.00...	🟩	✓			
CVP RED_B_cp		100.00%	100	100.00...	🟩	✓			
CVP opcode_cp		100.00%	100	100.00...	🟩	✓			
CVP opcode_shift_cp		100.00%	100	100.00...	🟩	✓			
CVP opcode_arith_cp		100.00%	100	100.00...	🟩	✓			
CVP opcode_bitwise_cp		100.00%	100	100.00...	🟩	✓			
CVP opcode_not_bitwise_cp		100.00%	100	100.00...	🟩	✓			
CVP red_op_B		100.00%	100	100.00...	🟩	✓			
CVP red_op_A		100.00%	100	100.00...	🟩	✓			
CVP serial_in		100.00%	100	100.00...	🟩	✓			
CVP direction		100.00%	100	100.00...	🟩	✓			
CVP cin		100.00%	100	100.00...	🟩	✓			
CROSS ADD_MULT_cross		100.00%	100	100.00...	🟩	✓			
CROSS ADD_cin_cross		100.00%	100	100.00...	🟩	✓			
CROSS SHIFT_direction_cross		100.00%	100	100.00...	🟩	✓			
CROSS SHIFT_serial_in_cross		100.00%	100	100.00...	🟩	✓			
CROSS red_op_A_cross		100.00%	100	100.00...	🟩	✓			
CROSS red_op_B_cross		100.00%	100	100.00...	🟩	✓			
CROSS invalid_red_op_A_cross		0.00%	100	0.00%	🟥	✓			
CROSS invalid_red_op_B_cross		0.00%	100	0.00%	🟥	✓			

Covergroups Filter: NoFilter

2.

2.1.

```
assert property (@(posedge clk) a |-> ##2 b);
```

2.2.

```
assert property (@(posedge clk) (a && b) |-> ##[1:3] c);
```

2.3.

```
sequence s11b;
    ##2 (~b);
endsequence
```

2.4.

2.4.1.

```
property p_decoder;
    @(posedge clk) $onehot(Y);
endproperty
```

2.4.2.

```
property p_valid;
    @(posedge clk) ($countones(D) == 0) |=> (~valid);
Endproperty
```

3.

3.1. Interface file:

```
interface counter_if (clk);
    parameter WIDTH = 4;
    input 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;

    modport DUT (
        input clk, rst_n, load_n, up_down, ce, data_load,
        output count_out, max_count, zero
    );

    modport TEST (
        input clk, count_out, max_count, zero,
        output rst_n, load_n, up_down, ce, data_load
    );
endinterface : counter_if
```

3.2. Design file:

```
module counter (counter_if.DUT c_if);

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

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

endmodule
```

3.3. 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 then randomized with constraint.	-	Immediate assertion to check count_out == 0.
COUNTER1	When load_n is low, data_load should be loaded into count_out.	Randomized with constraint.	Cover all values of data_load	Concurrent assertion to verify count_out == data_load when load_n == 0.
COUNTER2	When ce = 1, up_down = 1, counter increments each clock cycle.	Randomized with constraint.	Cover all values of count_out, and transition bins from max to zero	Concurrent assertion to verify correct counting sequence and max_count assertion at max.
COUNTER3	When ce = 1, up_down = 0, counter decrements each clock cycle.	Randomized with constraint.	Cover all values of count_out, and transition bins from zero to max	Concurrent assertion to verify correct counting sequence and zero assertion at 0.
COUNTER4	When ce = 0, counter does not change	-	-	Concurrent assertion to verify count_out doesn’t change.
COUNTER5	When count_out == 0, zero flag is high.	-	-	Concurrent assertion to verify zero == 1 once count_out == 0.

Label	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
COUNTER6	When count_out is max, max flag is high	-	-	Concurrent assertion to verify max_count == 1 once count_out == max.

3.4. Package file:

```
package counter_pkg;

    parameter int WIDTH = 4;

    class counter_txn;
        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 cg();
            load_n_cp: coverpoint load_n iff (rst_n) {
                option.weight = 0;
            }

            data_load_cp: coverpoint data_load {
                option.weight = 0;
            }

            data_load_cross: cross load_n_cp, data_load_cp {
                ignore_bins load_n_1 = binsof(load_n_cp) intersect {1};
            }

            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();
            cg = new();
        endfunction

    endclass : counter_txn

endpackage : counter_pkg
```

3.5. Testbench file:

```
import counter_pkg::*;

module counter_tb (counter_if.TEST c_if);
    int NUM_TEST_CASES = 500;

    counter_txn txn = new();

    initial begin
        $display("Starting Counter testbench...");
        assert_reset();

        repeat(NUM_TEST_CASES - 1) begin
            assert(txn.randomize());
            c_if.rst_n = txn.rst_n;
            c_if.load_n = txn.load_n;
            c_if.up_down = txn.up_down;
            c_if.ce = txn.ce;
            c_if.data_load = txn.data_load;
            @(negedge c_if.clk);
            txn.count_out = c_if.count_out;
            txn.max_count = c_if.max_count;
            txn.zero = c_if.zero;
            txn.cg.sample();
        end

        // Display completion message
        $display("Simulation Completed: %0d test cases executed.",
NUM_TEST_CASES);
        $stop;
    end

    task assert_reset();
        c_if.rst_n = 1;
        @(negedge c_if.clk);
        c_if.rst_n = 0;
        @(negedge c_if.clk);
        c_if.rst_n = 1;
    endtask

endmodule
```

3.6. Assertions file:

```
module counter_sva (counter_if.DUT c_if);
    always_comb begin
        if(~c_if.rst_n) begin
            assert_reset: assert final(c_if.count_out == 0);
        end
    end

    property load_p;
        @(posedge c_if.clk) disable iff (~c_if.rst_n) (~c_if.load_n) |=>
(c_if.count_out == $past(c_if.data_load));
    endproperty

    property count_stable_p;
        @(posedge c_if.clk) disable iff (~c_if.rst_n) (c_if.load_n &&
~c_if.ce) |=> (c_if.count_out == $past(c_if.count_out));
    endproperty

    property inc_count_p;
        @(posedge c_if.clk) disable iff (~c_if.rst_n) (c_if.load_n && c_if.ce
&& c_if.up_down) |=> (c_if.count_out == ($past(c_if.count_out) + 1'b1));
    endproperty

    property dec_count_p;
        @(posedge c_if.clk) disable iff (~c_if.rst_n) (c_if.load_n && c_if.ce
&& ~c_if.up_down) |=> (c_if.count_out == ($past(c_if.count_out) - 1'b1));
    endproperty

    property zero_p;
        @(posedge c_if.clk) disable iff (~c_if.rst_n) (c_if.count_out ==
{c_if.WIDTH{1'b0}}) |-> (c_if.zero == 1'b1);
    endproperty

    property max_p;
        @(posedge c_if.clk) disable iff (~c_if.rst_n) (c_if.count_out ==
{c_if.WIDTH{1'b1}}) |-> (c_if.max_count == 1'b1)
    endproperty

    assert_load_p: assert property(load_p);
    assert_count_stable_p: assert property(count_stable_p);
    assert_inc_count_p: assert property(inc_count_p);
    assert_dec_count_p: assert property(dec_count_p);
    assert_zero_p: assert property(zero_p);
    assert_max_p: assert property(max_p);

    cover_load_p: assert property(load_p);
    cover_count_stable_p: assert property(count_stable_p);
    cover_inc_count_p: assert property(inc_count_p);
    cover_dec_count_p: assert property(dec_count_p);
    cover_zero_p: assert property(zero_p);
    cover_max_p: assert property(max_p);
endmodule
```

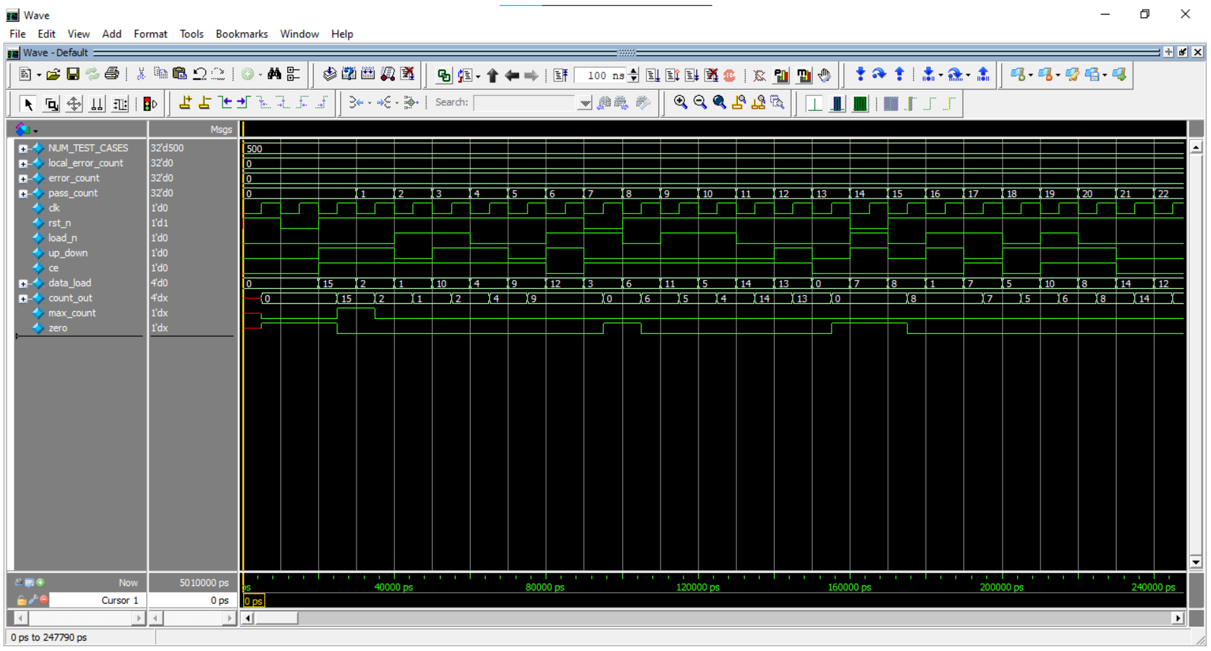
3.7. Top file:

```
`timescale 1ns / 1ps

module counter_top();
    bit clk;

    initial begin
        forever #5 clk = ~clk;
    end

    counter_if c_if(clk);
    counter DUT(c_if);
    counter_tb TEST(c_if);
    bind counter counter_sva ASSERT(c_if);
endmodule
```

3.8. Do file:

```
vlog counter_if.sv
vlog counter_pkg.sv
vlog counter_sva.sv
vlog counter_tb.sv
vlog counter.sv counter_top.sv +cover -covercells
vsim -voptargs=+acc work.counter_top -cover
add wave -position insertpoint sim:/counter_top/c_if/*
coverage save counter_tb.ucdb -onexit
run -all
```

3.9. Waveform snippet:



3.10. Transcript:

```
# Starting Counter testbench...
# Simulation Completed: 500 test cases executed.
```

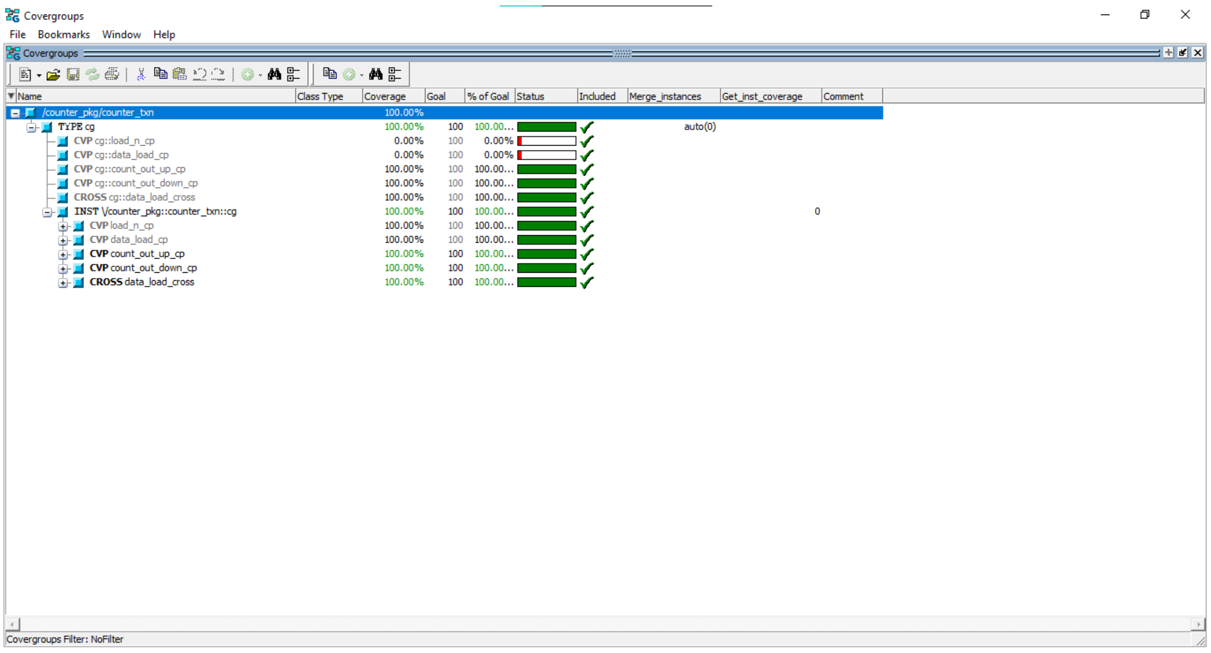
3.11. Bugs:

No bugs detected.

3.12. 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%
```

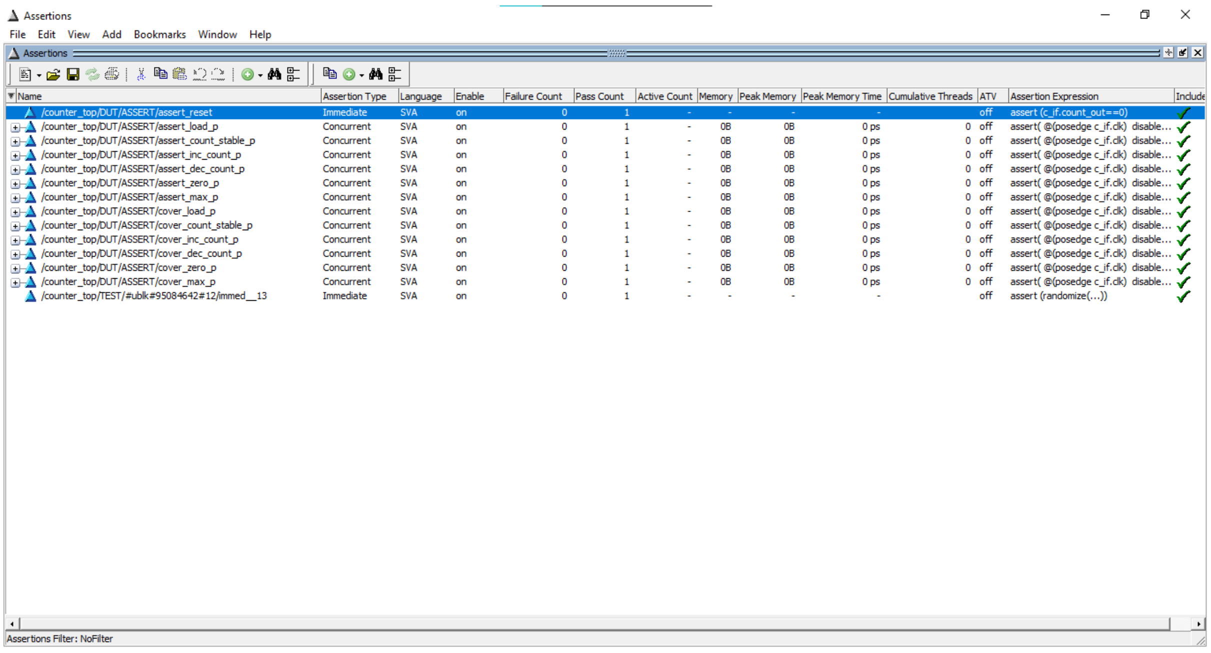
3.13. Functional coverage:



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/counter_pkg/counter_txn		100.00%							
TYPE cg		100.00%	100	100.00...	🟩	✓	auto(0)		
CVP cg::load_n_cp		0.00%	100	0.00%	🟥	✓			
CVP cg::data_load_cp		0.00%	100	0.00%	🟥	✓			
CVP cg::count_out_up_cp		100.00%	100	100.00...	🟩	✓			
CVP cg::count_out_down_cp		100.00%	100	100.00...	🟩	✓			
CROSS cg::data_load_cross		100.00%	100	100.00...	🟩	✓			
INST \/counter_pkg::counter_txn::cg		100.00%	100	100.00...	🟩	✓		0	
CVP load_n_cp		100.00%	100	100.00...	🟩	✓			
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...	🟩	✓			
CROSS data_load_cross		100.00%	100	100.00...	🟩	✓			

Covergroups Filter: NoFilter

3.14. Assertions coverage:


Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Include
/counter_top/DUT/ASSERT/assert_reset	Immediate	SVA	on	0	1	-	-	-	-		off	assert (c_if.count_out==0)	✓
/counter_top/DUT/ASSERT/assert_load_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/assert_count_stable_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/assert_inc_count_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/assert_dec_count_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/assert_zero_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/assert_max_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/cover_load_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/cover_count_stable_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/cover_inc_count_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/cover_dec_count_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/cover_zero_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/DUT/ASSERT/cover_max_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge c_if.clk) disable...	✓
/counter_top/TEST/#ublk#95084642#12/immed__13	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓

Assertions Filter: NoFilter

4.

4.1. Verification plan:

Label	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
CONFIG1	When reset is asserted high, the reset value of the specified register will be available on the output.	Directed at the start of simulation and throughout the simulation.	-	A checker to make sure output is correct.
CONFIG2	Write 16’hFFFF to all register and read them.	Directed to all the registers.	-	A checker to make sure output is correct.
CONFIG3	Write walking ones to all register and read them.	Directed to all the registers.	-	A checker to make sure output is correct.

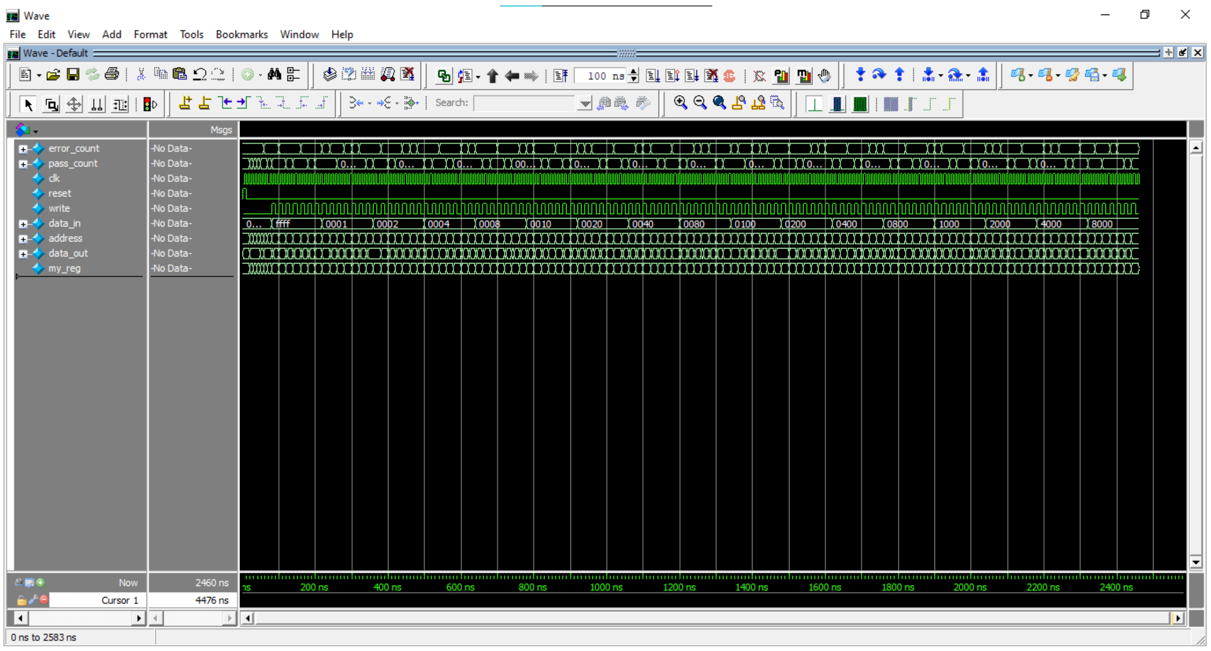
4.2. Testbench file:

```
module config_reg_tb ();
    // TESTBENCH VARIABLES
    int error_count = 0;
    int pass_count = 0;

    // DUT
    bit clk;
    bit reset;
    bit write;
    bit [15:0] data_in;
    bit [2:0] address;
    logic [15:0] data_out;

    // TEST
    // Associative array, as a golden model for the reset values of the
registers.
    bit [15:0] assert_assoc [string];

    // A user defined enumerated type that hold the values of the registers
    typedef enum {
        ADC0_REG,
        ADC1_REG,
        TEMP_SENSOR0_REG,
        TEMP_SENSOR1_REG,
        ANALOG_TEST,
        DIGITAL_TEST,
        AMP_GAIN,
        DIGITAL_CONFIG
    } reg_t;

    // A variable from the enum. This variable will be assigned to the address
connected to the DUT
    reg_t my_reg;

    config_reg DUT (clk, reset, write, data_in, address, data_out);

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

    initial begin
        $display("Starting CONFIG_REG testbench...");
        assert_assoc["ADC0_REG"] = 16'hFFFF;
        assert_assoc["ADC1_REG"] = 16'h0;
        assert_assoc["TEMP_SENSOR0_REG"] = 16'h0;
        assert_assoc["TEMP_SENSOR1_REG"] = 16'h0;
        assert_assoc["ANALOG_TEST"] = 16'hABCD;
        assert_assoc["DIGITAL_TEST"] = 16'h0;
        assert_assoc["AMP_GAIN"] = 16'h0;
        assert_assoc["DIGITAL_CONFIG"] = 16'h1;

        $display("Check reset values...");
        assert_reset();
        check_reset_values();

        $display("Write 16'hFFFF to each register then read...");
        write_value_then_check(16'hFFFF);

        $display("Check reset values...");
        assert_reset();
        check_reset_values();

        $display("Write walking ones to each register then read...");
        for (int i = 0; i < 16; i++) begin
            write_value_then_check(1 << i);
        end

        $display("Simulation Completed");
        $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();
        reset = 1;
        wait_cycles(1);
        reset = 0;
    endtask

    task check16Bits(input string reg_name, input bit [15:0] expected_data,
input logic [15:0] read_data);
        if (read_data !== expected_data) begin
            error_count++;
            $error("[MISMATCH] read from register %s: Expected: 0x%0h, Got:
0x%0h", reg_name, expected_data, read_data);
        end else begin
            pass_count++;
            $info("[MATCH] read from register %s: Expected: 0x%0h, Got: 
0x%0h", reg_name, expected_data, read_data);
        end
    endtask

    task check_reset_values();
        for (int i = 0; i < my_reg.last + 1; i++) begin
            address = my_reg;
            wait_cycles(1);
            check16Bits(my_reg.name, assert_assoc[my_reg.name], data_out);
            my_reg = my_reg.next;
        end
    endtask

    task write_value_then_check(input bit [15:0] write_data);
        data_in = write_data;
        for (int i = 0; i < my_reg.last + 1; i++) begin
            write = 1;
            address = my_reg;
            wait_cycles(1);
            write = 0;
            wait_cycles(1);
            check16Bits(my_reg.name, write_data, data_out);
            my_reg = my_reg.next;
        end
    endtask
endmodule
```

4.3. Do file:

```
vlog config_reg_buggy_questa.svp
vlog config_reg_tb.sv
vsim -voptargs=+acc work.config_reg_tb
add wave *
run -all
# quit -sim
```

4.4. Waveform snippet:


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
error_count -No Data-
pass_count -No Data-
clk -No Data-
reset -No Data-
write -No Data-
data_in -No Data-
address -No Data-
data_out -No Data-
my_reg -No Data-
ffff 0001 0002 0004 0008 0010 0020 0040 0080 0100 0200 0400 0800 1000 2000 4000 8000
Now 2460 ns
Cursor 1 4476 ns
200 ns 400 ns 600 ns 800 ns 1000 ns 1200 ns 1400 ns 1600 ns 1800 ns 2000 ns 2200 ns 2400 ns
0 ns to 2583 ns

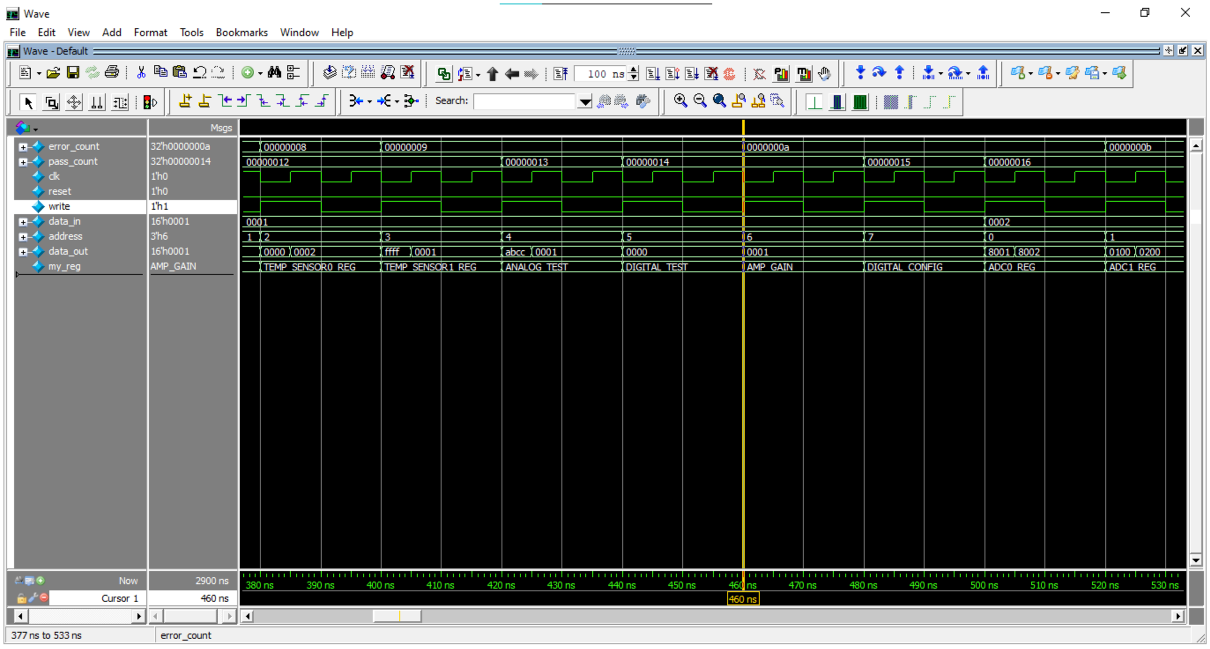
4.5. Transcript:

```
# Starting CONFIG_REG testbench...
...
# Simulation Completed
# Test Summary: Passed = 82, Failed = 70
```

4.6. Bugs report:

```
1. ADC0_REG:
Design Input for Bug to Appear:  Writing/reading walking 1’s.
Expected Behavior: Read 0001
Observed Behavior: Read 8001

# Write walking ones to each register then read...
# ** Error: [MISMATCH] read from register ADC0_REG: Expected: 0x1, Got: 0x8001

2. ADC1_REG:
Design Input for Bug to Appear:  Writing/reading walking 1’s.
Expected Behavior: Read 0001
Observed Behavior: Read 0100

# Write walking ones to each register then read...
# ** Error: [MISMATCH] read from register ADC1_REG: Expected: 0x1, Got: 0x100

3. TEMP_SENSOR0_REG:
Design Input for Bug to Appear:  Write FFFF to all registers. Then read
Expected Behavior: Read FFFF
Observed Behavior: Read FFFE

# Write 16'hFFFF to each register then read...
# ** Error: [MISMATCH] read from register TEMP_SENSOR0_REG: Expected: 0xffff,
Got: 0xfffe

4. TEMP_SENSOR1_REG:
Design Input for Bug to Appear:  Write FFFF to all registers. Then reset
Expected Behavior: Read 0000
Observed Behavior: Read FFFF

# Check reset values after writing 16’hFFFF...
# ** Error: [MISMATCH] read from register TEMP_SENSOR1_REG: Expected: 0x0,
Got: 0xffff

5. ANALOG_TEST:
Design Input for Bug to Appear:  Reset
Expected Behavior: Read ABCD
Observed Behavior: Read ABCC

# Check reset values...
# ** Error: [MISMATCH] read from register ANALOG_TEST: Expected: 0xabcd, Got:
0xabcc
```

```
6. DIGITAL_TEST:
Design Input for Bug to Appear:  Write FFFF to all registers. Then read.
Expected Behavior: Read FFFF
Observed Behavior: Read 0000

# Write 16'hFFFF to each register then read...
# ** Error: [MISMATCH] read from register DIGITAL_TEST: Expected: 0xffff, Got: 
0x0

7. AMP_GAIN:
Design Input for Bug to Appear:  Writing/reading walking 1’s.
Expected Behavior: Read 0001 after writing the data in value
Observed Behavior: Read 0001 before writing the data in as per the reset value
```



```
error_count   32'h0000000a
pass_count    32'h00000014
clk           1'h0
reset         1'h0
write         1'h1
data_in       16'h0001
address       3'h6
data_out      16'h0001
my_reg        AMP_GAIN
```

```
8. DIGITAL_CONFIG:
Design Input for Bug to Appear:  Write FFFF to all registers. Then read
Expected Behavior: Read FFFF
Observed Behavior: Read 7FFF

# Write 16'hFFFF to each register then read...
# ** Error: [MISMATCH] read from register DIGITAL_CONFIG: Expected: 0xffff, 
Got: 0x7fff
```