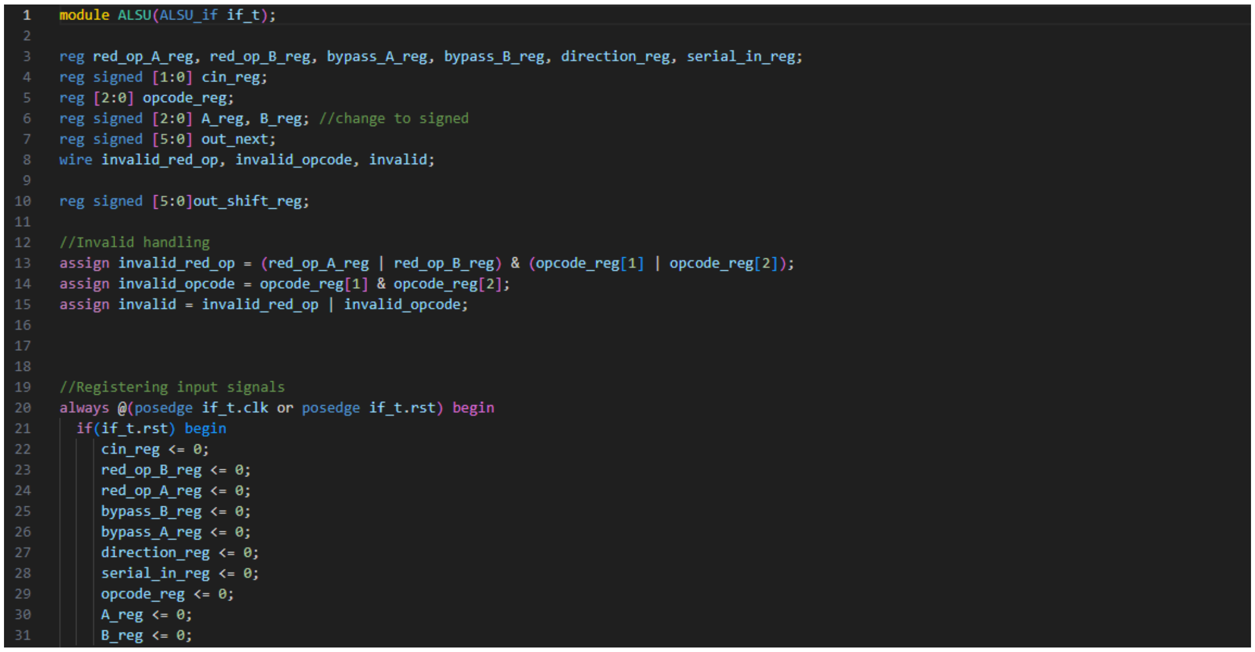
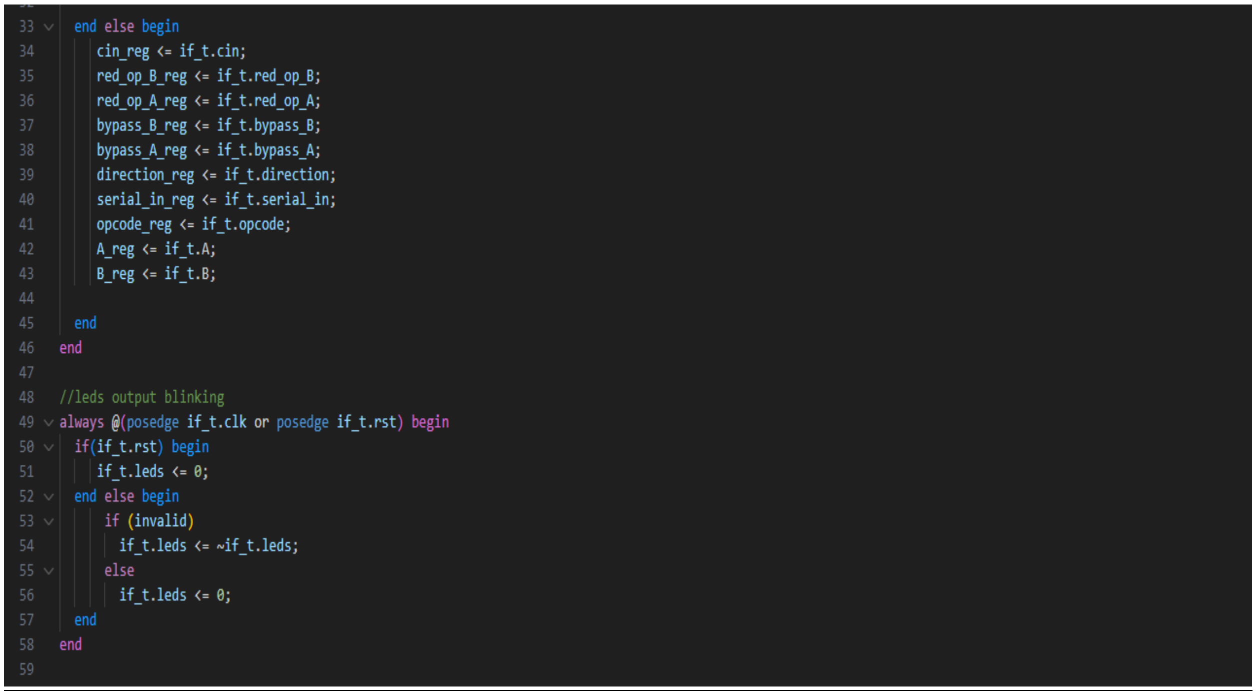
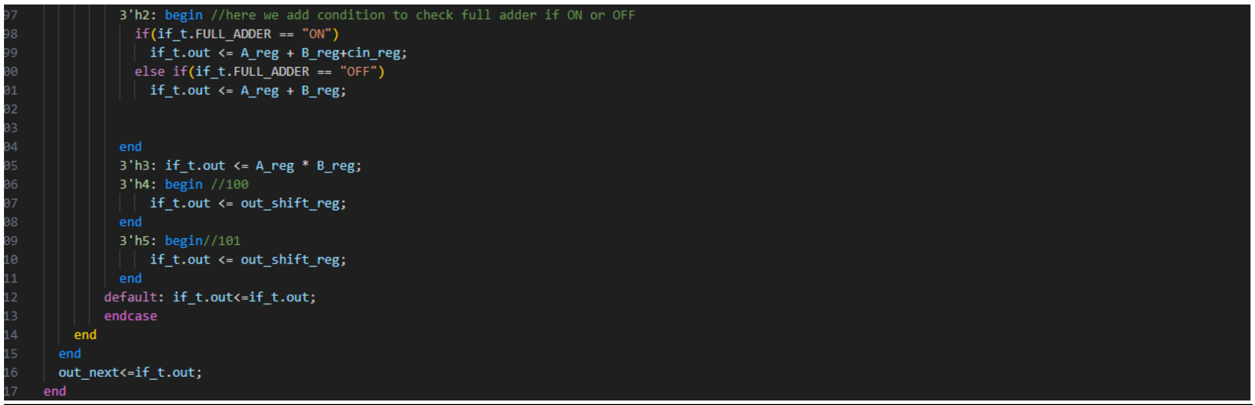
ALSU_SHIFT_reg



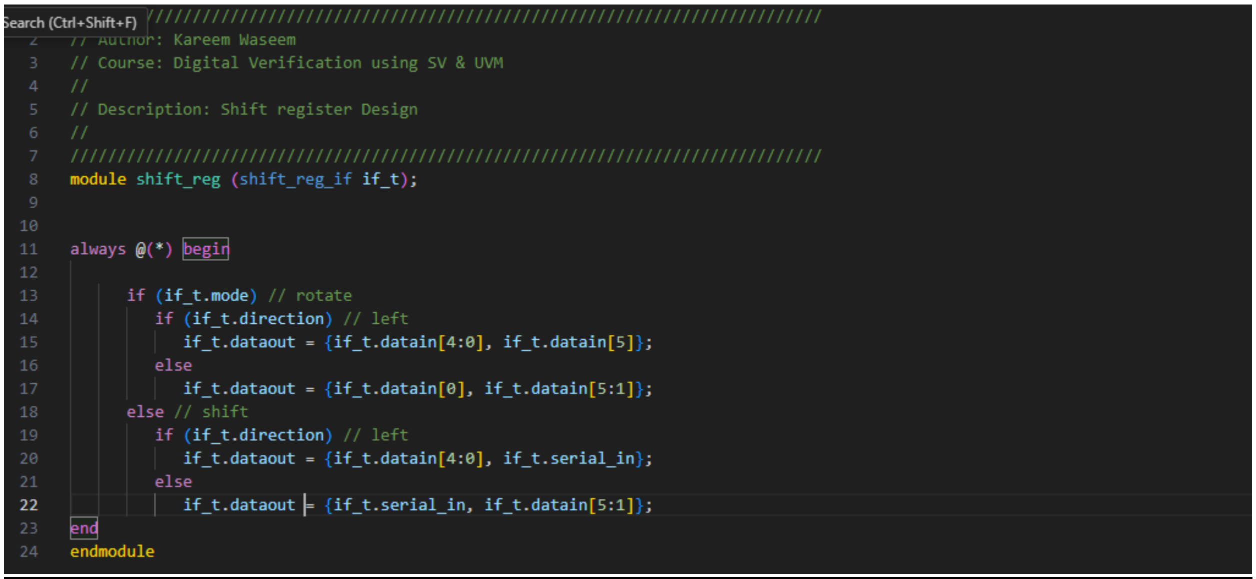
```verilog
module ALSU(ALSU_if if_t);

reg red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg signed [1:0] cin_reg;
reg [2:0] opcode_reg;
reg signed [2:0] A_reg, B_reg; //change to signed
reg signed [5:0] out_next;
wire invalid_red_op, invalid_opcode, invalid;

reg signed [5:0]out_shift_reg;

//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 if_t.clk or posedge if_t.rst) begin
  if(if_t.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;
```



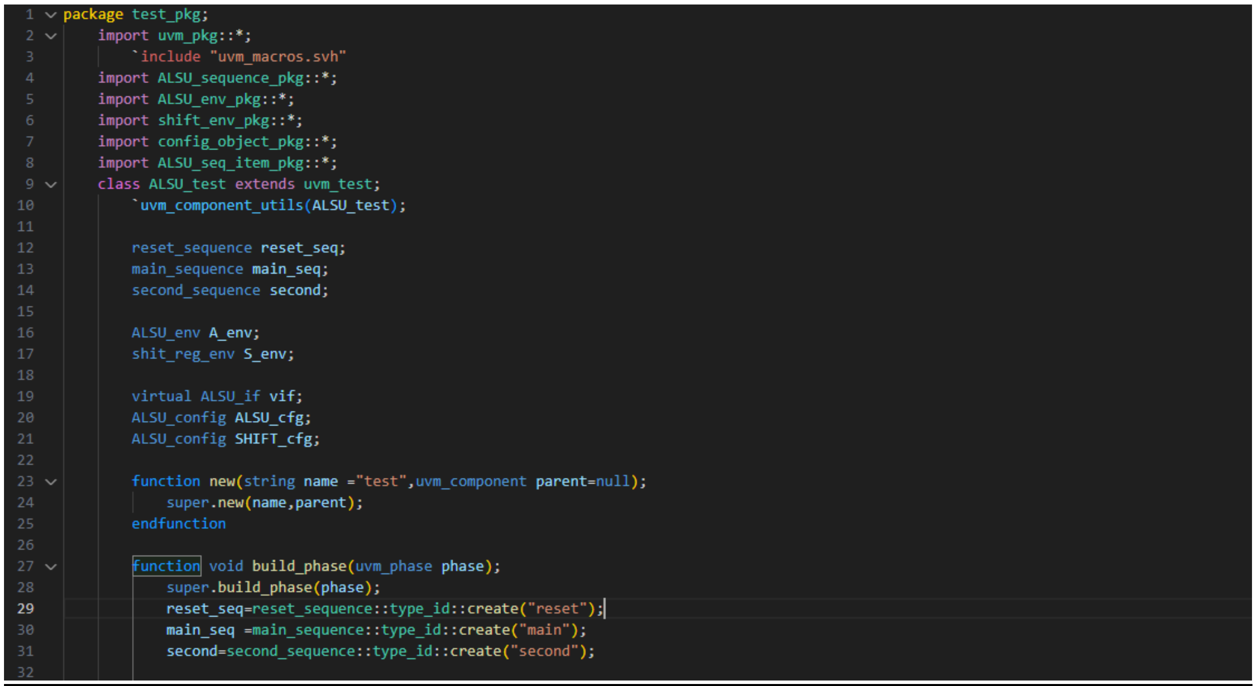
```verilog
  end else begin
     cin_reg <= if_t.cin;
     red_op_B_reg <= if_t.red_op_B;
     red_op_A_reg <= if_t.red_op_A;
     bypass_B_reg <= if_t.bypass_B;
     bypass_A_reg <= if_t.bypass_A;
     direction_reg <= if_t.direction;
     serial_in_reg <= if_t.serial_in;
     opcode_reg <= if_t.opcode;
     A_reg <= if_t.A;
     B_reg <= if_t.B;

  end
end

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

```



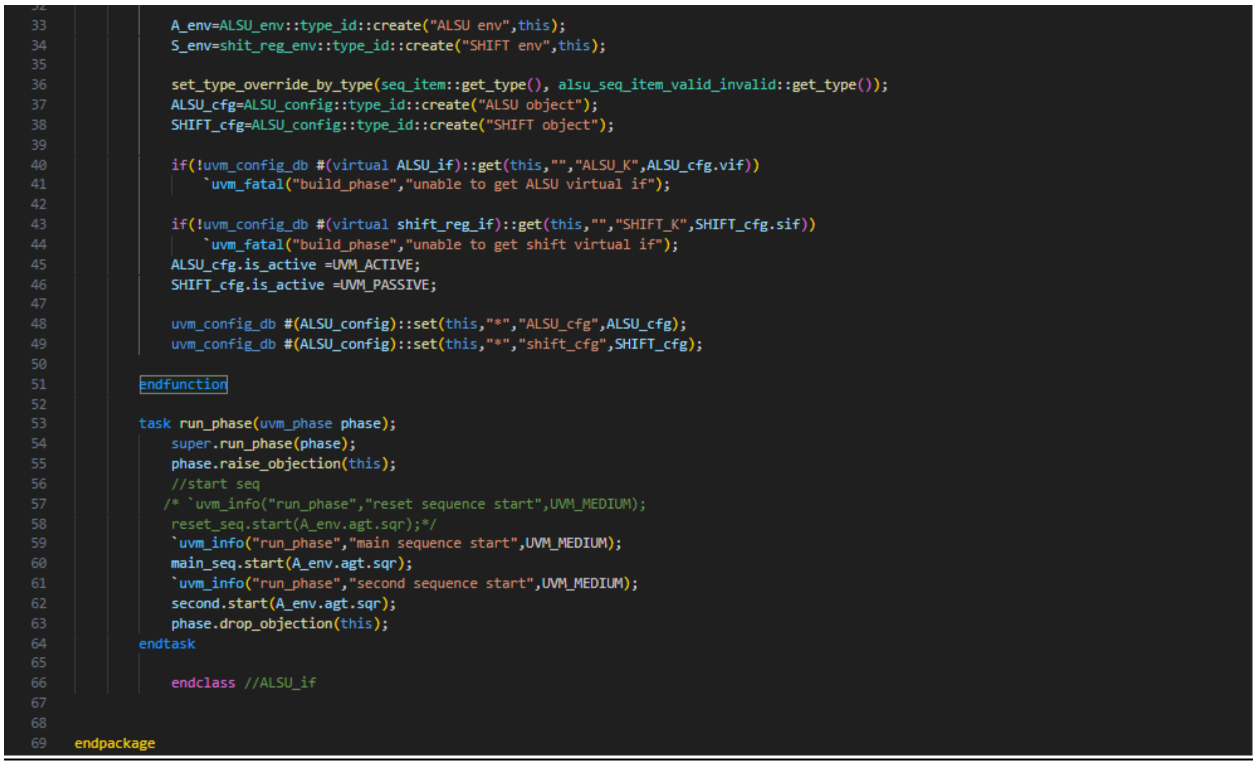
```
                3'h2: begin //here we add condition to check full adder if ON or OFF
                  if(if_t.FULL_ADDER == "ON")
                    if_t.out <= A_reg + B_reg+cin_reg;
                  else if(if_t.FULL_ADDER == "OFF")
                    if_t.out <= A_reg + B_reg;


                end
                3'h3: if_t.out <= A_reg * B_reg;
                3'h4: begin //100
                    if_t.out <= out_shift_reg;
                end
                3'h5: begin//101
                    if_t.out <= out_shift_reg;
                end
              default: if_t.out<=if_t.out;
              endcase
          end
        end
        out_next<=if_t.out;
      end
```



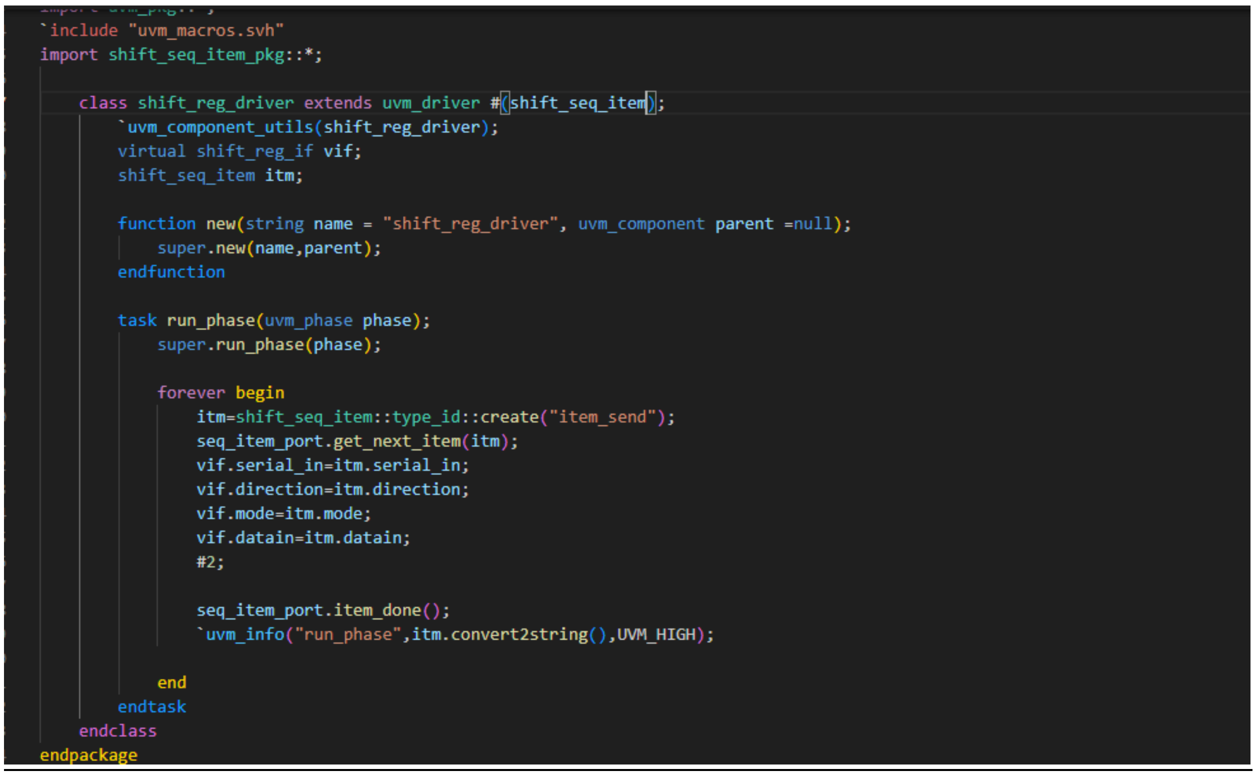
```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: Shift register Design
//
////////////////////////////////////////////////////////////////////////////////
module shift_reg (shift_reg_if if_t);


always @(*) begin

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



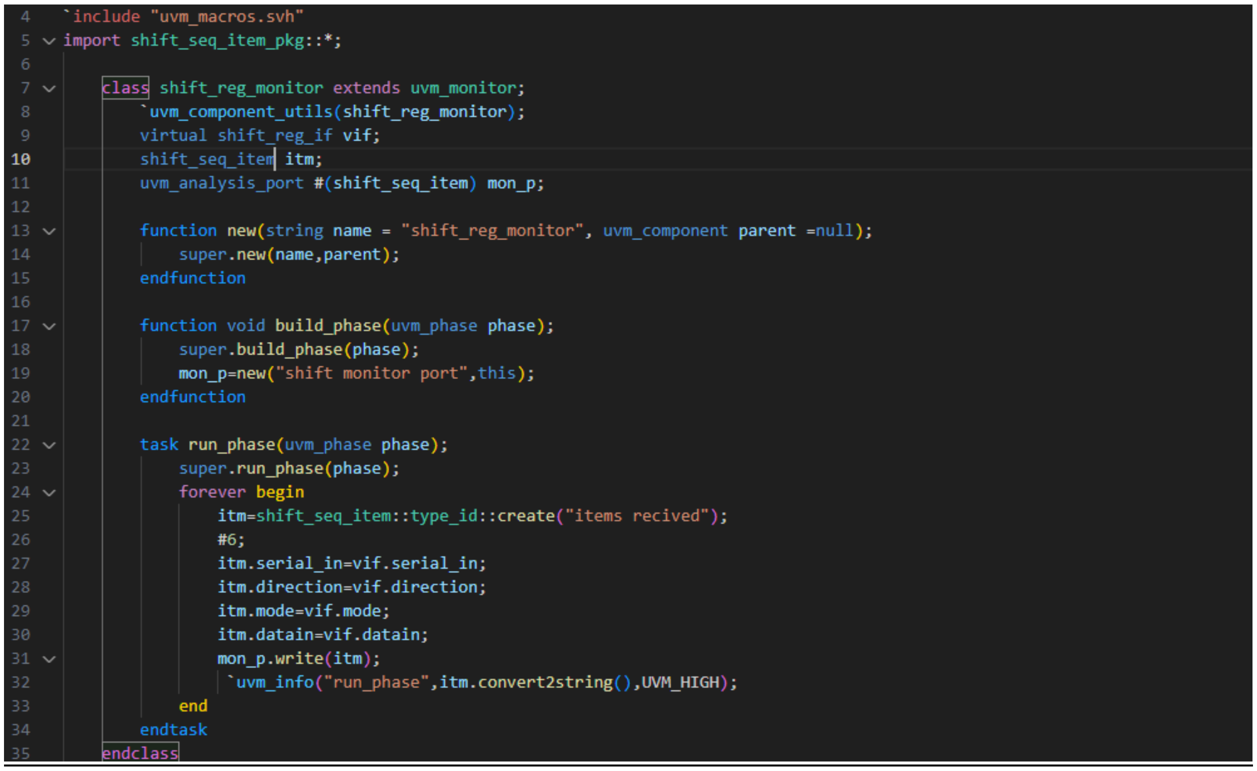
```
package test_pkg;
    import uvm_pkg::*;
        `include "uvm_macros.svh"
    import ALSU_sequence_pkg::*;
    import ALSU_env_pkg::*;
    import shift_env_pkg::*;
    import config_object_pkg::*;
    import ALSU_seq_item_pkg::*;
    class ALSU_test extends uvm_test;
        `uvm_component_utils(ALSU_test);

        reset_sequence reset_seq;
        main_sequence main_seq;
        second_sequence second;

        ALSU_env A_env;
        shit_reg_env S_env;

        virtual ALSU_if vif;
        ALSU_config ALSU_cfg;
        ALSU_config SHIFT_cfg;

        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);
            reset_seq=reset_sequence::type_id::create("reset");
            main_seq =main_sequence::type_id::create("main");
            second=second_sequence::type_id::create("second");

```



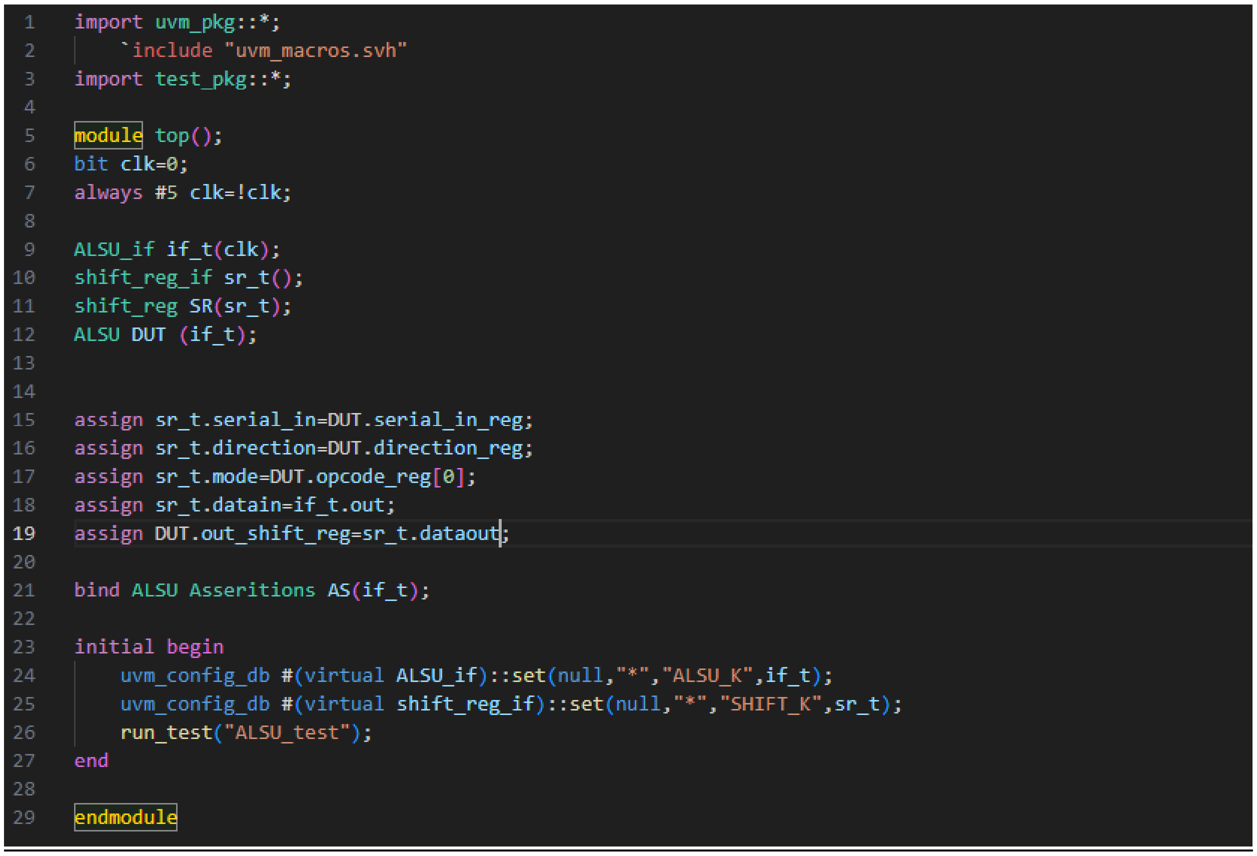
```
            A_env=ALSU_env::type_id::create("ALSU env",this);
            S_env=shit_reg_env::type_id::create("SHIFT env",this);

            set_type_override_by_type(seq_item::get_type(), alsu_seq_item_valid_invalid::get_type());
            ALSU_cfg=ALSU_config::type_id::create("ALSU object");
            SHIFT_cfg=ALSU_config::type_id::create("SHIFT object");

            if(!uvm_config_db #(virtual ALSU_if)::get(this,"","ALSU_K",ALSU_cfg.vif))
                `uvm_fatal("build_phase","unable to get ALSU virtual if");

            if(!uvm_config_db #(virtual shift_reg_if)::get(this,"","SHIFT_K",SHIFT_cfg.sif))
                `uvm_fatal("build_phase","unable to get shift virtual if");
            ALSU_cfg.is_active =UVM_ACTIVE;
            SHIFT_cfg.is_active =UVM_PASSIVE;

            uvm_config_db #(ALSU_config)::set(this,"*","ALSU_cfg",ALSU_cfg);
            uvm_config_db #(ALSU_config)::set(this,"*","shift_cfg",SHIFT_cfg);

        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            //start seq
           /* `uvm_info("run_phase","reset sequence start",UVM_MEDIUM);
            reset_seq.start(A_env.agt.sqr);*/
            `uvm_info("run_phase","main sequence start",UVM_MEDIUM);
            main_seq.start(A_env.agt.sqr);
            `uvm_info("run_phase","second sequence start",UVM_MEDIUM);
            second.start(A_env.agt.sqr);
            phase.drop_objection(this);
        endtask

            endclass //ALSU_if


endpackage
```



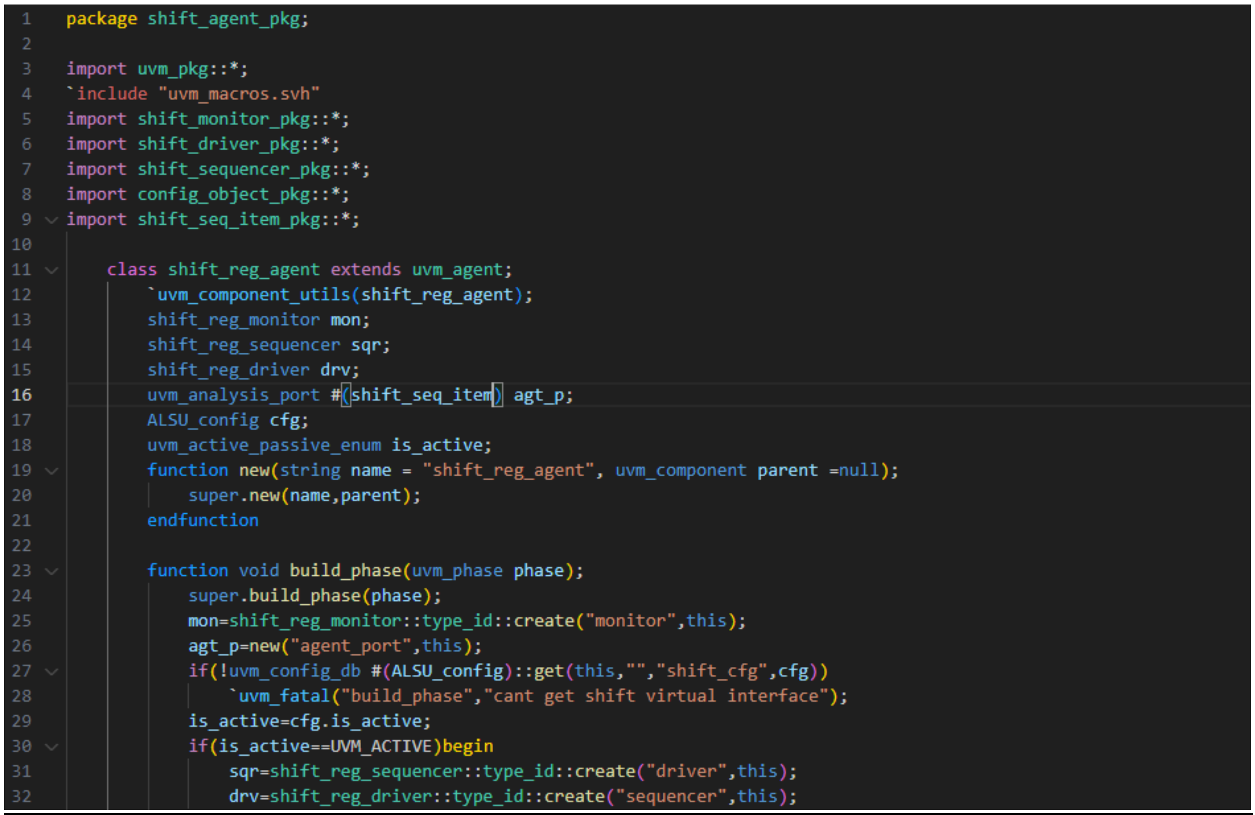
```
`include "uvm_macros.svh"
import shift_seq_item_pkg::*;

    class shift_reg_driver extends uvm_driver #(shift_seq_item);
        `uvm_component_utils(shift_reg_driver);
        virtual shift_reg_if vif;
        shift_seq_item itm;

        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);

            forever begin
                itm=shift_seq_item::type_id::create("item_send");
                seq_item_port.get_next_item(itm);
                vif.serial_in=itm.serial_in;
                vif.direction=itm.direction;
                vif.mode=itm.mode;
                vif.datain=itm.datain;
                #2;

                seq_item_port.item_done();
                `uvm_info("run_phase",itm.convert2string(),UVM_HIGH);

            end
        endtask
    endclass
endpackage
```



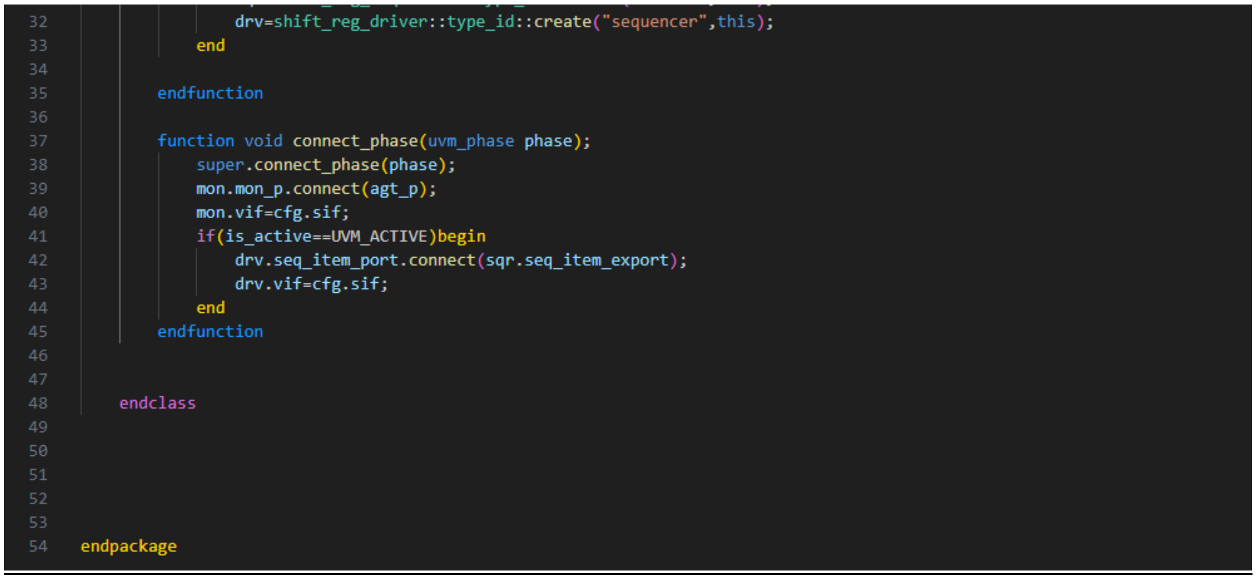
```
`include "uvm_macros.svh"
import shift_seq_item_pkg::*;

    class shift_reg_monitor extends uvm_monitor;
        `uvm_component_utils(shift_reg_monitor);
        virtual shift_reg_if vif;
        shift_seq_item itm;
        uvm_analysis_port #(shift_seq_item) mon_p;

        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_p=new("shift monitor port",this);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                itm=shift_seq_item::type_id::create("items recived");
                #6;
                itm.serial_in=vif.serial_in;
                itm.direction=vif.direction;
                itm.mode=vif.mode;
                itm.datain=vif.datain;
                mon_p.write(itm);
                 `uvm_info("run_phase",itm.convert2string(),UVM_HIGH);
            end
        endtask
    endclass
```



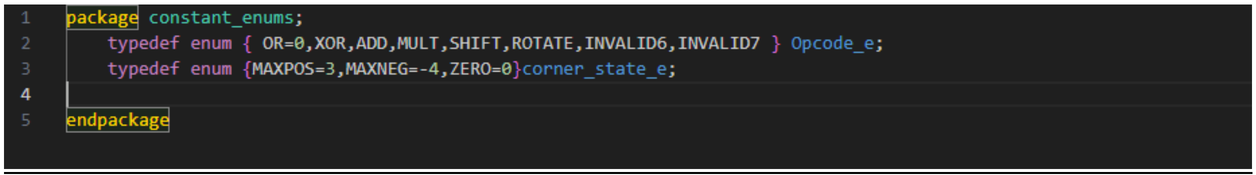
```
import uvm_pkg::*;
    `include "uvm_macros.svh"
import test_pkg::*;

module top();
bit clk=0;
always #5 clk=!clk;

ALSU_if if_t(clk);
shift_reg_if sr_t();
shift_reg SR(sr_t);
ALSU DUT (if_t);


assign sr_t.serial_in=DUT.serial_in_reg;
assign sr_t.direction=DUT.direction_reg;
assign sr_t.mode=DUT.opcode_reg[0];
assign sr_t.datain=if_t.out;
assign DUT.out_shift_reg=sr_t.dataout;

bind ALSU Asseritions AS(if_t);

initial begin
    uvm_config_db #(virtual ALSU_if)::set(null,"*","ALSU_K",if_t);
    uvm_config_db #(virtual shift_reg_if)::set(null,"*","SHIFT_K",sr_t);
    run_test("ALSU_test");
end

endmodule
```



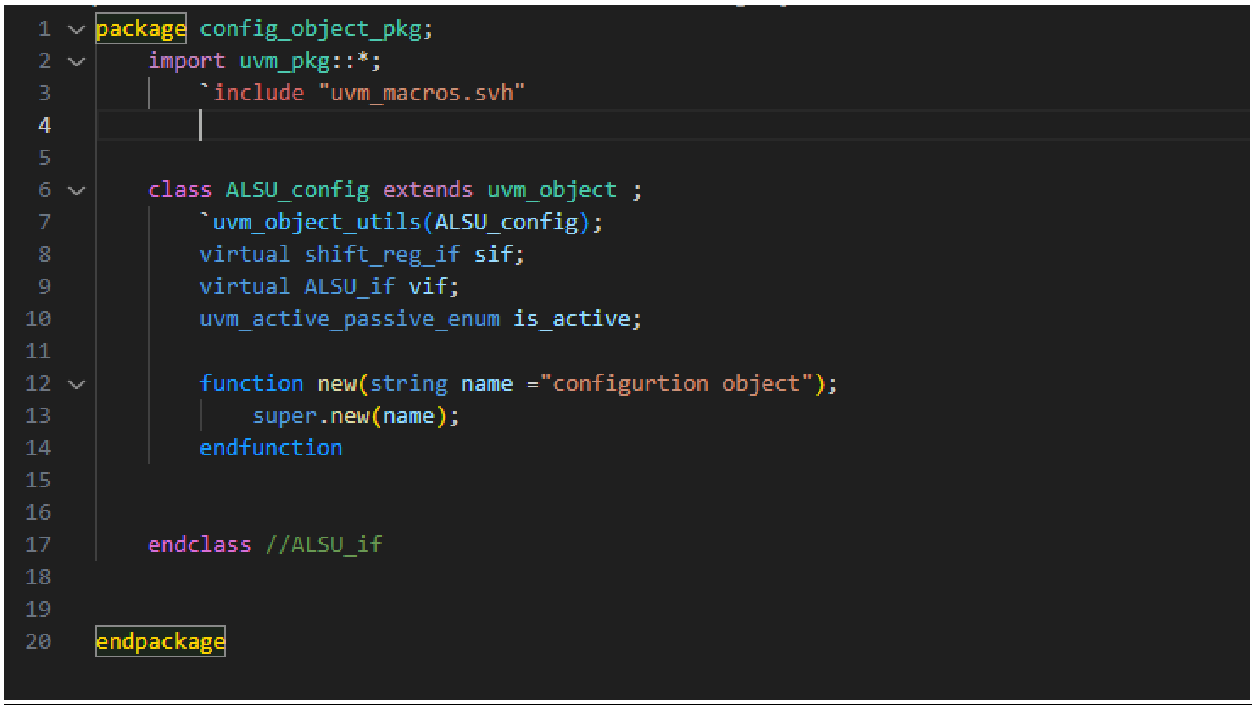
```
package shift_agent_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh"
import shift_monitor_pkg::*;
import shift_driver_pkg::*;
import shift_sequencer_pkg::*;
import config_object_pkg::*;
import shift_seq_item_pkg::*;

    class shift_reg_agent extends uvm_agent;
        `uvm_component_utils(shift_reg_agent);
        shift_reg_monitor mon;
        shift_reg_sequencer sqr;
        shift_reg_driver drv;
        uvm_analysis_port #(shift_seq_item) agt_p;
        ALSU_config cfg;
        uvm_active_passive_enum is_active;
        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);
            mon=shift_reg_monitor::type_id::create("monitor",this);
            agt_p=new("agent_port",this);
            if(!uvm_config_db #(ALSU_config)::get(this,"","shift_cfg",cfg))
                `uvm_fatal("build_phase","cant get shift virtual interface");
            is_active=cfg.is_active;
            if(is_active==UVM_ACTIVE)begin
                sqr=shift_reg_sequencer::type_id::create("driver",this);
                drv=shift_reg_driver::type_id::create("sequencer",this);
```



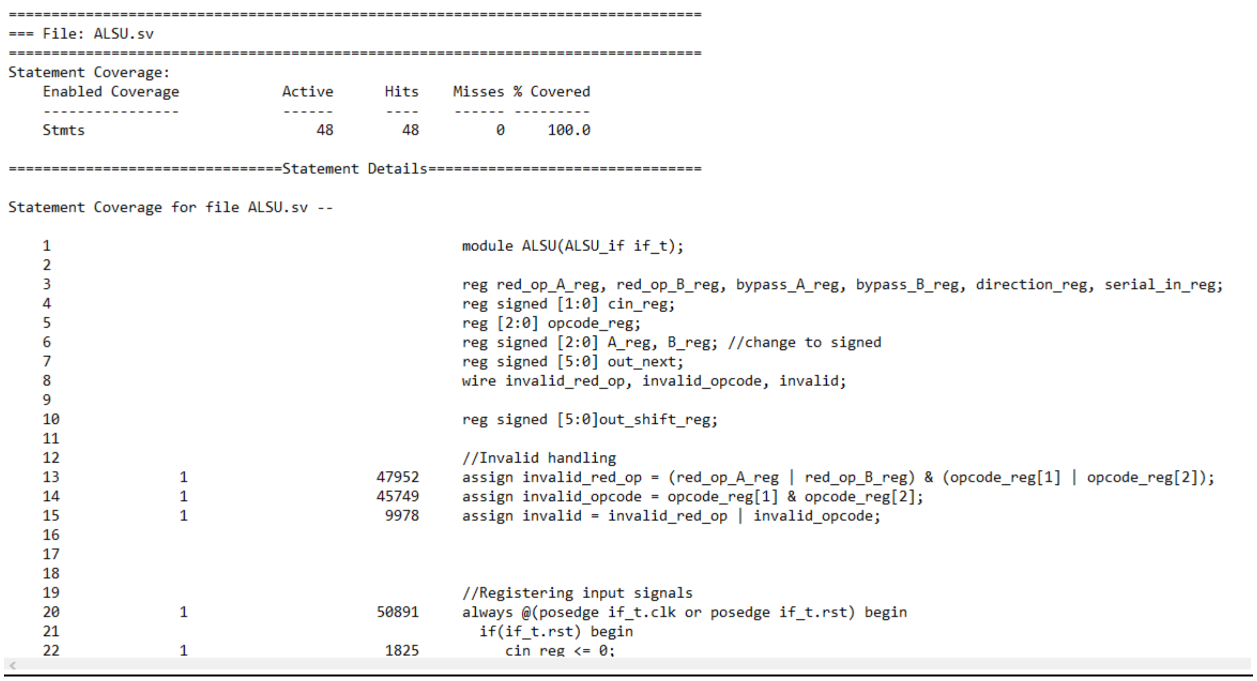
```
                drv=shift_reg_driver::type_id::create("sequencer",this);
            end

        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            mon.mon_p.connect(agt_p);
            mon.vif=cfg.sif;
            if(is_active==UVM_ACTIVE)begin
                drv.seq_item_port.connect(sqr.seq_item_export);
                drv.vif=cfg.sif;
            end
        endfunction


    endclass





endpackage
```



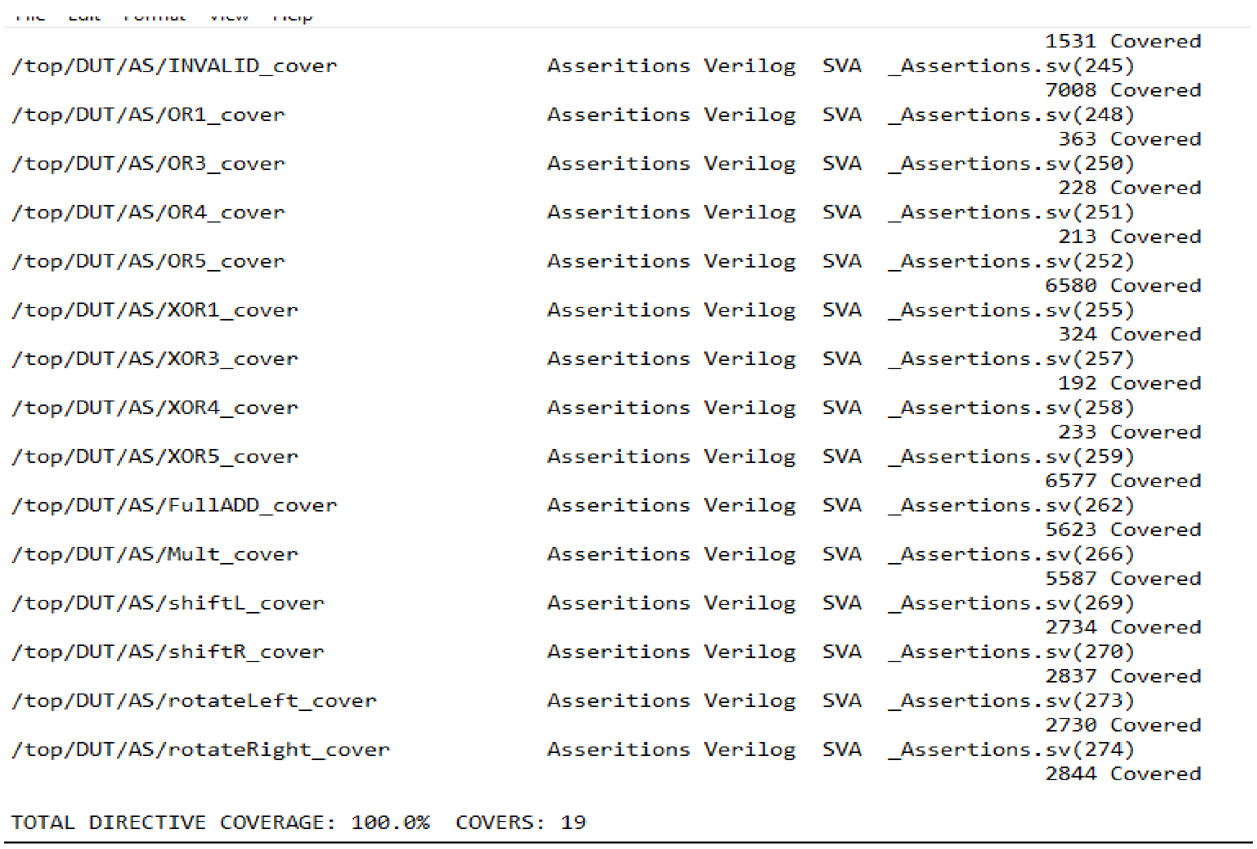
```
package constant_enums;
    typedef enum { OR=0,XOR,ADD,MULT,SHIFT,ROTATE,INVALID6,INVALID7 } Opcode_e;
    typedef enum {MAXPOS=3,MAXNEG=-4,ZERO=0}corner_state_e;

endpackage
```



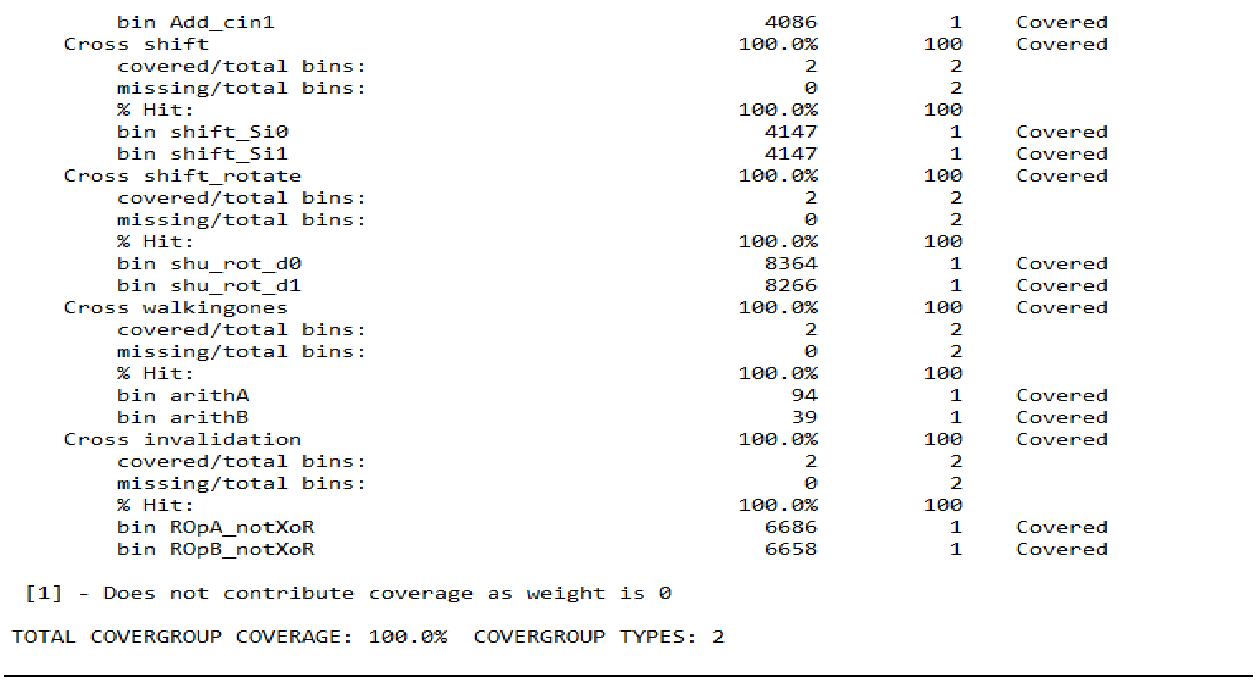
```
package config_object_pkg;
    import uvm_pkg::*;
        `include "uvm_macros.svh"
        

    class ALSU_config extends uvm_object ;
        `uvm_object_utils(ALSU_config);
        virtual shift_reg_if sif;
        virtual ALSU_if vif;
        uvm_active_passive_enum is_active;

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


    endclass //ALSU_if


endpackage
```



```
=================================================================================
=== File: ALSU.sv
=================================================================================
Statement Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           48        48         0     100.0

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

Statement Coverage for file ALSU.sv --

    1                                                module ALSU(ALSU_if if_t);
    2
    3                                                reg red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
    4                                                reg signed [1:0] cin_reg;
    5                                                reg [2:0] opcode_reg;
    6                                                reg signed [2:0] A_reg, B_reg; //change to signed
    7                                                reg signed [5:0] out_next;
    8                                                wire invalid_red_op, invalid_opcode, invalid;
    9
    10                                               reg signed [5:0]out_shift_reg;
    11
    12                                               //Invalid handling
    13              1                    47952       assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
    14              1                    45749       assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
    15              1                     9978       assign invalid = invalid_red_op | invalid_opcode;
    16
    17
    18
    19                                               //Registering input signals
    20              1                    50891       always @(posedge if_t.clk or posedge if_t.rst) begin
    21                                                 if(if_t.rst) begin
    22              1                     1825            cin_reg <= 0;
```

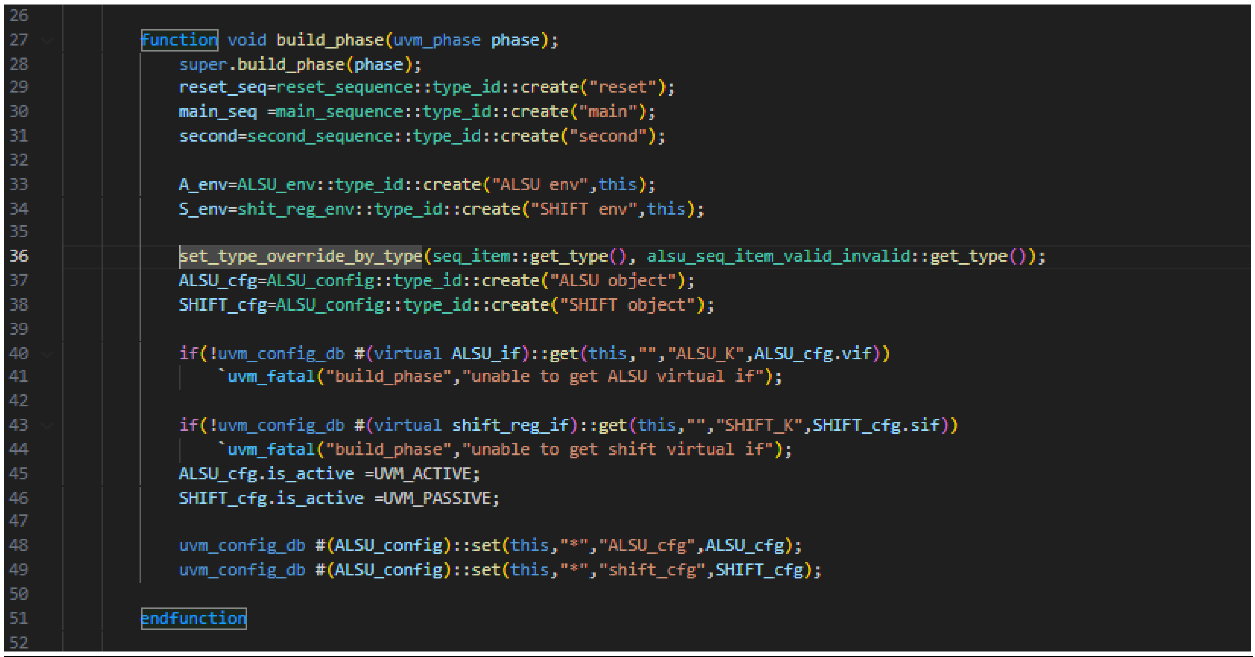


```
                                                                                1531 Covered
/top/DUT/AS/INVALID_cover               Asseritions Verilog  SVA  _Assertions.sv(245)
                                                                                7008 Covered
/top/DUT/AS/OR1_cover                   Asseritions Verilog  SVA  _Assertions.sv(248)
                                                                                 363 Covered
/top/DUT/AS/OR3_cover                   Asseritions Verilog  SVA  _Assertions.sv(250)
                                                                                 228 Covered
/top/DUT/AS/OR4_cover                   Asseritions Verilog  SVA  _Assertions.sv(251)
                                                                                 213 Covered
/top/DUT/AS/OR5_cover                   Asseritions Verilog  SVA  _Assertions.sv(252)
                                                                                6580 Covered
/top/DUT/AS/XOR1_cover                  Asseritions Verilog  SVA  _Assertions.sv(255)
                                                                                 324 Covered
/top/DUT/AS/XOR3_cover                  Asseritions Verilog  SVA  _Assertions.sv(257)
                                                                                 192 Covered
/top/DUT/AS/XOR4_cover                  Asseritions Verilog  SVA  _Assertions.sv(258)
                                                                                 233 Covered
/top/DUT/AS/XOR5_cover                  Asseritions Verilog  SVA  _Assertions.sv(259)
                                                                                6577 Covered
/top/DUT/AS/FullADD_cover               Asseritions Verilog  SVA  _Assertions.sv(262)
                                                                                5623 Covered
/top/DUT/AS/Mult_cover                  Asseritions Verilog  SVA  _Assertions.sv(266)
                                                                                5587 Covered
/top/DUT/AS/shiftL_cover                Asseritions Verilog  SVA  _Assertions.sv(269)
                                                                                2734 Covered
/top/DUT/AS/shiftR_cover                Asseritions Verilog  SVA  _Assertions.sv(270)
                                                                                2837 Covered
/top/DUT/AS/rotateLeft_cover            Asseritions Verilog  SVA  _Assertions.sv(273)
                                                                                2730 Covered
/top/DUT/AS/rotateRight_cover           Asseritions Verilog  SVA  _Assertions.sv(274)
                                                                                2844 Covered

TOTAL DIRECTIVE COVERAGE: 100.0%  COVERS: 19
```

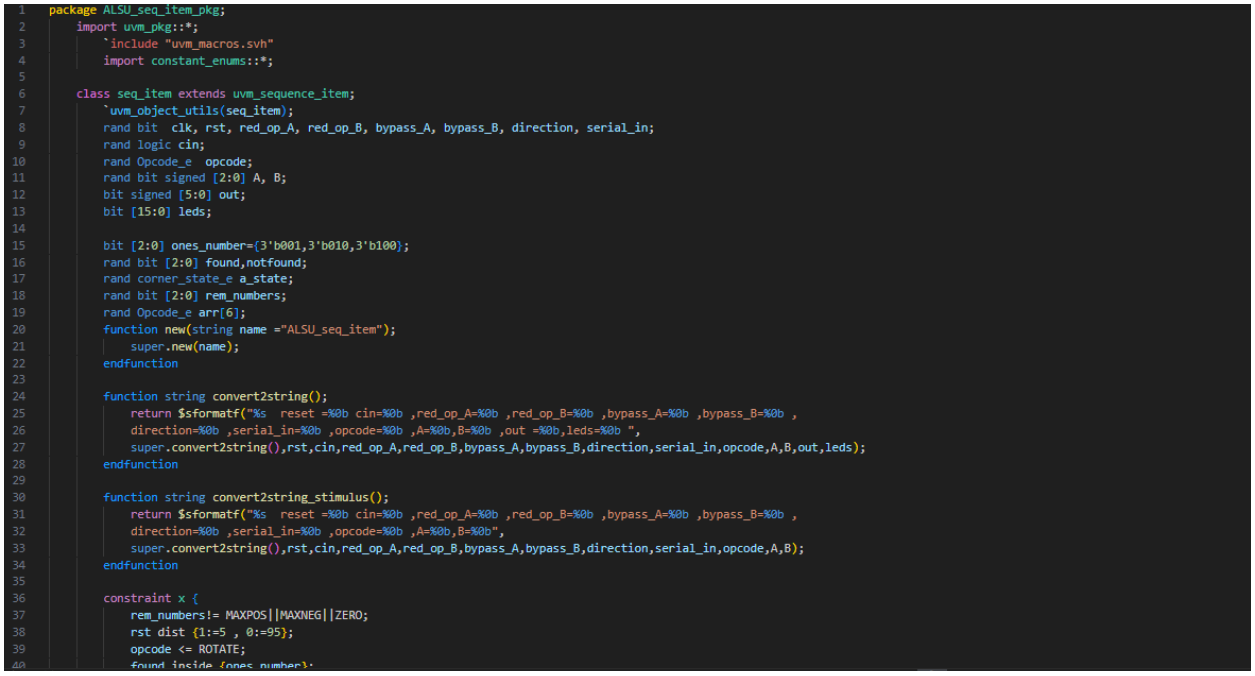


```
        bin Add_cin1                                      4086          1     Covered
    Cross shift                                         100.0%        100     Covered
        covered/total bins:                                  2          2
        missing/total bins:                                  0          2
        % Hit:                                          100.0%        100
        bin shift_Si0                                     4147          1     Covered
        bin shift_Si1                                     4147          1     Covered
    Cross shift_rotate                                  100.0%        100     Covered
        covered/total bins:                                  2          2
        missing/total bins:                                  0          2
        % Hit:                                          100.0%        100
        bin shu_rot_d0                                    8364          1     Covered
        bin shu_rot_d1                                    8266          1     Covered
    Cross walkingones                                   100.0%        100     Covered
        covered/total bins:                                  2          2
        missing/total bins:                                  0          2
        % Hit:                                          100.0%        100
        bin arithA                                          94          1     Covered
        bin arithB                                          39          1     Covered
    Cross invalidation                                  100.0%        100     Covered
        covered/total bins:                                  2          2
        missing/total bins:                                  0          2
        % Hit:                                          100.0%        100
        bin ROpA_notXoR                                   6686          1     Covered
        bin ROpB_notXoR                                   6658          1     Covered

 [1] - Does not contribute coverage as weight is 0

TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 2
```

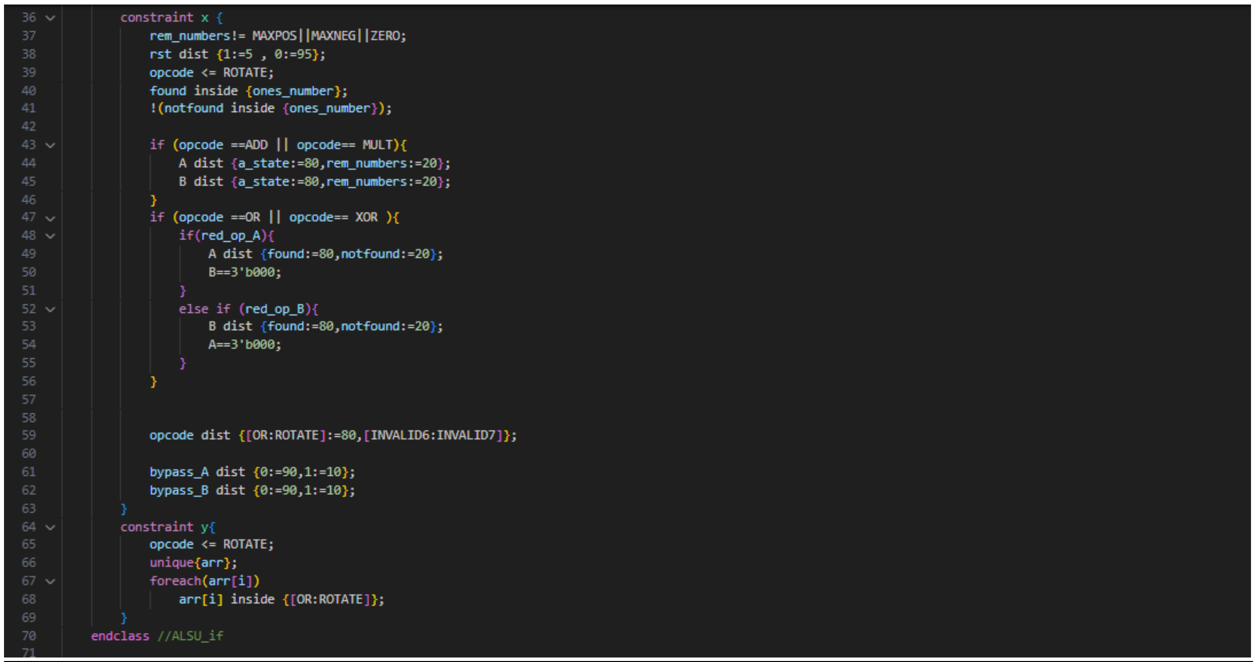
Part 2



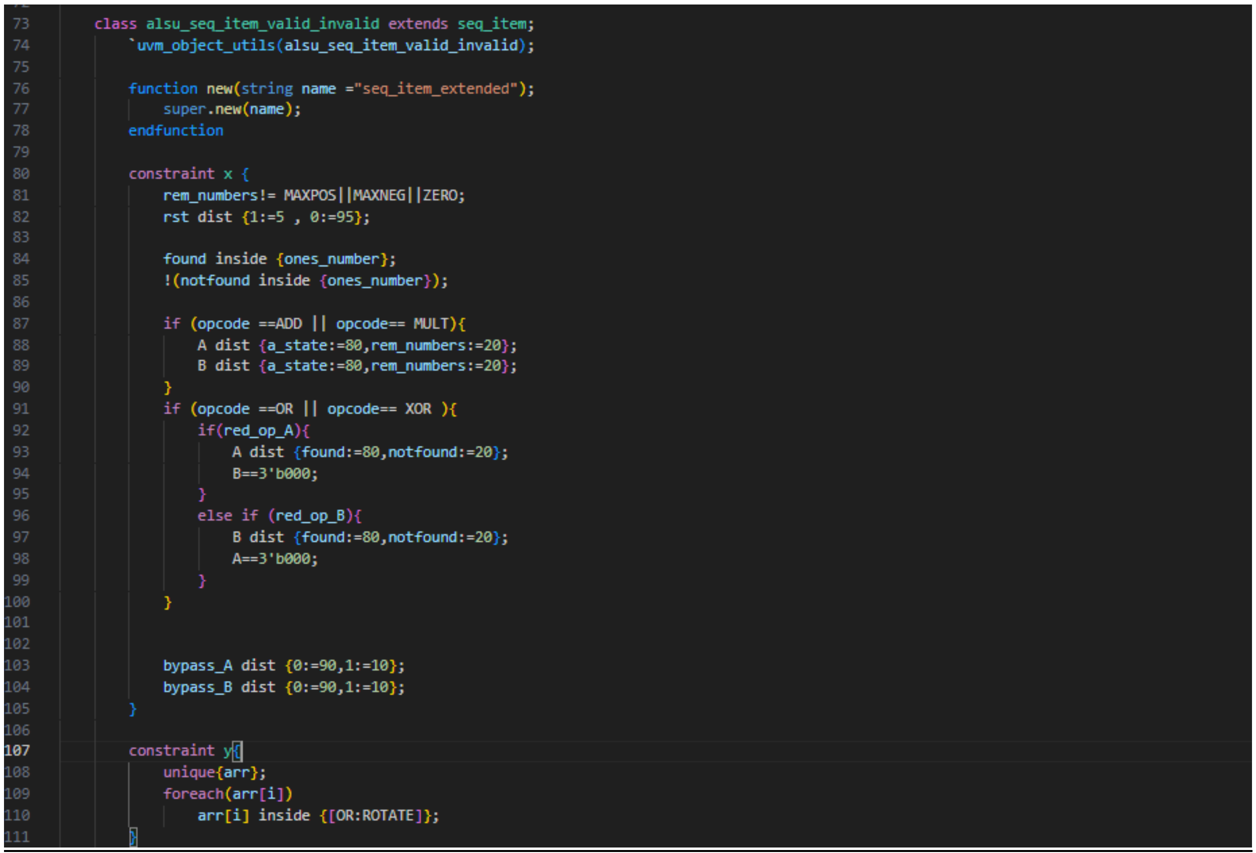
```
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        reset_seq=reset_sequence::type_id::create("reset");
        main_seq =main_sequence::type_id::create("main");
        second=second_sequence::type_id::create("second");

        A_env=ALSU_env::type_id::create("ALSU env",this);
        S_env=shit_reg_env::type_id::create("SHIFT env",this);

        set_type_override_by_type(seq_item::get_type(), alsu_seq_item_valid_invalid::get_type());
        ALSU_cfg=ALSU_config::type_id::create("ALSU object");
        SHIFT_cfg=ALSU_config::type_id::create("SHIFT object");

        if(!uvm_config_db #(virtual ALSU_if)::get(this,"","ALSU_K",ALSU_cfg.vif))
            `uvm_fatal("build_phase","unable to get ALSU virtual if");

        if(!uvm_config_db #(virtual shift_reg_if)::get(this,"","SHIFT_K",SHIFT_cfg.sif))
            `uvm_fatal("build_phase","unable to get shift virtual if");
        ALSU_cfg.is_active =UVM_ACTIVE;
        SHIFT_cfg.is_active =UVM_PASSIVE;

        uvm_config_db #(ALSU_config)::set(this,"*","ALSU_cfg",ALSU_cfg);
        uvm_config_db #(ALSU_config)::set(this,"*","shift_cfg",SHIFT_cfg);

    endfunction

```



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

    class seq_item extends uvm_sequence_item;
        `uvm_object_utils(seq_item);
        rand bit  clk, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
        rand logic cin;
        rand Opcode_e  opcode;
        rand bit signed [2:0] A, B;
        bit signed [5:0] out;
        bit [15:0] leds;

        bit [2:0] ones_number={3'b001,3'b010,3'b100};
        rand bit [2:0] found,notfound;
        rand corner_state_e a_state;
        rand bit [2:0] rem_numbers;
        rand Opcode_e arr[6];
        function new(string name ="ALSU_seq_item");
            super.new(name);
        endfunction

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

        function string convert2string_stimulus();
            return $sformatf("%s  reset =%0b cin=%0b ,red_op_A=%0b ,red_op_B=%0b ,bypass_A=%0b ,bypass_B=%0b ,
            direction=%0b ,serial_in=%0b ,opcode=%0b ,A=%0b,B=%0b",
            super.convert2string(),rst,cin,red_op_A,red_op_B,bypass_A,bypass_B,direction,serial_in,opcode,A,B);
        endfunction

        constraint x {
            rem_numbers!= MAXPOS||MAXNEG||ZERO;
            rst dist {1:=5 , 0:=95};
            opcode <= ROTATE;
            found inside {ones_number};
```

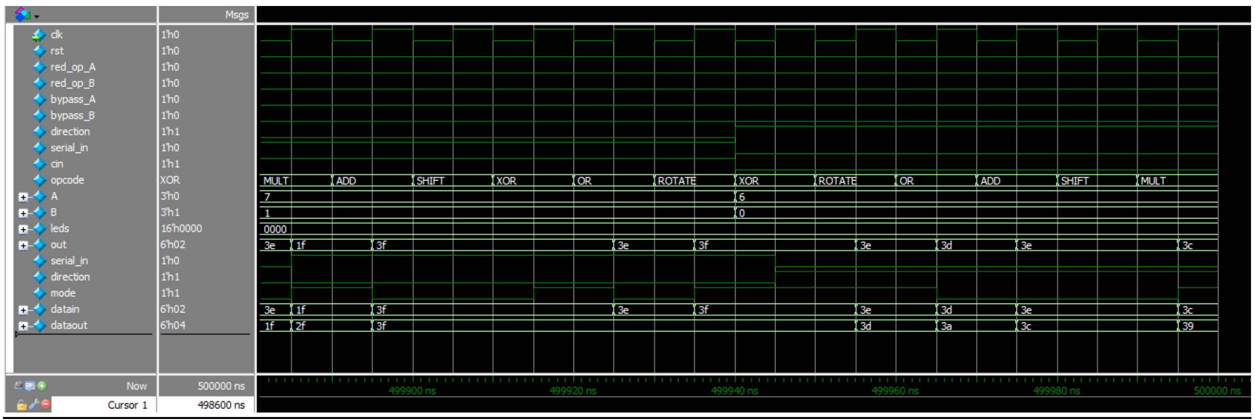


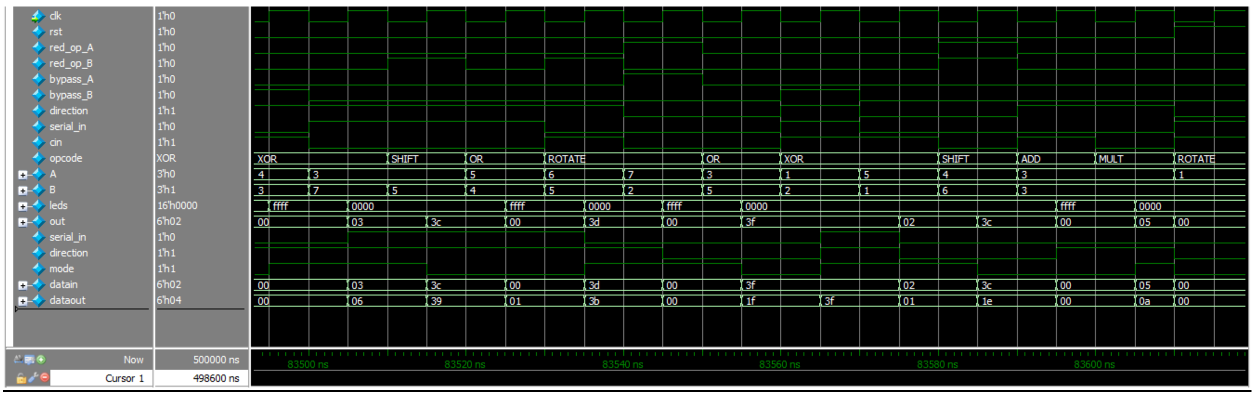
```
    constraint x {
        rem_numbers!= MAXPOS||MAXNEG||ZERO;
        rst dist {1:=5 , 0:=95};
        opcode <= ROTATE;
        found inside {ones_number};
        !(notfound inside {ones_number});

        if (opcode ==ADD || opcode== MULT){
            A dist {a_state:=80,rem_numbers:=20};
            B dist {a_state:=80,rem_numbers:=20};
        }
        if (opcode ==OR || opcode== XOR ){
            if(red_op_A){
                A dist {found:=80,notfound:=20};
                B==3'b000;
            }
            else if (red_op_B){
                B dist {found:=80,notfound:=20};
                A==3'b000;
            }
        }


        opcode dist {[OR:ROTATE]:=80,[INVALID6:INVALID7]};

        bypass_A dist {0:=90,1:=10};
        bypass_B dist {0:=90,1:=10};
    }
    constraint y{
        opcode <= ROTATE;
        unique{arr};
        foreach(arr[i])
            arr[i] inside {[OR:ROTATE]};
    }
endclass //ALSU_if
```



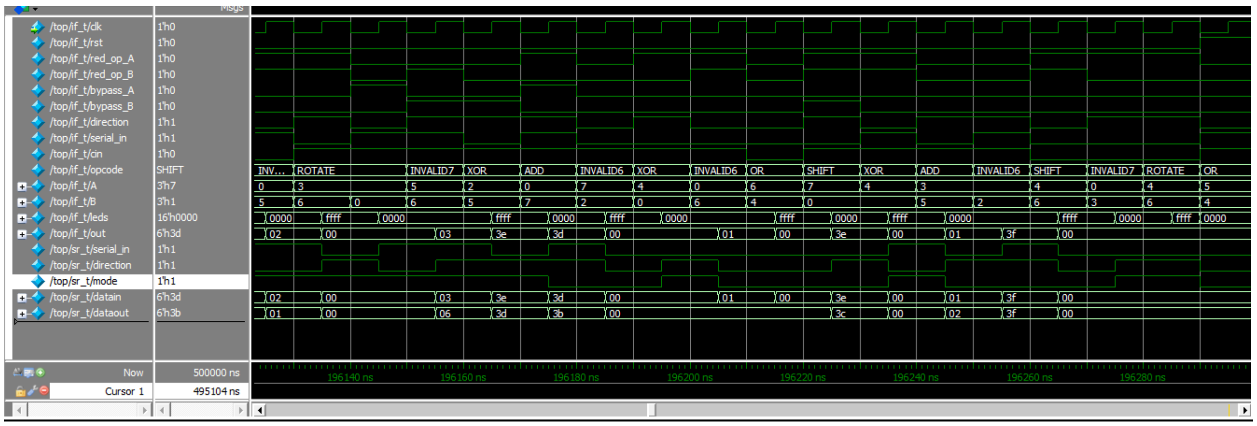
```
class alsu_seq_item_valid_invalid extends seq_item;
    `uvm_object_utils(alsu_seq_item_valid_invalid);

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

    constraint x {
        rem_numbers!= MAXPOS||MAXNEG||ZERO;
        rst dist {1:=5 , 0:=95};

        found inside {ones_number};
        !(notfound inside {ones_number});

        if (opcode ==ADD || opcode== MULT){
            A dist {a_state:=80,rem_numbers:=20};
            B dist {a_state:=80,rem_numbers:=20};
        }
        if (opcode ==OR || opcode== XOR ){
            if(red_op_A){
                A dist {found:=80,notfound:=20};
                B==3'b000;
            }
            else if (red_op_B){
                B dist {found:=80,notfound:=20};
                A==3'b000;
            }
        }


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

    constraint y{
        unique{arr};
        foreach(arr[i])
            arr[i] inside {[OR:ROTATE]};
    }
```

valid





Invalid



/top/if_t/clk 1'h0
/top/if_t/rst 1'h0
/top/if_t/red_op_A 1'h0
/top/if_t/red_op_B 1'h0
/top/if_t/bypass_A 1'h0
/top/if_t/bypass_B 1'h0
/top/if_t/direction 1'h1
/top/if_t/serial_in 1'h1
/top/if_t/cin 1'h0
/top/if_t/opcode SHIFT
/top/if_t/A 3'h7
/top/if_t/B 3'h1
/top/if_t/leds 16'h0000
/top/if_t/out 6'h3d
/top/sr_t/serial_in 1'h1
/top/sr_t/direction 1'h1
/top/sr_t/mode 1'h1
/top/sr_t/datain 6'h3d
/top/sr_t/dataout 6'h3b
Now 500000 ns
Cursor 1 495104 ns