UVM

Assignment6_extra

By: Mohamed Sayed Mohamed Soliman

#part 1

1- ALSU CODE Design

```
module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input [2:0] opcode;
input signed [2:0] A, B;    // first bug --> we must put it signed
output reg [15:0] leds;
output reg signed [5:0] out;  // second bug --> we must put it signed
reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_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 (invalid)
        out <= 0;
    else 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 begin
        case (opcode_reg) //  third bug is to used the opcode_reg  not the opcode
          3'h0: begin
            if (red_op_A_reg && red_op_B_reg)
              out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;  // third bug is to replace AND with OR
            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;  // fourth bug is to replace OR with XOR
            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:begin
                if(ALSUif.FULL_ADDER == "ON")  // fifth bug to add Cin operation in case of full adde
                ALSUif.out <= A_reg + B_reg + cin_reg ;
                else
                ALSUif.out <= A_reg + B_reg ;
                end
          3'h3: ALSUif.out <= A_reg * B_reg;
          3'h4: begin
            ALSUif.out <= ALSUif.out_shift_reg ;
          end
          3'h5: begin
            ALSUif.out <= ALSUif.out_shift_reg ;

          end
        default : ALSUif.out <= 0 ;
         endcase
     end
   end
 end

 endmodule
```

2- Shift code design

```
module shift_reg (shift_reg_if.DUT_shift shift_if );

always @(*) begin
        if (shift_if.mode) // rotate
            if (shift_if.direction) // left
            shift_if.dataout <= {shift_if.datain[4:0], shift_if.datain[5]};
            else
            shift_if.dataout <= {shift_if.datain[0], shift_if.datain[5:1]};
        else // shift
            if (shift_if.direction) // left
            shift_if.dataout <= {shift_if.datain[4:0], shift_if.serial_in};
            else
            shift_if.dataout <= {shift_if.serial_in, shift_if.datain[5:1]};
end
endmodule
```

3- ALSU Interface

```
interface ALSU_if (clk);
  input clk ;
  logic  rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
  logic [2:0] opcode;
  logic signed [2:0] A, B;
  logic  [15:0] leds;
  logic  signed [5:0] out;

  modport DUT (input clk , rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in , opcode , A , B , output leds , out ) ;

endinterface
```

4- Shift Interface

```
interface shift_reg_if ();
  logic serial_in, direction, mode;
  logic signed [5:0] datain, dataout;


  modport DUT_shift (input  direction, serial_in , mode , datain , output dataout ) ;
endinterface : shift_reg_if
```

5- top module

```
import ALSU_shift_test_pkg::*;
import uvm_pkg::* ;
`include "uvm_macros.svh"

module ALSU_shift_top();
bit clk ;

initial
  begin
    clk = 0 ;
    forever
    #1 clk = ~clk ;
  end
  ALSU_if ALSUif (clk) ;
  shift_reg_if shift_if() ;
  ALSU dut_AL (ALSUif) ;
  shift_reg DUT_SH ( shift_if ) ;

  assign shift_if.serial_in = ALSUif.serial_in ;
  assign shift_if.direction = ALSUif.direction ;
  assign shift_if.datain = ALSUif.out ;
  assign shift_if.mode = ALSUif.opcode[0] ;
  assign ALSUif.out_shift_reg = shift_if.dataout ;
  bind ALSU  SVA  sva(ALSUif) ;
  initial
    begin
        uvm_config_db#(virtual ALSU_if)::set(null , "uvm_test_top" , "ALSUif" , ALSUif );
        uvm_config_db#(virtual shift_reg_if)::set(null , "uvm_test_top" , "SHIFT_IF" , shift_if );
        run_test("test");
    end


endmodule
```

6- alsu_test

```
package ALSU_shift_test_pkg ;
  import ALSU_env_pkg::*;
  import shift_reg_env_pkg::*;
  import ALSU_config_pkg::*;
  import shift_reg_config_pkg::*;
  import ALSU_seq_reset_pkg::*;
  import ALSU_seq_main_pkg::*;
  import uvm_pkg::* ;
  `include "uvm_macros.svh"

  class test extends uvm_test;
      `uvm_component_utils(test)
    ALSU_env  AL_env ;
    shift_reg_env  shift_env ;
    ALSU_config  alsu_config_obj_test ;
    shift_reg_config  shift_cfg ;
    ALSU_reset_sequence reset_sequence  ;
    ALSU_main_sequence  main_sequence ;

     function new(string name = "test" , uvm_component parent = null );
        super.new(name ,parent ) ;
     endfunction

     function void build_phase(uvm_phase phase) ;
      super.build_phase(phase);
      AL_env = ALSU_env::type_id::create("AL_env",this );
      shift_env = shift_reg_env::type_id::create("shift_env",this );
      alsu_config_obj_test = ALSU_config::type_id::create("alsu_config_obj_test");
      shift_cfg = shift_reg_config::type_id::create("shift_cfg");
      main_sequence = ALSU_main_sequence::type_id::create("main_sequence" );
      reset_sequence = ALSU_reset_sequence::type_id::create("reset_sequence");
      if(!uvm_config_db#(virtual ALSU_if)::get (this , "" , "ALSUif" , alsu_config_obj_test.alsu_config_vif ))
        `uvm_fatal("run_phase" , "test - unable to get the virtual interface ") ;
      uvm_config_db#(ALSU_config)::set (this , "*" , "CFG" , alsu_config_obj_test ) ;

        if(!uvm_config_db#(virtual shift_reg_if)::get (this , "" , "SHIFT_IF" , shift_cfg.shift_vif ))
        `uvm_fatal("run_phase" , "test - unable to get the virtual interface ") ;
      uvm_config_db#(shift_reg_config)::set (this , "*" , "CFG" , shift_cfg ) ;
      alsu_config_obj_test.is_active = UVM_ACTIVE ;
      shift_cfg.is_active = UVM_PASSIVE ;
     endfunction

     task run_phase(uvm_phase phase) ;
      super.run_phase(phase);
      phase.raise_objection(this);
      `uvm_info("run_phase","reset assert" ,  UVM_LOW)
      reset_sequence.start(AL_env.agt.sqr);
      `uvm_info("run_phase","reset deassert" ,  UVM_LOW)

      `uvm_info("run_phase","stimulus generation started" ,  UVM_LOW)
      main_sequence.start(AL_env.agt.sqr);
      `uvm_info("run_phase","stimulus generation ended" ,  UVM_LOW)
      phase.drop_objection(this);
     endtask:run_phase

    endclass
  endpackage
```

7- alsu_env

```
package ALSU_env_pkg ;

  import ALSU_agent_pkg::*;
   import ALSU_scoreboard_pkg::*;
   import ALSU_coverage_pkg::*;
   import uvm_pkg::* ;
   `include "uvm_macros.svh"


   class ALSU_env extends uvm_env;
      `uvm_component_utils(ALSU_env)

      ALSU_agent  agt ;
      ALSU_scoreboard sb ;
      ALSU_coverage cov ;

      function new(string name = "ALSU_env" , uvm_component parent = null );
         super.new(name ,parent ) ;
      endfunction

      function void build_phase(uvm_phase phase) ;
         super.build_phase(phase);
        agt = ALSU_agent::type_id::create("agt",this );
         sb = ALSU_scoreboard::type_id::create("sb",this );
         cov = ALSU_coverage::type_id::create("cov",this );
      endfunction


      function void connect_phase(uvm_phase phase) ;
         agt.agt_ap.connect(sb.sb_export);
         agt.agt_ap.connect(cov.cov_export);
      endfunction

   endclass
endpackage
```

8- Shift env

```
C: > Users > CS > Downloads > Karem Wasem Diploma > session6_Assignmen_extra > ALSO_shift > shift_reg_env.sv
package shift_reg_env_pkg ;

   import shift_reg_agent_pkg::*;
   import shift_reg_scoreboard_pkg::*;
   import shift_reg_coverage_pkg::*;
   import uvm_pkg::* ;
   `include "uvm_macros.svh"


   class shift_reg_env extends uvm_env;
      `uvm_component_utils(shift_reg_env)

      shift_reg_agent  agt ;
      shift_reg_scoreboard sb ;
      shift_reg_coverage cov ;
      function new(string name = "shift_reg_env" , uvm_component parent = null );
         super.new(name ,parent ) ;
      endfunction

      function void build_phase(uvm_phase phase) ;
         super.build_phase(phase);
         agt = shift_reg_agent::type_id::create("agt",this );
         sb = shift_reg_scoreboard::type_id::create("sb",this );
         cov = shift_reg_coverage::type_id::create("cov",this );
      endfunction

      function void connect_phase(uvm_phase phase) ;
         agt.agt_ap.connect(sb.sb_export);
         agt.agt_ap.connect(cov.cov_export);
      endfunction
   endclass
endpackage
```

9- reset sequence

```
package ALSU_seq_reset_pkg ;

    import ALSU_seq_item_pkg::*;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_reset_sequence extends uvm_sequence #(ALSU_seq_item);
        `uvm_object_utils(ALSU_reset_sequence)

        ALSU_seq_item  seq_item ;
        function new(string name = "ALSU_reset_sequence" );
            super.new(name) ;
        endfunction
        task body;
            seq_item = ALSU_seq_item::type_id::create("seq_item");
            start_item(seq_item);
            seq_item.rst = 1 ;
            seq_item.red_op_A = 0 ;
            seq_item.red_op_B = 0 ;
            seq_item.bypass_A = 0 ;
            seq_item.bypass_B = 0 ;
            seq_item.direction = 0 ;
            seq_item.serial_in = 0 ;
            seq_item.opcode = 0 ;
            seq_item.A = 0 ;
            seq_item.B = 0 ;
            seq_item.cin = 0 ;
            finish_item(seq_item);

        endtask
    endclass
endpackage
```

10- Main sequence

```
package ALSU_seq_main_pkg;

    import ALSU_seq_item_pkg::*;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_main_sequence extends uvm_sequence #(ALSU_seq_item);
        `uvm_object_utils(ALSU_main_sequence)

        ALSU_seq_item  seq_item ;
        function new(string name = "ALSU_main_sequence" );
            super.new(name) ;
        endfunction
        task body;
            repeat(100000)
            begin
                seq_item = ALSU_seq_item::type_id::create("seq_item");
                start_item(seq_item);
                assert(seq_item.randomize()) ;
                finish_item(seq_item);
            end

        endtask
    endclass
endpackage
```

11- ALSU agent

```
package ALSU_agent_pkg ;

   import ALSU_driver_pkg::*;
   import ALSU_sequencer_pkg::*;
   import ALSU_monitor_pkg::*;
   import ALSU_config_pkg::*;
   import ALSU_seq_item_pkg::* ;
   import uvm_pkg::* ;
   `include "uvm_macros.svh"


   class ALSU_agent extends uvm_agent;
      `uvm_component_utils(ALSU_agent)

    ALSU_driver  driver ;
      ALSU_sequencer sqr ;
      ALSU_monitor mon  ;
      ALSU_config  ALSU_cfg ;
      uvm_analysis_port #(ALSU_seq_item) agt_ap ;

      function new(string name = "ALSU_agent" , uvm_component parent = null );
         super.new(name ,parent ) ;
      endfunction
```

```

      function void build_phase(uvm_phase phase) ;
         super.build_phase(phase);
         if(!uvm_config_db#(ALSU_config)::get (this , "" , "CFG" , ALSU_cfg ) )
         `uvm_fatal("build_phase" , "driver - unable to get the virtual interface ") ;

         driver = ALSU_driver::type_id::create("driver",this );
         sqr = ALSU_sequencer::type_id::create("sqr",this );
         mon = ALSU_monitor::type_id::create("mon",this );
         agt_ap = new("agt_ap" , this) ;
      endfunction

      function void connect_phase(uvm_phase phase) ;
        driver.alsu_driver_vif = ALSU_cfg.alsu_config_vif ;
         mon.ALSU_vif = ALSU_cfg.alsu_config_vif ;
         driver.seq_item_port.connect(sqr.seq_item_export) ;
         mon.mon_ap.connect(agt_ap);
      endfunction
   endclass
endpackage
```

12- Shift agent

```
package shift_reg_agent_pkg ;

    import shift_reg_driver_pkg::*;
    import shift_reg_sequencer_pkg::*;
    import shift_reg_monitor_pkg::*;
    import shift_reg_config_pkg::*;
    import shift_reg_seq_item_pkg::* ;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"


    class shift_reg_agent extends uvm_agent;
        `uvm_component_utils(shift_reg_agent)

        shift_reg_driver  shift_driver ;
        shift_reg_sequencer sqr ;
        shift_reg_monitor mon  ;
        shift_reg_config  shift_cfg ;
        uvm_analysis_port #(shift_reg_seq_item) agt_ap ;

        function new(string name = "shift_reg_agent" , uvm_component parent = null );
            super.new(name ,parent ) ;
        endfunction

        function void build_phase(uvm_phase phase) ;
            super.build_phase(phase);
            if(!uvm_config_db#(shift_reg_config)::get (this , "" , "CFG" , shift_cfg ) )
            `uvm_fatal("build_phase" , "driver - unable to get the virtual interface ") ;
            shift_driver = shift_reg_driver::type_id::create("shift_driver",this );
            sqr = shift_reg_sequencer::type_id::create("sqr",this );
            mon = shift_reg_monitor::type_id::create("mon",this );
            agt_ap = new("agt_ap" , this) ;
        endfunction

        function void connect_phase(uvm_phase phase) ;
            shift_driver.shift_vif = shift_cfg.shift_vif ;
            mon.shift_vif = shift_cfg.shift_vif ;
            shift_driver.seq_item_port.connect(sqr.seq_item_export) ;
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

13- Shift scoreboard

```
package shift_reg_scoreboard_pkg ;
    import shift_reg_seq_item_pkg::* ;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"
    class shift_reg_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(shift_reg_scoreboard)
        uvm_analysis_export #(shift_reg_seq_item) sb_export ;
        uvm_tlm_analysis_fifo #(shift_reg_seq_item) sb_fifo ;
        shift_reg_seq_item seq_item_cov ;
        logic [5:0] dataout_ref ;
        int error_count = 0 ;
        int correct_count = 0 ;

        function new(string name = "shift_reg_scoreboard" , uvm_component parent = null );
            super.new(name ,parent ) ;
        endfunction

        function void build_phase(uvm_phase phase) ;
            super.build_phase(phase);
            sb_export =  new("sb_export" , this) ;
            sb_fifo =  new("sb_fifo" , this) ;
        endfunction

        function void connect_phase(uvm_phase phase) ;
            sb_export.connect(sb_fifo.analysis_export);
        endfunction
```

```
    task run_phase(uvm_phase phase) ;
        super.run_phase(phase);
        forever
        begin
            sb_fifo.get(seq_item_cov) ;
            ref_model(seq_item_cov) ;
            if(seq_item_cov.dataout != dataout_ref )
            begin
                `uvm_error("run_phase" , $sformatf("comparsion failed trasnsaction received by the dut %s shile the reference out %ob"\
                ,seq_item_cov.convert2string , dataout_ref )) ;
                error_count++ ;
            end
            else
            begin
                `uvm_info("run_phase" ,seq_item_cov.convert2string_stimulus() , UVM_HIGH ) ;
                correct_count++ ;
            end
        end
    endtask
```

```
        task ref_model(shift_reg_seq_item seq_item_cov);
         if (seq_item_cov.mode) // rotate
            if (seq_item_cov.direction) // left
            dataout_ref <= {seq_item_cov.datain[4:0], seq_item_cov.datain[5]};
            else
            dataout_ref <= {seq_item_cov.datain[0], seq_item_cov.datain[5:1]};
         else // shift
            if (seq_item_cov.direction) // left
            dataout_ref <= {seq_item_cov.datain[4:0], seq_item_cov.serial_in};
            else
            dataout_ref <= {seq_item_cov.serial_in, seq_item_cov.datain[5:1]};
        endtask

        function void report_phase(uvm_phase phase) ;
         super.report_phase(phase);
         `uvm_info("report_phase" ,$sformatf("total successful %0d  " ,correct_count ) , UVM_MEDIUM ) ;
         `uvm_info("report_phase" ,$sformatf("total FAILED %0d  " ,error_count ) , UVM_MEDIUM ) ;
      endfunction
    endclass
  endpackage
```

14- ALSU scoreboard

```
package ALSU_scoreboard_pkg ;
   import ALSU_seq_item_pkg::* ;
   import uvm_pkg::* ;
   `include "uvm_macros.svh"
   class ALSU_scoreboard extends uvm_scoreboard;
      `uvm_component_utils(ALSU_scoreboard)
      uvm_analysis_export #(ALSU_seq_item) sb_export ;
      uvm_tlm_analysis_fifo #(ALSU_seq_item) sb_fifo ;
      ALSU_seq_item seq_item_cov ;
      logic [5:0] dataout_ref ;
      logic [15:0] leds_ref ;

        logic cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
        logic [2:0] opcode_reg;
        logic signed  [2:0]  A_reg, B_reg;
        logic invalid_red_op, invalid_opcode, invalid;
      int error_count = 0 ;
      int correct_count = 0 ;
      function new(string name = "ALSU_scoreboard" , uvm_component parent = null );
         super.new(name ,parent ) ;
      endfunction
      function void build_phase(uvm_phase phase) ;
         super.build_phase(phase);
         sb_export =  new("sb_export" , this) ;
         sb_fifo =  new("sb_fifo" , this) ;
      endfunction

      function void connect_phase(uvm_phase phase) ;
         sb_export.connect(sb_fifo.analysis_export);
      endfunction
```

```
task run_phase(uvm_phase phase) ;
   super.run_phase(phase);
   forever
   begin
      sb_fifo.get(seq_item_cov) ;
      ref_model(seq_item_cov) ;
      if(seq_item_cov.out != dataout_ref && seq_item_cov.leds != leds_ref  )
      begin
         `uvm_error("run_phase" , $sformatf("comparsion failed trasnsaction received by the dut %s shile \
          the reference out %ob" ,seq_item_cov.convert2string , dataout_ref )) ;
         error_count++ ;
      end
      else
      begin
         `uvm_info("run_phase" ,seq_item_cov.convert2string_stimulus() , UVM_HIGH ) ;
         correct_count++ ;
      end
   end
  endtask

   task ref_model(ALSU_seq_item seq_item_cov);
      invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
      invalid_opcode = opcode_reg[1] & opcode_reg[2];
      invalid = invalid_red_op | invalid_opcode;
         fork
            begin
               if(seq_item_cov.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 <= seq_item_cov.cin;
                  red_op_B_reg <= seq_item_cov.red_op_B;
                  red_op_A_reg <= seq_item_cov.red_op_A;
                  bypass_B_reg <= seq_item_cov.bypass_B;
                  bypass_A_reg <= seq_item_cov.bypass_A;
                  direction_reg <= seq_item_cov.direction;
                  serial_in_reg <= seq_item_cov.serial_in;
                  opcode_reg <= seq_item_cov.opcode;
                  A_reg <= seq_item_cov.A;
                  B_reg <= seq_item_cov.B;
               end
            end

```

```
begin
    if(seq_item_cov.rst) begin
        dataout_ref <= 0;
    end
    else begin
    if (invalid)
    dataout_ref <= 0;
    else if (bypass_A_reg && bypass_B_reg)
    dataout_ref <=  A_reg;
    else if (bypass_A_reg)
    dataout_ref <= A_reg;
    else if (bypass_B_reg)
    dataout_ref <= B_reg;
    else begin
        case (opcode_reg)
            3'h0: begin
            if (red_op_A_reg && red_op_B_reg)
            dataout_ref <=|A_reg;
            else if (red_op_A_reg)
            dataout_ref <= |A_reg;
            else if (red_op_B_reg)
            dataout_ref <= |B_reg;
            else
            dataout_ref <= A_reg | B_reg;
            end
            3'h1: begin
            if (red_op_A_reg && red_op_B_reg)
            dataout_ref <=  ^A_reg;
            else if (red_op_A_reg)
            dataout_ref <= ^A_reg;
            else if (red_op_B_reg)
            dataout_ref <= ^B_reg;
            else
            dataout_ref <= A_reg ^ B_reg;
            end
            3'h2:begin
                dataout_ref <= A_reg + B_reg + cin_reg ;
                end
            3'h3: dataout_ref <= A_reg * B_reg;
            3'h4: begin
            if (direction_reg)
            dataout_ref <= {dataout_ref[4:0], serial_in_reg};
            else
            dataout_ref <= {serial_in_reg, dataout_ref[5:1]};
            end
            3'h5: begin
            if (direction_reg)
            dataout_ref <= {dataout_ref[4:0], dataout_ref[5]};
            else
            dataout_ref <= {dataout_ref[0], dataout_ref[5:1]};
            end
        default : dataout_ref <= 0 ;
        endcase
    end

    end
end
```

```
                join
                if(seq_item_cov.rst) begin
                    leds_ref <= 0;
                end else begin
                    if (invalid)
                    leds_ref <= ~leds_ref;
                    else
                    leds_ref <= 0;
                end
            endtask

            function void report_phase(uvm_phase phase) ;
             super.report_phase(phase);
             `uvm_info("report_phase" ,$sformatf("total successful %0d  " ,correct_count ) , UVM_MEDIUM ) ;
             `uvm_info("report_phase" ,$sformatf("total FAILED %0d  " ,error_count ) , UVM_MEDIUM ) ;
           endfunction
        endclass
    endpackage
```

15- ALSU coverage

```
package ALSU_coverage_pkg ;

    import ALSU_seq_item_pkg::* ;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_coverage extends uvm_component;
        `uvm_component_utils(ALSU_coverage)

        uvm_analysis_export #(ALSU_seq_item) cov_export ;
        uvm_tlm_analysis_fifo #(ALSU_seq_item) cov_fifo ;
        ALSU_seq_item seq_item_cov ;
        typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
        typedef enum  {Or , Xor , Add , Mult , Shift ,Rotate} opcode_valid_e ;
        localparam MAXPOS = 3  ;
        localparam MAXNEG = -4 ;
        localparam zero = 0 ;
        covergroup cvr_gp  ;
          A_cp : coverpoint seq_item_cov.A {
              bins A_data_0 = {0} ;
              bins A_data_max = {MAXPOS} ;
              bins A_data_min = {MAXNEG} ;
              bins A_data_default = default ;
              bins A_data_walkingones[] = {1, 2, -4} iff (seq_item_cov.red_op_A && !seq_item_cov.red_op_B);
          }

          B_cp : coverpoint seq_item_cov.B {
              bins B_data_0 = {0} ;
              bins B_data_max = {MAXPOS} ;
              bins B_data_min = {MAXNEG} ;
              bins B_data_default = default ;
              bins B_data_walkingones[] = {1, 2, -4} iff (!seq_item_cov.red_op_A && seq_item_cov.red_op_B);
          }
```

```
ALU_cp : coverpoint seq_item_cov.opcode {
    bins Bins_shift[] = {SHIFT , ROTATE} ;
    bins Bins_arith[] = {ADD , MULT} ;
    bins Bins_bitwise[] = {OR , XOR} ;
    illegal_bins Bins_invalid = {INVALID_6 , INVALID_7} ;
}

cin_cp : coverpoint seq_item_cov.cin {
    bins cin_data = {0 , 1} ;
}

direction_cp : coverpoint seq_item_cov.direction {
    bins direction_data = {0 , 1} ;
}

serial_in_cp : coverpoint seq_item_cov.serial_in {
    bins serial_in_data = {0 , 1} ;
}

red_op_A_cp : coverpoint seq_item_cov.red_op_A {
    bins red_op_A_LOW_data = {0} ;
    bins red_op_A_HIGH_data = {1} ;
}
red_op_B_cp : coverpoint seq_item_cov.red_op_B {
    bins red_op_B_LOW_data = {0} ;
    bins red_op_B_HIGH_data = {1} ;
}
```

```
add_mult_cp1 : cross  A_cp , B_cp , ALU_cp
    {

        bins zero_A_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_0) && binsof(B_cp.B_data_0) ;
        bins max_pos_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_max) && binsof(B_cp.B_data_max) ;
        option.cross_auto_bin_max = 0 ;

    }


opcode_cp2 : cross  cin_cp , direction_cp , serial_in_cp , ALU_cp ,red_op_A_cp , red_op_B_cp
    {

        bins cin_add = binsof(cin_cp.cin_data) && binsof(ALU_cp.Bins_arith) intersect {ADD} ;
        bins serialin_shift = binsof(serial_in_cp.serial_in_data) && binsof(ALU_cp) intersect {SHIFT} ;
        bins direction_shift_rota = binsof(direction_cp) && binsof(ALU_cp.Bins_shift) ;
        option.cross_auto_bin_max = 0 ;

    }


or_xor_cp3 : cross  A_cp , B_cp , ALU_cp , red_op_A_cp , red_op_B_cp
    {
        bins or_xor_data_A  = binsof(ALU_cp.Bins_bitwise) && binsof(A_cp.A_data_walkingones) && binsof(B_cp.B_data_0) \
         && binsof(red_op_B_cp.red_op_B_LOW_data) && binsof(red_op_A_cp.red_op_A_HIGH_data) ;
        bins or_xor_data_B  = binsof(ALU_cp.Bins_bitwise) && binsof(B_cp.B_data_walkingones) && binsof(A_cp.A_data_0) \
        && binsof(red_op_A_cp.red_op_A_LOW_data) && binsof(red_op_B_cp.red_op_B_HIGH_data) ;
        option.cross_auto_bin_max = 0 ;
    }
```

```
INVALID_cp4 : cross  ALU_cp , red_op_A_cp , red_op_B_cp
        {
            bins Bins_shift_data  = binsof(ALU_cp.Bins_shift) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
            bins Bins_arith_data  = binsof(ALU_cp.Bins_arith) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
            option.cross_auto_bin_max = 0 ;
        }

endgroup
function new(string name = "ALSU_coverage" , uvm_component parent = null );
    super.new(name ,parent ) ;
    cvr_gp=new();
endfunction

function void build_phase(uvm_phase phase) ;
    super.build_phase(phase);
    cov_export =  new("cov_export" , this) ;
    cov_fifo =  new("cov_fifo" , this) ;

endfunction
```

```
        endfunction

        function void connect_phase(uvm_phase phase) ;
            super.connect_phase(phase);
            cov_export.connect(cov_fifo.analysis_export);
        endfunction

        task run_phase(uvm_phase phase) ;
            super.run_phase(phase);
            forever
            begin
                cov_fifo.get(seq_item_cov) ;
                cvr_gp.sample();
            end
        endtask
    endclass
endpackage
```

16- ALSU sequencer

```
package ALSU_sequencer_pkg ;

    import uvm_pkg::* ;
    import ALSU_seq_item_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_sequencer extends uvm_sequencer #(ALSU_seq_item);
        `uvm_component_utils(ALSU_sequencer)
        function new(string name = "ALSU_sequencer" , uvm_component parent = null );
            super.new(name ,parent ) ;
        endfunction

    endclass
endpackage
```

17- Shift sequencer

```
package shift_reg_sequencer_pkg ;

    import uvm_pkg::* ;
    import shift_reg_seq_item_pkg::* ;
    `include "uvm_macros.svh"


    class shift_reg_sequencer extends uvm_sequencer #(shift_reg_seq_item);
        `uvm_component_utils(shift_reg_sequencer)
        function new(string name = "shift_reg_sequencer" , uvm_component parent = null );
            super.new(name ,parent ) ;
        endfunction

    endclass
endpackage
```

18- Shift monito

```
package shift_reg_monitor_pkg ;
    import uvm_pkg::* ;
    import shift_reg_seq_item_pkg::* ;
    import shared_pkg::* ;
    `include "uvm_macros.svh"
    class shift_reg_monitor extends uvm_monitor ;
        `uvm_component_utils(shift_reg_monitor)
        virtual shift_reg_if shift_vif ;
        shift_reg_seq_item  seq_item ;
        uvm_analysis_port #(shift_reg_seq_item) mon_ap ;
        function new(string name = "shift_reg_monitor" , uvm_component parent = null );
            super.new(name ,parent ) ;
        endfunction
        function void build_phase(uvm_phase phase) ;
            super.build_phase(phase);
           mon_ap = new("mon_ap" , this) ;
        endfunction
        task run_phase(uvm_phase phase) ;
            super.run_phase(phase);
            forever
            begin
                seq_item = shift_reg_seq_item::type_id::create("seq_item");
                #2 ;
                seq_item.serial_in = shift_vif.serial_in ;
                seq_item.direction = direction_e'(shift_vif.direction) ;
                seq_item.mode = mode_e'(shift_vif.mode) ;
                seq_item.datain = shift_vif.datain ;
                //seq_item.dataout = shift_vif.dataout ;
                mon_ap.write(seq_item);
                `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )

            end
           endtask
    endclass
endpackage
```

19- ALSU monitor

```
package ALSU_monitor_pkg ;

   import uvm_pkg::* ;
   import ALSU_seq_item_pkg::* ;
   `include "uvm_macros.svh"

   class ALSU_monitor extends uvm_monitor ;
      `uvm_component_utils(ALSU_monitor)

      virtual ALSU_if ALSU_vif ;
      ALSU_seq_item  seq_item ;

      uvm_analysis_port #(ALSU_seq_item) mon_ap ;

      function new(string name = "ALSU_monitor" , uvm_component parent = null );
         super.new(name ,parent ) ;
      endfunction

      function void build_phase(uvm_phase phase) ;
         super.build_phase(phase);
        mon_ap = new("mon_ap" , this) ;
      endfunction

      task run_phase(uvm_phase phase) ;
         super.run_phase(phase);
         forever
         begin
            seq_item = ALSU_seq_item::type_id::create("seq_item");

            @(negedge ALSU_vif.clk ) ;
            seq_item.serial_in = ALSU_vif.serial_in ;
            seq_item.red_op_A = ALSU_vif.red_op_A ;
            seq_item.red_op_B = ALSU_vif.red_op_B ;
            seq_item.bypass_A = ALSU_vif.bypass_A ;
            seq_item.bypass_B = ALSU_vif.bypass_B ;
            seq_item.direction = ALSU_vif.direction ;
            seq_item.serial_in = ALSU_vif.serial_in ;
            seq_item.opcode = ALSU_vif.opcode ;
            seq_item.A = ALSU_vif.A ;
            seq_item.B = ALSU_vif.B ;
            seq_item.cin = ALSU_vif.cin ;
            mon_ap.write(seq_item);
            `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )

         end
         endtask
   endclass
endpackage
```

20- alsu_driver

```
package ALSU_driver_pkg ;

    import uvm_pkg::* ;
    import ALSU_seq_item_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_driver extends uvm_driver  #(ALSU_seq_item);
        `uvm_component_utils(ALSU_driver)
        virtual ALSU_if alsu_driver_vif ;
      ALSU_seq_item  seq_item ;

        function new(string name = "ALSU_driver" , uvm_component parent = null );
            super.new(name ,parent ) ;
        endfunction

        task run_phase(uvm_phase phase) ;
            super.run_phase(phase);
            alsu_driver_vif.rst = 1 ;
            alsu_driver_vif.red_op_A = 0 ;
            alsu_driver_vif.red_op_B = 0 ;
            alsu_driver_vif.bypass_A = 0 ;
            alsu_driver_vif.bypass_B = 0 ;
            alsu_driver_vif.direction = 0 ;
            alsu_driver_vif.serial_in = 0 ;
            alsu_driver_vif.opcode = 0 ;
            alsu_driver_vif.A = 0 ;
            alsu_driver_vif.B = 0 ;
            alsu_driver_vif.cin = 0 ;
            @(negedge alsu_driver_vif.clk ) ;
            alsu_driver_vif.rst = 0 ;

            forever
            begin
                seq_item = ALSU_seq_item::type_id::create("seq_item");
                seq_item_port.get_next_item(seq_item);
                @(negedge alsu_driver_vif.clk ) ;
                alsu_driver_vif.serial_in = seq_item.serial_in ;
                alsu_driver_vif.red_op_A = seq_item.red_op_A ;
                alsu_driver_vif.red_op_B = seq_item.red_op_B ;
                alsu_driver_vif.bypass_A = seq_item.bypass_A ;
                alsu_driver_vif.bypass_B = seq_item.bypass_B ;
                alsu_driver_vif.direction = seq_item.direction ;
                alsu_driver_vif.serial_in = seq_item.serial_in ;
                alsu_driver_vif.opcode = seq_item.opcode ;
                alsu_driver_vif.A = seq_item.A ;
                alsu_driver_vif.B = seq_item.B ;
                alsu_driver_vif.cin = seq_item.cin ;
                                seq_item_port.item_done();
                `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )
            end
          endtask
    endclass
endpackage
```

21- Shift driver

```
package shift_reg_driver_pkg ;
    import uvm_pkg::* ;
    import shift_reg_seq_item_pkg::* ;
    `include "uvm_macros.svh"
    class shift_reg_driver extends uvm_driver  #(shift_reg_seq_item);
        `uvm_component_utils(shift_reg_driver)
        virtual shift_reg_if shift_vif ;
        shift_reg_seq_item  seq_item ;
        function new(string name = "shift_reg_driver" , uvm_component parent = null );
            super.new(name ,parent ) ;
        endfunction
        task run_phase(uvm_phase phase) ;
            super.run_phase(phase);
            shift_vif.serial_in = 0 ;
            shift_vif.direction = 0 ;
            shift_vif.mode = 0 ;
            shift_vif.datain = 0 ;
            #2 ;
            forever
            begin
                seq_item = shift_reg_seq_item::type_id::create("seq_item");
                seq_item_port.get_next_item(seq_item);
                #2 ;
                shift_vif.serial_in = seq_item.serial_in ;
                shift_vif.direction =seq_item.direction ;
                shift_vif.mode = seq_item.mode ;
                shift_vif.datain = seq_item.datain ;
                //@(negedge shift_vif.clk ) ;
                seq_item_port.item_done();
                `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )

            end
           endtask
    endclass
endpackage
```

22- ALSU seq_item

```
package ALSU_seq_item_pkg ;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"

    class ALSU_seq_item extends uvm_sequence_item;
        `uvm_object_utils(ALSU_seq_item)
        typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
        typedef enum  {Or , Xor , Add , Mult , Shift ,Rotate} opcode_valid_e ;
        localparam MAXPOS = 3  ;
        localparam MAXNEG = -4 ;
        localparam zero = 0 ;
        rand bit  rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
        rand bit [2:0] opcode;
        rand bit signed [2:0] A, B;
        logic  [15:0] leds;
        logic  signed [5:0] out;

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

        constraint input_A {
          if ( (opcode == ADD ) || (opcode == MULT ) )
          {
            A dist { MAXPOS:/25 ,  MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
          }
          else if (((opcode == XOR ) || (opcode == OR )) && (red_op_A == 1) && (red_op_B == 0)  )
          {
            B == 0 ;
            A dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
          }
          else
          {
            A inside { [MAXNEG : MAXPOS] } ;
          }
        }

        constraint input_B {
          if ( (opcode == ADD ) || (opcode == MULT ) )
          {
            B dist { MAXPOS:/25 ,  MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
          }
          else if (((opcode == XOR ) || (opcode == OR )) && (red_op_B == 1) && (red_op_A == 0) )
          {
            A == 0 ;
            B dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
          }
          else
          {
            B inside { [MAXNEG : MAXPOS]  } ;
          }
        }

        constraint input_opcode {opcode dist {[0:3]:/45 , [4:5]:/50 ,[6:7]:/1  } ; }

        constraint input_bypass_A {bypass_A dist {1:=90 , 0:=10  } ; }
        constraint input_bypass_B {bypass_B dist {1:=90 , 0:=10  } ; }

        constraint input_red_op_A {red_op_A dist {1:/10 , 0:/90  } ; }
        constraint input_red_op_B {red_op_B dist {1:/10 , 0:/90  } ; }

        function new(string name = "ALSU_seq_item" );
            super.new(name ) ;
        endfunction

        function string  convert2string();
      return $sformatf ("%s , rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d \
       , bypass_B = %0d , opcode = %0d , A = %0d , B = %0d , leds = %0d  , out = %0d" , super.convert2string , rst , serial_in , \
      direction , cin , red_op_A , red_op_B , bypass_A , bypass_B , opcode  , A , B , leds , out   ) ;
        endfunction

        function string  convert2string_stimulus();
            return $sformatf ("rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d \
             , bypass_B = %0d , opcode = %0d , A = %0d , B = %0d " ,  rst , serial_in , direction , cin , red_op_A  , red_op_B \
             , bypass_A , bypass_B , opcode  , A , B   ) ;
              endfunction

    endclass
endpackage
```

23- Shift seq_item

```
package shift_reg_seq_item_pkg ;
   import uvm_pkg::* ;
   import shared_pkg::* ;
   `include "uvm_macros.svh"

   class shift_reg_seq_item extends uvm_sequence_item;
      `uvm_object_utils(shift_reg_seq_item)


      logic  serial_in;
      logic  direction ;
      logic  mode;
      logic  [5:0] datain ;
      logic  [5:0] dataout;



      function new(string name = "shift_reg_seq_item" );
         super.new(name ) ;
      endfunction

      function string  convert2string();
   return $sformatf ("%s , serial_in = %0d , direction = %0d , mode = %0d , datain = %0d  , dataout = %0d"\
      , super.convert2string , serial_in , direction , mode , datain ,  dataout   ) ;
      endfunction

      function string  convert2string_stimulus();
         return $sformatf (" serial_in = %0d , direction = %0d , mode = %0d , datain = %0d " ,  serial_in , direction , mode , datain   ) ;
            endfunction

   endclass
endpackage
```

24- alsu_config_obj

```
package ALSU_config_pkg ;

   import uvm_pkg::* ;
   `include "uvm_macros.svh"

   class ALSU_config extends uvm_object;
      `uvm_object_utils(ALSU_config)

      virtual ALSU_if alsu_config_vif ;
      function new(string name = "ALSU_config");
         super.new(name) ;
      endfunction
   endclass
endpackage
```

25- Shift config

```
package shift_reg_config_pkg ;

    import uvm_pkg::* ;
    `include "uvm_macros.svh"


    class shift_reg_config extends uvm_object;
        `uvm_object_utils(shift_reg_config)

        virtual shift_reg_if shift_vif ;
        uvm_active_passive_enum  is_active ;
        function new(string name = "shift_reg_config");
            super.new(name) ;
        endfunction
    endclass
endpackage
```

26- Do file

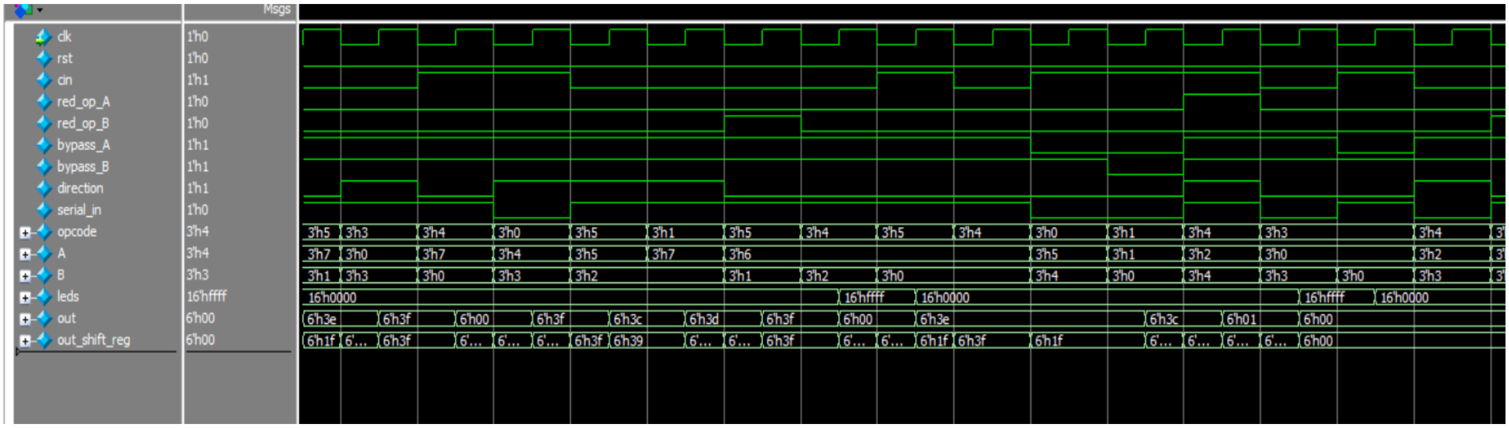
```
vlib work
vlog *v +cover
vsim -voptargs=+acc work.ALSU_shift_top -classdebug -uvmcontrol=all -cover
add wave /ALSU_shift_top/ALSUif/*
coverage save top.ucdb -onexit
run -all

quit -sim
vcover report top.ucdb -all -details  -output coverage.txt
```

27- Transcript

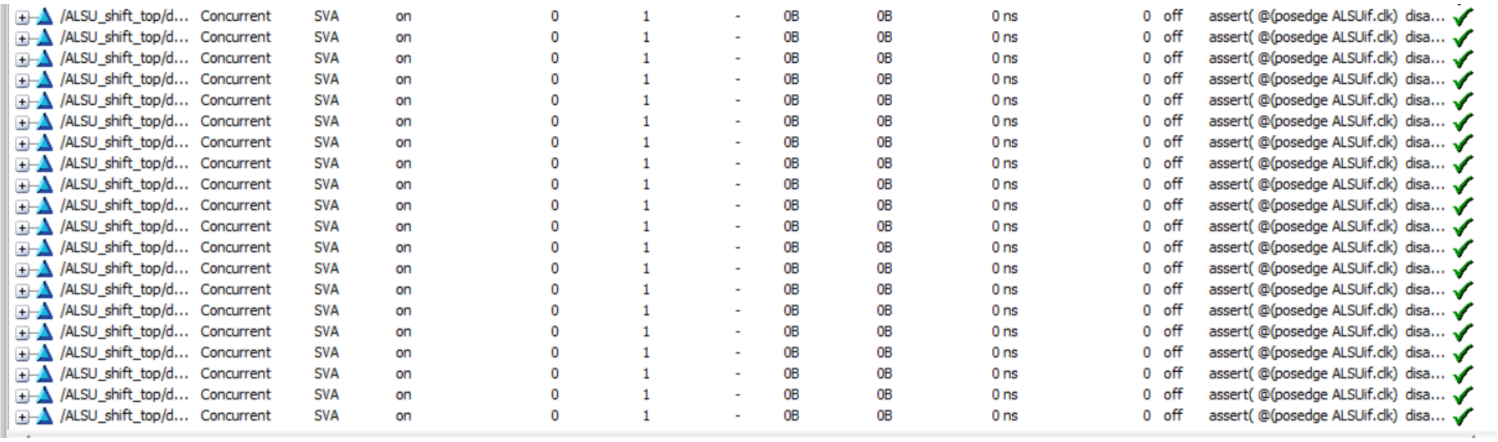
```
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 200004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ALSU_scoreboard.sv(160) @ 200004: uvm_test_top.AL_env.sb [report_phase] total successful 100002
# UVM_INFO ALSU_scoreboard.sv(161) @ 200004: uvm_test_top.AL_env.sb [report_phase] total FAILED 0
# UVM_INFO shift_reg_scoreboard.sv(67) @ 200004: uvm_test_top.shift_env.sb [report_phase] total successful 100002
# UVM_INFO shift_reg_scoreboard.sv(68) @ 200004: uvm_test_top.shift_env.sb [report_phase] total FAILED 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   12
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     4
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_10.4c/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
```

28- Waveform



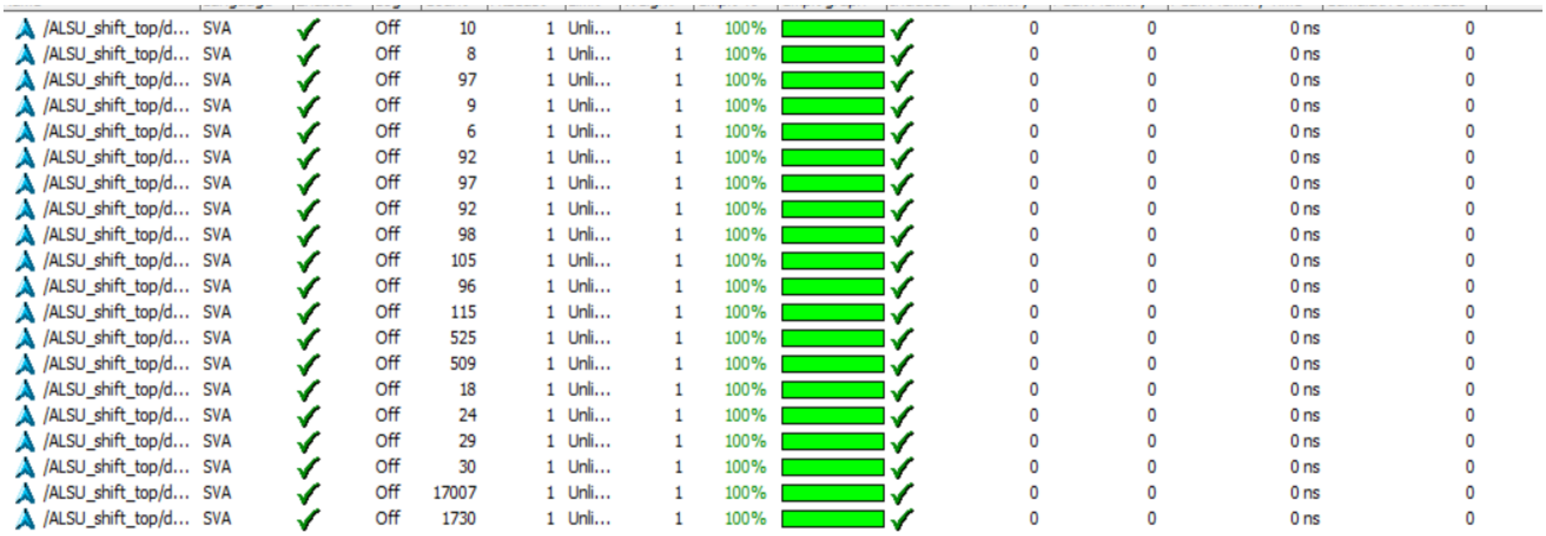
	Msgs
clk	1'h0
rst	1'h0
cin	1'h1
red_op_A	1'h0
red_op_B	1'h0
bypass_A	1'h1
bypass_B	1'h1
direction	1'h1
serial_in	1'h0
opcode	3'h4
A	3'h4
B	3'h3
leds	16'hffff
out	6'h00
out_shift_reg	6'h00

29- Assertion



/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓

30- Assertion coverage



/ALSU_shift_top/d...	SVA	✓	Off	10	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	8	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	97	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	9	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	6	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	92	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	97	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	92	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	98	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	105	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	115	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	525	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	509	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	18	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	24	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	29	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	30	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	17007	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	1730	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0

#part2

31- CODE Design

```
module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input [2:0] opcode;
input signed [2:0] A, B;    // first bug --> we must put it signed
output reg [15:0] leds;
output reg signed [5:0] out;  // second bug --> we must put it signed
reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_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;
      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 (invalid)
        out <= 0;
    else 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 begin
        case (opcode_reg) //  third bug is to used the opcode_reg  not the opcode
          3'h0: begin
            if (red_op_A_reg && red_op_B_reg)
              out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;  // third bug is to replace AND with OR
            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;  // fourth bug is to replace OR with XOR
            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:begin
                if(FULL_ADDER == "ON")  // fifth bug to add Cin operation in case of full adde
                    out <= A_reg + B_reg + cin_reg ;
                else
                    out <= A_reg + B_reg ;
                end
          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
        default : out <= 0 ;
         endcase
      end
    end
  end

  endmodule
```

32- Interface

```
interface ALSU_if (clk);
  input clk ;
  logic  rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
  logic [2:0] opcode;
  logic signed [2:0] A, B;
  logic  [15:0] leds;
  logic  signed [5:0] out;

  modport DUT (input clk , rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in , opcode , A , B , output leds , out ) ;

endinterface
```

33- top module

C: > Users > CS > Downloads > Karem Wasem Diploma > session6_Assignmen > ALSU > ALSU_top.sv

```
import ALSU_test_pkg::*;
import uvm_pkg::* ;
`include "uvm_macros.svh"



module ALSU_top();
bit clk ;

initial
  begin
    clk = 0 ;
    forever
    #1 clk = ~clk ;
  end
  ALSU_if ALSUif (clk) ;
  ALSU dut ( ALSUif.A, ALSUif.B, ALSUif.cin, ALSUif.serial_in, ALSUif.red_op_A, ALSUif.red_op_B, \
  ALSUif.opcode, ALSUif.bypass_A, ALSUif.bypass_B, ALSUif.clk, ALSUif.rst, ALSUif.direction, ALSUif.leds, ALSUif.out) ;
  bind ALSU  SVA  sva(ALSUif.DUT) ;
  initial
    begin
        uvm_config_db#(virtual ALSU_if)::set(null , "uvm_test_top" , "ALSUif" , ALSUif );
        run_test("ALSU_test");
    end


endmodule
```

34- alsu_test

```
package ALSU_test_pkg ;

  import ALSU_env_pkg::*;
  import ALSU_config_pkg::*;
  import ALSU_seq_reset_pkg::*;
  import ALSU_seq_main_pkg::*;
  import ALSU_seq_item_pkg::* ;
  import alsu_seq_item_valid_invalid_pkg::* ;
  import uvm_pkg::* ;
  `include "uvm_macros.svh"


  class ALSU_test extends uvm_test;
      `uvm_component_utils(ALSU_test)
    ALSU_env  env ;
    ALSU_config  alsu_config_obj_test ;
    ALSU_reset_sequence reset_sequence  ;
    ALSU_main_sequence  main_sequence ;

     function new(string name = "ALSU_test" , uvm_component parent = null );
        super.new(name ,parent ) ;
     endfunction

     function void build_phase(uvm_phase phase) ;
      super.build_phase(phase);
      env = ALSU_env::type_id::create("env",this );
      alsu_config_obj_test = ALSU_config::type_id::create("alsu_config_obj_test");
      main_sequence = ALSU_main_sequence::type_id::create("main_sequence" );
      reset_sequence = ALSU_reset_sequence::type_id::create("reset_sequence");
      set_type_override_by_type(ALSU_seq_item::get_type(), alsu_seq_item_valid_invalid::get_type());
      if(!uvm_config_db#(virtual ALSU_if)::get (this , "" , "ALSUif" , alsu_config_obj_test.alsu_config_vif ))
        `uvm_fatal("run_phase" , "test - unable to get the virtual interface ") ;

      uvm_config_db#(ALSU_config)::set (this , "*" , "CFG" , alsu_config_obj_test ) ;

    endfunction

    task run_phase(uvm_phase phase) ;
     super.run_phase(phase);
     phase.raise_objection(this);
     `uvm_info("run_phase","reset assert" ,  UVM_LOW)
     reset_sequence.start(env.agt.sqr);
     `uvm_info("run_phase","reset deassert" ,  UVM_LOW)

     `uvm_info("run_phase","stimulus generation started" ,  UVM_LOW)
     main_sequence.start(env.agt.sqr);
     `uvm_info("run_phase","stimulus generation ended" ,  UVM_LOW)
     phase.drop_objection(this);
    endtask:run_phase

  endclass
endpackage
```

35- alsu_env

```
package ALSU_env_pkg ;

  import ALSU_agent_pkg::*;
   import ALSU_scoreboard_pkg::*;
   import ALSU_coverage_pkg::*;
   import uvm_pkg::* ;
   `include "uvm_macros.svh"


   class ALSU_env extends uvm_env;
      `uvm_component_utils(ALSU_env)

      ALSU_agent  agt ;
      ALSU_scoreboard sb ;
      ALSU_coverage cov ;

      function new(string name = "ALSU_env" , uvm_component parent = null );
         super.new(name ,parent ) ;
      endfunction

      function void build_phase(uvm_phase phase) ;
         super.build_phase(phase);
        agt = ALSU_agent::type_id::create("agt",this );
         sb = ALSU_scoreboard::type_id::create("sb",this );
         cov = ALSU_coverage::type_id::create("cov",this );
      endfunction


      function void connect_phase(uvm_phase phase) ;
         agt.agt_ap.connect(sb.sb_export);
         agt.agt_ap.connect(cov.cov_export);
      endfunction

   endclass
endpackage
```

36- reset sequence

```
package ALSU_seq_reset_pkg ;

    import ALSU_seq_item_pkg::*;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_reset_sequence extends uvm_sequence #(ALSU_seq_item);
        `uvm_object_utils(ALSU_reset_sequence)

        ALSU_seq_item  seq_item ;
        function new(string name = "ALSU_reset_sequence" );
            super.new(name) ;
        endfunction
        task body;
            seq_item = ALSU_seq_item::type_id::create("seq_item");
            start_item(seq_item);
            seq_item.rst = 1 ;
            seq_item.red_op_A = 0 ;
            seq_item.red_op_B = 0 ;
            seq_item.bypass_A = 0 ;
            seq_item.bypass_B = 0 ;
            seq_item.direction = 0 ;
            seq_item.serial_in = 0 ;
            seq_item.opcode = 0 ;
            seq_item.A = 0 ;
            seq_item.B = 0 ;
            seq_item.cin = 0 ;
            finish_item(seq_item);

        endtask
    endclass
endpackage
```

37- Main sequence

```
package ALSU_seq_main_pkg;

    import ALSU_seq_item_pkg::*;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_main_sequence extends uvm_sequence #(ALSU_seq_item);
        `uvm_object_utils(ALSU_main_sequence)

        ALSU_seq_item  seq_item ;
        function new(string name = "ALSU_main_sequence" );
            super.new(name) ;
        endfunction
        task body;
            repeat(100000)
            begin
                seq_item = ALSU_seq_item::type_id::create("seq_item");
                start_item(seq_item);
                assert(seq_item.randomize()) ;
                finish_item(seq_item);
            end

        endtask
    endclass
endpackage
```

38- ALSU agent

```
package ALSU_agent_pkg ;

    import ALSU_driver_pkg::*;
    import ALSU_sequencer_pkg::*;
    import ALSU_monitor_pkg::*;
    import ALSU_config_pkg::*;
    import ALSU_seq_item_pkg::* ;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_agent extends uvm_agent;
        `uvm_component_utils(ALSU_agent)

     ALSU_driver  driver ;
       ALSU_sequencer sqr ;
       ALSU_monitor mon  ;
       ALSU_config  ALSU_cfg ;
       uvm_analysis_port #(ALSU_seq_item) agt_ap ;

       function new(string name = "ALSU_agent" , uvm_component parent = null );
           super.new(name ,parent ) ;
       endfunction
```

```
        function void build_phase(uvm_phase phase) ;
            super.build_phase(phase);
            if(!uvm_config_db#(ALSU_config)::get (this , "" , "CFG" , ALSU_cfg ) )
            `uvm_fatal("build_phase" , "driver - unable to get the virtual interface ") ;

            driver = ALSU_driver::type_id::create("driver",this );
            sqr = ALSU_sequencer::type_id::create("sqr",this );
            mon = ALSU_monitor::type_id::create("mon",this );
            agt_ap = new("agt_ap" , this) ;
        endfunction

        function void connect_phase(uvm_phase phase) ;
          driver.alsu_driver_vif = ALSU_cfg.alsu_config_vif ;
            mon.ALSU_vif = ALSU_cfg.alsu_config_vif ;
            driver.seq_item_port.connect(sqr.seq_item_export) ;
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
  endpackage
```

39- ALSU scoreboard

```
package ALSU_scoreboard_pkg ;
   import ALSU_seq_item_pkg::* ;
   import uvm_pkg::* ;
   `include "uvm_macros.svh"
   class ALSU_scoreboard extends uvm_scoreboard;
      `uvm_component_utils(ALSU_scoreboard)
      uvm_analysis_export #(ALSU_seq_item) sb_export ;
      uvm_tlm_analysis_fifo #(ALSU_seq_item) sb_fifo ;
      ALSU_seq_item seq_item_cov ;
      logic [5:0] dataout_ref ;
      logic [15:0] leds_ref ;

        logic cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
        logic [2:0] opcode_reg;
        logic signed  [2:0]  A_reg, B_reg;
        logic invalid_red_op, invalid_opcode, invalid;
      int error_count = 0 ;
      int correct_count = 0 ;
      function new(string name = "ALSU_scoreboard" , uvm_component parent = null );
         super.new(name ,parent ) ;
      endfunction
      function void build_phase(uvm_phase phase) ;
         super.build_phase(phase);
         sb_export =  new("sb_export" , this) ;
         sb_fifo =  new("sb_fifo" , this) ;
      endfunction

      function void connect_phase(uvm_phase phase) ;
         sb_export.connect(sb_fifo.analysis_export);
      endfunction
```

```
task run_phase(uvm_phase phase) ;
   super.run_phase(phase);
   forever
   begin
      sb_fifo.get(seq_item_cov) ;
      ref_model(seq_item_cov) ;
      if(seq_item_cov.out != dataout_ref && seq_item_cov.leds != leds_ref  )
      begin
         `uvm_error("run_phase" , $sformatf("comparsion failed trasnsaction received by the dut %s shile \
          the reference out %ob" ,seq_item_cov.convert2string , dataout_ref )) ;
         error_count++ ;
      end
      else
      begin
         `uvm_info("run_phase" ,seq_item_cov.convert2string_stimulus() , UVM_HIGH ) ;
         correct_count++ ;
      end
   end
  endtask

   task ref_model(ALSU_seq_item seq_item_cov);
      invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
      invalid_opcode = opcode_reg[1] & opcode_reg[2];
      invalid = invalid_red_op | invalid_opcode;
         fork
            begin
               if(seq_item_cov.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 <= seq_item_cov.cin;
                  red_op_B_reg <= seq_item_cov.red_op_B;
                  red_op_A_reg <= seq_item_cov.red_op_A;
                  bypass_B_reg <= seq_item_cov.bypass_B;
                  bypass_A_reg <= seq_item_cov.bypass_A;
                  direction_reg <= seq_item_cov.direction;
                  serial_in_reg <= seq_item_cov.serial_in;
                  opcode_reg <= seq_item_cov.opcode;
                  A_reg <= seq_item_cov.A;
                  B_reg <= seq_item_cov.B;
               end
            end

```

```
begin
    if(seq_item_cov.rst) begin
        dataout_ref <= 0;
    end
    else begin
    if (invalid)
    dataout_ref <= 0;
    else if (bypass_A_reg && bypass_B_reg)
    dataout_ref <=  A_reg;
    else if (bypass_A_reg)
    dataout_ref <= A_reg;
    else if (bypass_B_reg)
    dataout_ref <= B_reg;
    else begin
        case (opcode_reg)
            3'h0: begin
            if (red_op_A_reg && red_op_B_reg)
            dataout_ref <=|A_reg;
            else if (red_op_A_reg)
            dataout_ref <= |A_reg;
            else if (red_op_B_reg)
            dataout_ref <= |B_reg;
            else
            dataout_ref <= A_reg | B_reg;
            end
            3'h1: begin
            if (red_op_A_reg && red_op_B_reg)
            dataout_ref <=  ^A_reg;
            else if (red_op_A_reg)
            dataout_ref <= ^A_reg;
            else if (red_op_B_reg)
            dataout_ref <= ^B_reg;
            else
            dataout_ref <= A_reg ^ B_reg;
            end
            3'h2:begin
                dataout_ref <= A_reg + B_reg + cin_reg ;
                end
            3'h3: dataout_ref <= A_reg * B_reg;
            3'h4: begin
            if (direction_reg)
            dataout_ref <= {dataout_ref[4:0], serial_in_reg};
            else
            dataout_ref <= {serial_in_reg, dataout_ref[5:1]};
            end
            3'h5: begin
            if (direction_reg)
            dataout_ref <= {dataout_ref[4:0], dataout_ref[5]};
            else
            dataout_ref <= {dataout_ref[0], dataout_ref[5:1]};
            end
        default : dataout_ref <= 0 ;
        endcase
    end

    end
end
```

```
                join
                if(seq_item_cov.rst) begin
                    leds_ref <= 0;
                end else begin
                    if (invalid)
                    leds_ref <= ~leds_ref;
                    else
                    leds_ref <= 0;
                end
            endtask

            function void report_phase(uvm_phase phase) ;
             super.report_phase(phase);
             `uvm_info("report_phase" ,$sformatf("total successful %0d  " ,correct_count ) , UVM_MEDIUM ) ;
             `uvm_info("report_phase" ,$sformatf("total FAILED %0d  " ,error_count ) , UVM_MEDIUM ) ;
           endfunction
        endclass
    endpackage
```

10-ALSU coverage

```
package ALSU_coverage_pkg ;

    import ALSU_seq_item_pkg::* ;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_coverage extends uvm_component;
        `uvm_component_utils(ALSU_coverage)

        uvm_analysis_export #(ALSU_seq_item) cov_export ;
        uvm_tlm_analysis_fifo #(ALSU_seq_item) cov_fifo ;
        ALSU_seq_item seq_item_cov ;
        typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
        typedef enum  {Or , Xor , Add , Mult , Shift ,Rotate} opcode_valid_e ;
        localparam MAXPOS = 3  ;
        localparam MAXNEG = -4 ;
        localparam zero = 0 ;
        covergroup cvr_gp  ;
          A_cp : coverpoint seq_item_cov.A {
              bins A_data_0 = {0} ;
              bins A_data_max = {MAXPOS} ;
              bins A_data_min = {MAXNEG} ;
              bins A_data_default = default ;
              bins A_data_walkingones[] = {1, 2, -4} iff (seq_item_cov.red_op_A && !seq_item_cov.red_op_B);
          }

          B_cp : coverpoint seq_item_cov.B {
              bins B_data_0 = {0} ;
              bins B_data_max = {MAXPOS} ;
              bins B_data_min = {MAXNEG} ;
              bins B_data_default = default ;
              bins B_data_walkingones[] = {1, 2, -4} iff (!seq_item_cov.red_op_A && seq_item_cov.red_op_B);
          }
```

```
ALU_cp : coverpoint seq_item_cov.opcode {
    bins Bins_shift[] = {SHIFT , ROTATE} ;
    bins Bins_arith[] = {ADD , MULT} ;
    bins Bins_bitwise[] = {OR , XOR} ;
    illegal_bins Bins_invalid = {INVALID_6 , INVALID_7} ;
}

cin_cp : coverpoint seq_item_cov.cin {
    bins cin_data = {0 , 1} ;
}

direction_cp : coverpoint seq_item_cov.direction {
    bins direction_data = {0 , 1} ;
}

serial_in_cp : coverpoint seq_item_cov.serial_in {
    bins serial_in_data = {0 , 1} ;
}

red_op_A_cp : coverpoint seq_item_cov.red_op_A {
    bins red_op_A_LOW_data = {0} ;
    bins red_op_A_HIGH_data = {1} ;
}
red_op_B_cp : coverpoint seq_item_cov.red_op_B {
    bins red_op_B_LOW_data = {0} ;
    bins red_op_B_HIGH_data = {1} ;
}
```

```
add_mult_cp1 : cross  A_cp , B_cp , ALU_cp
    {

        bins zero_A_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_0) && binsof(B_cp.B_data_0) ;
        bins max_pos_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_max) && binsof(B_cp.B_data_max) ;
        option.cross_auto_bin_max = 0 ;

    }


opcode_cp2 : cross  cin_cp , direction_cp , serial_in_cp , ALU_cp ,red_op_A_cp , red_op_B_cp
    {

        bins cin_add = binsof(cin_cp.cin_data) && binsof(ALU_cp.Bins_arith) intersect {ADD} ;
        bins serialin_shift = binsof(serial_in_cp.serial_in_data) && binsof(ALU_cp) intersect {SHIFT} ;
        bins direction_shift_rota = binsof(direction_cp) && binsof(ALU_cp.Bins_shift) ;
        option.cross_auto_bin_max = 0 ;

    }


or_xor_cp3 : cross  A_cp , B_cp , ALU_cp , red_op_A_cp , red_op_B_cp
    {
        bins or_xor_data_A  = binsof(ALU_cp.Bins_bitwise) && binsof(A_cp.A_data_walkingones) && binsof(B_cp.B_data_0) \
         && binsof(red_op_B_cp.red_op_B_LOW_data) && binsof(red_op_A_cp.red_op_A_HIGH_data) ;
        bins or_xor_data_B  = binsof(ALU_cp.Bins_bitwise) && binsof(B_cp.B_data_walkingones) && binsof(A_cp.A_data_0) \
        && binsof(red_op_A_cp.red_op_A_LOW_data) && binsof(red_op_B_cp.red_op_B_HIGH_data) ;
        option.cross_auto_bin_max = 0 ;
    }
```

```
INVALID_cp4 : cross  ALU_cp , red_op_A_cp , red_op_B_cp
        {
            bins Bins_shift_data  = binsof(ALU_cp.Bins_shift) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
            bins Bins_arith_data  = binsof(ALU_cp.Bins_arith) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
            option.cross_auto_bin_max = 0 ;
        }

    endgroup
    function new(string name = "ALSU_coverage" , uvm_component parent = null );
        super.new(name ,parent ) ;
        cvr_gp=new();
    endfunction

    function void build_phase(uvm_phase phase) ;
        super.build_phase(phase);
        cov_export =  new("cov_export" , this) ;
        cov_fifo =  new("cov_fifo" , this) ;

    endfunction
```

```
        endfunction

        function void connect_phase(uvm_phase phase) ;
            super.connect_phase(phase);
            cov_export.connect(cov_fifo.analysis_export);
        endfunction

        task run_phase(uvm_phase phase) ;
            super.run_phase(phase);
            forever
            begin
                cov_fifo.get(seq_item_cov) ;
                cvr_gp.sample();
            end
        endtask
    endclass
endpackage
```

11- ALSU sequencer

```
package ALSU_sequencer_pkg ;

    import uvm_pkg::* ;
    import ALSU_seq_item_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_sequencer extends uvm_sequencer #(ALSU_seq_item);
        `uvm_component_utils(ALSU_sequencer)
        function new(string name = "ALSU_sequencer" , uvm_component parent = null );
            super.new(name ,parent ) ;
        endfunction

    endclass
endpackage
```

12-ALSU monitor

```
package ALSU_monitor_pkg ;

   import uvm_pkg::* ;
   import ALSU_seq_item_pkg::* ;
   `include "uvm_macros.svh"

   class ALSU_monitor extends uvm_monitor ;
      `uvm_component_utils(ALSU_monitor)

      virtual ALSU_if ALSU_vif ;
      ALSU_seq_item  seq_item ;

      uvm_analysis_port #(ALSU_seq_item) mon_ap ;

      function new(string name = "ALSU_monitor" , uvm_component parent = null );
         super.new(name ,parent ) ;
      endfunction

      function void build_phase(uvm_phase phase) ;
         super.build_phase(phase);
        mon_ap = new("mon_ap" , this) ;
      endfunction

      task run_phase(uvm_phase phase) ;
         super.run_phase(phase);
         forever
         begin
            seq_item = ALSU_seq_item::type_id::create("seq_item");

            @(negedge ALSU_vif.clk ) ;
            seq_item.serial_in = ALSU_vif.serial_in ;
            seq_item.red_op_A = ALSU_vif.red_op_A ;
            seq_item.red_op_B = ALSU_vif.red_op_B ;
            seq_item.bypass_A = ALSU_vif.bypass_A ;
            seq_item.bypass_B = ALSU_vif.bypass_B ;
            seq_item.direction = ALSU_vif.direction ;
            seq_item.serial_in = ALSU_vif.serial_in ;
            seq_item.opcode = ALSU_vif.opcode ;
            seq_item.A = ALSU_vif.A ;
            seq_item.B = ALSU_vif.B ;
            seq_item.cin = ALSU_vif.cin ;
            mon_ap.write(seq_item);
            `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )

         end
         endtask
   endclass
endpackage
```

13-alsu_driver

```
package ALSU_driver_pkg ;

    import uvm_pkg::* ;
    import ALSU_seq_item_pkg::* ;
    `include "uvm_macros.svh"


    class ALSU_driver extends uvm_driver  #(ALSU_seq_item);
        `uvm_component_utils(ALSU_driver)
        virtual ALSU_if alsu_driver_vif ;
      ALSU_seq_item  seq_item ;

        function new(string name = "ALSU_driver" , uvm_component parent = null );
            super.new(name ,parent ) ;
        endfunction

        task run_phase(uvm_phase phase) ;
            super.run_phase(phase);
            alsu_driver_vif.rst = 1 ;
            alsu_driver_vif.red_op_A = 0 ;
            alsu_driver_vif.red_op_B = 0 ;
            alsu_driver_vif.bypass_A = 0 ;
            alsu_driver_vif.bypass_B = 0 ;
            alsu_driver_vif.direction = 0 ;
            alsu_driver_vif.serial_in = 0 ;
            alsu_driver_vif.opcode = 0 ;
            alsu_driver_vif.A = 0 ;
            alsu_driver_vif.B = 0 ;
            alsu_driver_vif.cin = 0 ;
            @(negedge alsu_driver_vif.clk ) ;
            alsu_driver_vif.rst = 0 ;

            forever
            begin
                seq_item = ALSU_seq_item::type_id::create("seq_item");
                seq_item_port.get_next_item(seq_item);
                @(negedge alsu_driver_vif.clk ) ;
                alsu_driver_vif.serial_in = seq_item.serial_in ;
                alsu_driver_vif.red_op_A = seq_item.red_op_A ;
                alsu_driver_vif.red_op_B = seq_item.red_op_B ;
                alsu_driver_vif.bypass_A = seq_item.bypass_A ;
                alsu_driver_vif.bypass_B = seq_item.bypass_B ;
                alsu_driver_vif.direction = seq_item.direction ;
                alsu_driver_vif.serial_in = seq_item.serial_in ;
                alsu_driver_vif.opcode = seq_item.opcode ;
                alsu_driver_vif.A = seq_item.A ;
                alsu_driver_vif.B = seq_item.B ;
                alsu_driver_vif.cin = seq_item.cin ;
                                seq_item_port.item_done();
                `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )
            end
          endtask
    endclass
endpackage
```

14-ALSU seq_item

```
package ALSU_seq_item_pkg ;
    import uvm_pkg::* ;
    `include "uvm_macros.svh"

    class ALSU_seq_item extends uvm_sequence_item;
        `uvm_object_utils(ALSU_seq_item)
        typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
        typedef enum  {Or , Xor , Add , Mult , Shift ,Rotate} opcode_valid_e ;
        localparam MAXPOS = 3  ;
        localparam MAXNEG = -4 ;
        localparam zero = 0 ;
        rand bit  rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
        rand bit [2:0] opcode;
        rand bit signed [2:0] A, B;
        logic  [15:0] leds;
        logic  signed [5:0] out;

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

        constraint input_A {
          if ( (opcode == ADD ) || (opcode == MULT ) )
          {
            A dist { MAXPOS:/25 ,  MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
          }
          else if (((opcode == XOR ) || (opcode == OR )) && (red_op_A == 1) && (red_op_B == 0)  )
          {
            B == 0 ;
            A dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
          }
          else
          {
            A inside { [MAXNEG : MAXPOS] } ;
          }
         }

        constraint input_B {
         if ( (opcode == ADD ) || (opcode == MULT ) )
         {
           B dist { MAXPOS:/25 ,  MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
         }
         else if (((opcode == XOR ) || (opcode == OR )) && (red_op_B == 1) && (red_op_A == 0) )
         {
           A == 0 ;
           B dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
         }
         else
         {
           B inside { [MAXNEG : MAXPOS]  } ;
         }
        }

        constraint input_opcode {opcode dist {[0:3]:/50 , [4:5]:/50 } ; }

        constraint input_bypass_A {bypass_A dist {1:=90 , 0:=10  } ; }
        constraint input_bypass_B {bypass_B dist {1:=90 , 0:=10  } ; }

        constraint input_red_op_A {red_op_A dist {1:/10 , 0:/90  } ; }
        constraint input_red_op_B {red_op_B dist {1:/10 , 0:/90  } ; }

      function new(string name = "ALSU_seq_item" );
          super.new(name ) ;
      endfunction

      function string  convert2string();
    return $sformatf ("%s , rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d \
     , bypass_B = %0d , opcode = %0d , A = %0d , B = %0d , leds = %0d  , out = %0d" , super.convert2string , rst , serial_in , \
     direction , cin , red_op_A , red_op_B , bypass_A , bypass_B , opcode  , A , B , leds , out   ) ;
      endfunction

      function string  convert2string_stimulus();
         return $sformatf ("rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d \
          , bypass_B = %0d , opcode = %0d , A = %0d , B = %0d " ,  rst , serial_in , direction , cin , red_op_A  , red_op_B \
          , bypass_A , bypass_B , opcode  , A , B   ) ;
           endfunction

    endclass
endpackage
```

15-ALSU seq_item_valid_invalid

```
package alsu_seq_item_valid_invalid_pkg ;
   import uvm_pkg::* ;
   import ALSU_seq_item_pkg::* ;
   `include "uvm_macros.svh"

   class alsu_seq_item_valid_invalid extends ALSU_seq_item;
      `uvm_object_utils(alsu_seq_item_valid_invalid)

       constraint input_opcode {opcode dist {[0:3]:/40 , [4:5]:/40 ,[6:7]:/20  } ; }

       constraint input_bypass_A {bypass_A dist {1:=90 , 0:=10  } ; }
       constraint input_bypass_B {bypass_B dist {1:=90 , 0:=10  } ; }

       constraint input_red_op_A {red_op_A dist {1:/10 , 0:/90  } ; }
       constraint input_red_op_B {red_op_B dist {1:/10 , 0:/90  } ; }


      function new(string name = "alsu_seq_item_valid_invalid" );
         super.new(name ) ;
      endfunction

      function string  convert2string();
   return $sformatf ("%s , rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d , bypass_B = %0d \
   , opcode = %0d , A = %0d , B = %0d , leds = %0d  , out = %0d" , super.convert2string , rst , serial_in , direction , cin , red_op_A , red_op_B \
   , bypass_A , bypass_B , opcode , A , B , leds , out   ) ;
      endfunction

      function string  convert2string_stimulus();
         return $sformatf ("rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d , bypass_B = %0d \
          , opcode = %0d , A = %0d , B = %0d " ,  rst , serial_in , direction , cin , red_op_A  , red_op_B , bypass_A , bypass_B , opcode  , A , B   ) ;
            endfunction

   endclass
endpackage
```

16-alsu_config_obj

```
package ALSU_config_pkg ;

   import uvm_pkg::* ;
   `include "uvm_macros.svh"

   class ALSU_config extends uvm_object;
      `uvm_object_utils(ALSU_config)

      virtual ALSU_if alsu_config_vif ;
      function new(string name = "ALSU_config");
         super.new(name) ;
      endfunction
   endclass
endpackage
```

17- Do file

C: > Users > CS > Downloads > Karem Wasem Diploma > session6_Assignmen > ALSU > run.do

```
vlib work
vlog *v +cover
vsim -voptargs=+acc work.ALSU_top -classdebug -uvmcontrol=all -cover
add wave /ALSU_top/ALSUif/*
coverage save top.ucdb -onexit
run -all
quit -sim
vcover report top.ucdb -all -details -output coverage.txt
```

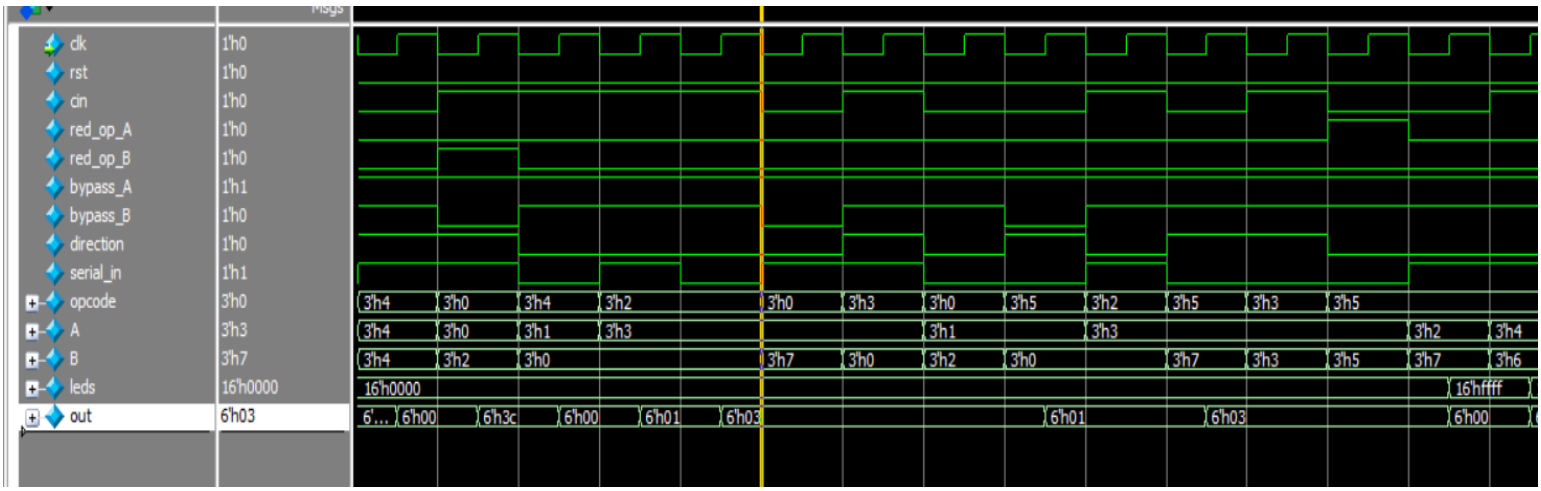
18-Transcript without the override method call

```
# *  To turn off, set 'recording_detail' to off:                          *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ************************************************************************
# UVM_INFO ALSU_test.sv(43) @ 4: uvm_test_top [run_phase] reset deassert
# UVM_INFO ALSU_test.sv(45) @ 4: uvm_test_top [run_phase] stimulus generation started
# UVM_INFO ALSU_test.sv(47) @ 200004: uvm_test_top [run_phase] stimulus generation ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 200004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ALSU_scoreboard.sv(160) @ 200004: uvm_test_top.env.sb [report_phase] total successful 100002
# UVM_INFO ALSU_scoreboard.sv(161) @ 200004: uvm_test_top.env.sb [report_phase] total FAILED 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   10
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_10.4c/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 200004 ns  Iteration: 61  Instance: /ALSU_top
```

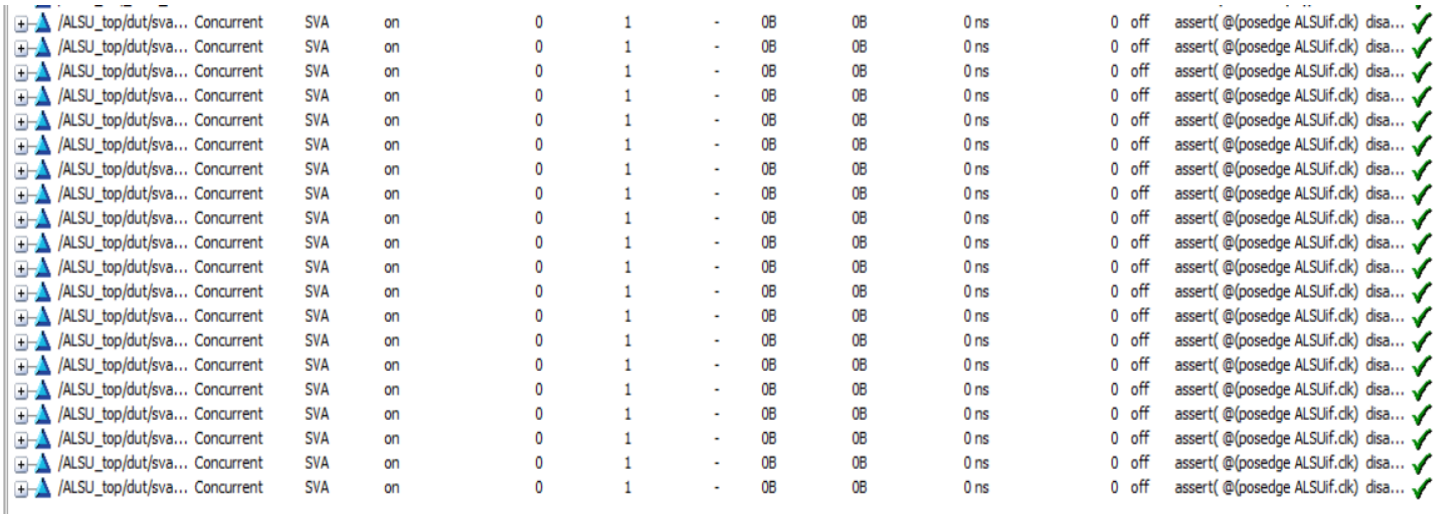
19-Transcript without the override method call

```
# ** Error: (vsim-8565) Illegal state bin was hit at value=6. The bin counter for the illegal bin '\/ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19899.
#    Time: 199936 ns  Iteration: 2  Region: /uvm_pkg::uvm_task_phase::execute
# ** Error: (vsim-8565) Illegal state bin was hit at value=6. The bin counter for the illegal bin '\/ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19900.
#    Time: 199938 ns  Iteration: 2  Region: /uvm_pkg::uvm_task_phase::execute
# ** Error: (vsim-8565) Illegal state bin was hit at value=7. The bin counter for the illegal bin '\/ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19901.
#    Time: 199970 ns  Iteration: 2  Region: /uvm_pkg::uvm_task_phase::execute
# ** Error: (vsim-8565) Illegal state bin was hit at value=6. The bin counter for the illegal bin '\/ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19902.
#    Time: 199986 ns  Iteration: 2  Region: /uvm_pkg::uvm_task_phase::execute
# ** Error: (vsim-8565) Illegal state bin was hit at value=6. The bin counter for the illegal bin '\/ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19903.
#    Time: 200004 ns  Iteration: 2  Region: /uvm_pkg::uvm_task_phase::execute
# UVM_INFO ALSU_test.sv(47) @ 200004: uvm_test_top [run_phase] stimulus generation ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 200004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ALSU_scoreboard.sv(160) @ 200004: uvm_test_top.env.sb [report_phase] total successful 100002
# UVM_INFO ALSU_scoreboard.sv(161) @ 200004: uvm_test_top.env.sb [report_phase] total FAILED 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   10
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]     4
```

20- Waveform



21-Assertion



22-Assertion coverage

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/ALSU_top/dut/sva...	SVA	✓	Off	15	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	14	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	74	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	8	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	15	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	96	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	89	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	105	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	114	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	94	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	89	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	111	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	512	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	512	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	18	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	14	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	25	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	20	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	17079	1	Unli...	1	100%		✓	0	0	0 ns	0
/ALSU_top/dut/sva...	SVA	✓	Off	1627	1	Unli...	1	100%		✓	0	0	0 ns	0