MARIAM MOHAMED MARZOUK

ASSIGNMENT_6_Extended

ALSU:

```
module ALSU(ALSU_if alif,shift_reg_if sh_if);
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;
reg signed [5:0] out_next;
wire invalid_red_op, invalid_opcode, invalid;
reg signed [5:0]out_shift_reg;
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;
assign sh_if.serial_in=serial_in_reg;
assign sh_if.direction=direction_reg;
assign sh_if.mode=opcode_reg[0];
assign sh_if.datain=alif.out;
assign out_shift_reg=sh_if.dataout;
always @(posedge alif.clk or posedge alif.reset) begin
  if(alif.reset) 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 <= alif.cin;
     red_op_B_reg <= alif.red_op_B;red_op_A_reg <= alif.red_op_A;
     bypass_B_reg <= alif.bypass_B;bypass_A_reg <= alif.bypass_A;
     direction_reg <= alif.direction;serial_in_reg <= alif.serial_in;
     opcode_reg <= alif.opcode;A_reg <= alif.A;B_reg <= alif.B;
  end end
always @(posedge alif.clk or posedge alif.reset) begin
  if(alif.reset) begin
     alif.leds <= 0;
  end else begin
      if (invalid)
        alif.leds <= ~alif.leds; else alif.leds <= 0;
  end end
always @(posedge alif.clk or posedge alif.reset) begin
  if(alif.reset) begin
    alif.out <= 0;
  end else begin
    if (bypass_A_reg && bypass_B_reg)
      alif.out <= (alif.INPUT_PRIORITY == "A")?A_reg: B_reg;
    else if (bypass_A_reg)
      alif.out <= A_reg;
    else if (bypass_B_reg)
      alif.out <= B_reg;
    else if (invalid) // cahnge the priority of invalid bits after bypass_reg
        alif.out <= 0;
    else begin
        case (opcode_reg)
          3'h0: begin //change Opcode to OR not AND
            if (red_op_A_reg && red_op_B_reg)
              alif.out <= (alif.INPUT_PRIORITY == "A")? |A_reg: |B_reg;
            else if (red_op_A_reg)
              alif.out <= |A_reg;
            else if (red_op_B_reg)
              alif.out <= |B_reg;
            else
              alif.out <= A_reg | B_reg;
          end
          3'h1: begin // change opcode to XOR not OR
            if (red_op_A_reg && red_op_B_reg)
              alif.out <= (alif.INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
            else if (red_op_A_reg)
              alif.out <= ^A_reg;
            else if (red_op_B_reg)
              alif.out <= ^B_reg;
            else
              alif.out <= A_reg ^ B_reg;
          end
          3'h2: begin //here we add condition to check full adder if ON or OFF
            if(alif.FULL_ADDER == "ON")
              alif.out <= A_reg + B_reg+cin_reg;
            else if(alif.FULL_ADDER == "OFF")
              alif.out <= A_reg + B_reg;
          end
          3'h3: alif.out <= A_reg * B_reg;
          3'h4: begin //100
              alif.out <= out_shift_reg;
          end
          3'h5: begin//101
              alif.out <= out_shift_reg;
          end
        default: alif.out<=alif.out;
        endcase
    end end
  out_next<=alif.out;
end
endmodule
```

ALSU INTERFACE:

```
interface ALSU_if(clk);

input bit clk;

parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
logic reset, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
logic signed cin;
logic  [2:0] opcode;
logic signed [2:0] A, B;
logic  [15:0] leds;
logic signed[5:0] out;

endinterface
```

ALSU DRIVER:

```
package ALSU_driver_pkg;
    import uvm_pkg::*;
        `include "uvm_macros.svh"
    import ALSU_seq_item_pkg::*;
    class ALSU_driver extends uvm_driver #(seq_item);
        `uvm_component_utils(ALSU_driver);
        virtual ALSU_if alsu_config_vif;
        seq_item item;
        function new(string name ="driver",uvm_component parent=null);
            super.new(name,parent);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                item=seq_item::type_id::create("seq item");
                seq_item_port.get_next_item(item);
                alsu_config_vif.reset=item.reset;
                alsu_config_vif.A=item.A;
                alsu_config_vif.B=item.B;
                alsu_config_vif.cin=item.cin;
                alsu_config_vif.red_op_A=item.red_op_A;
                alsu_config_vif.red_op_B=item.red_op_B;
                alsu_config_vif.bypass_A=item.bypass_A;
                alsu_config_vif.bypass_B=item.bypass_B;
                alsu_config_vif.direction=item.direction;
                alsu_config_vif.serial_in=item.serial_in;
                alsu_config_vif.opcode=item.opcode;
                @(negedge alsu_config_vif.clk);
                seq_item_port.item_done();
            end
        endtask
    endclass //ALSU_if
endpackage
```

ALSU ENVIROMENT:

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

    class ALSU_env extends uvm_env;
        `uvm_component_utils(ALSU_env);
        ALSU_coverage cov;
        ALSU_agent agt;

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

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agt.agt_p.connect(cov.cov_ep);
        endfunction

    endclass //ALSU_if


endpackage
```

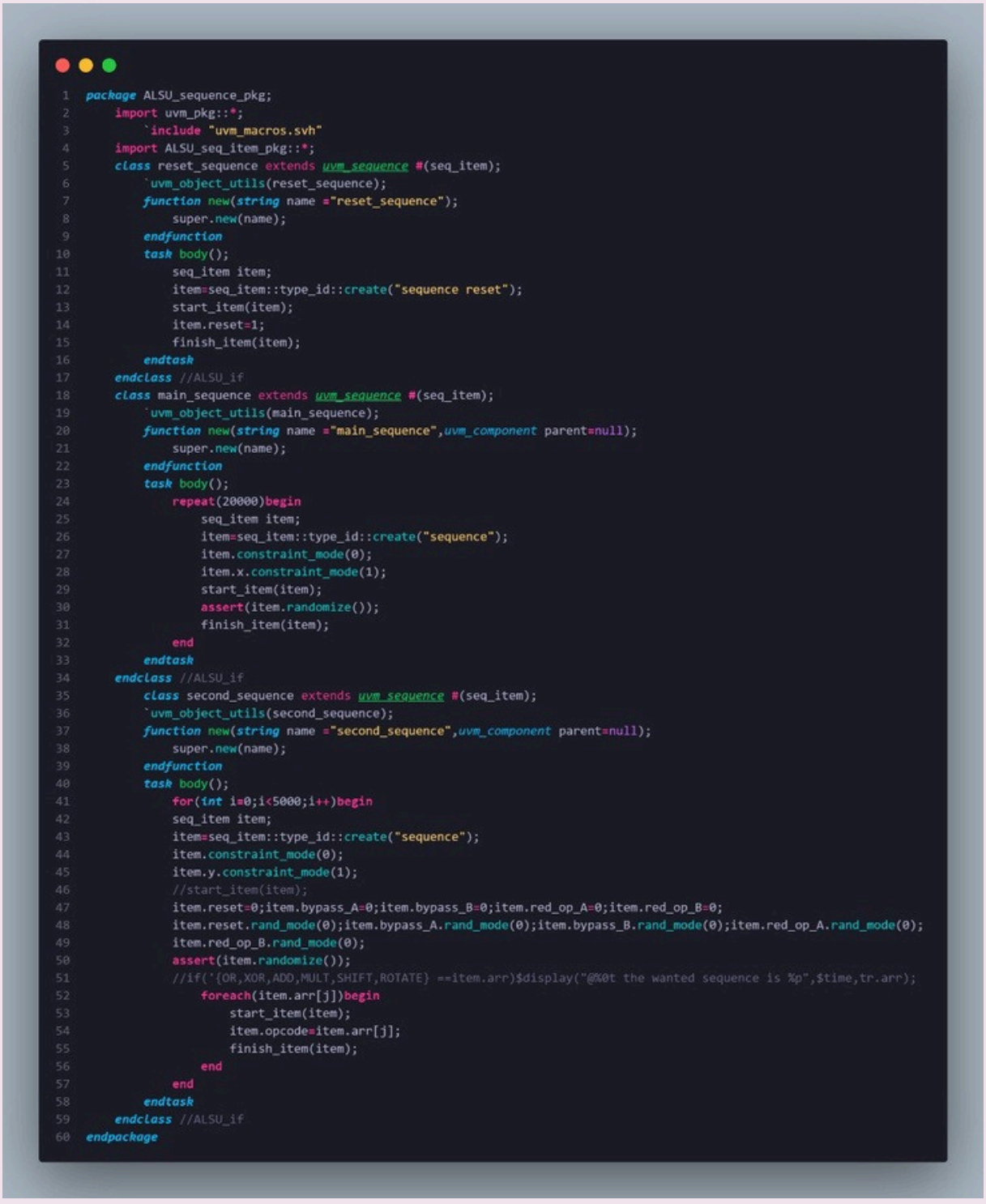
ALSU MONITOR:

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

    class ALSU_mointor extends uvm_monitor ;
        `uvm_component_utils(ALSU_mointor);
        virtual ALSU_if alsu_config_vif;
        seq_item item;
        uvm_analysis_port #(seq_item) mon_p;
        function new(string name ="",uvm_component parent=null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            mon_p=new("alsu monitor port",this);
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                item=seq_item::type_id::create("items recived");
                @(negedge alsu_config_vif.clk);
                item.reset=alsu_config_vif.reset;
                item.A=alsu_config_vif.A;
                item.B=alsu_config_vif.B;
                item.cin=alsu_config_vif.cin;
                item.red_op_A=alsu_config_vif.red_op_A;
                item.red_op_B=alsu_config_vif.red_op_B;
                item.bypass_A=alsu_config_vif.bypass_A;
                item.bypass_B=alsu_config_vif.bypass_B;
                item.direction=alsu_config_vif.direction;
                item.serial_in=alsu_config_vif.serial_in;
                item.opcode=alsu_config_vif.opcode;
                mon_p.write(item);
                 `uvm_info("run_phase",item.convert2string(),UVM_HIGH);
            end
        endtask
    endclass //ALSU_if
endpackage
```

ALSU AGENT:

```
package ALSU_agent_pkg;
    import uvm_pkg::*;
        `include "uvm_macros.svh"
    import ALSU_monitor_pkg::*;
    import ALSU_sequencer_pkg::*;
    import ALSU_driver_pkg::*;
    import ALSU_seq_item_pkg::*;
    import config_object_pkg::*;

    class  ALSU_agent extends uvm_agent;
        `uvm_component_utils(ALSU_agent);
        ALSU_mointor mon;
        ALSU_sequencer sqr;
        ALSU_driver driver;
        ALSU_config cfg;
        uvm_analysis_port #(seq_item) agt_p;
        uvm_active_passive_enum is_active;


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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);

            mon=ALSU_mointor::type_id::create("monitor",this);
            agt_p=new("agent port",this);
            if(!uvm_config_db #(ALSU_config)::get(this,"","ALSU_cfg",cfg))
                `uvm_fatal("build_phase","cant get ALSU virtual interface");
            is_active=cfg.is_active;
            if(is_active==UVM_ACTIVE)begin
                sqr=ALSU_sequencer::type_id::create("sequencer",this);
                driver=ALSU_driver::type_id::create("driver",this);
            end
        endfunction

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


    endclass //ALSU_if


endpackage
```

ALSU SEQUENCE:



```
package ALSU_sequence_pkg;
    import uvm_pkg::*;
        `include "uvm_macros.svh"
    import ALSU_seq_item_pkg::*;
    class reset_sequence extends uvm_sequence #(seq_item);
        `uvm_object_utils(reset_sequence);
        function new(string name ="reset_sequence");
            super.new(name);
        endfunction
        task body();
            seq_item item;
            item=seq_item::type_id::create("sequence reset");
            start_item(item);
            item.reset=1;
            finish_item(item);
        endtask
    endclass //ALSU_if
    class main_sequence extends uvm_sequence #(seq_item);
        `uvm_object_utils(main_sequence);
        function new(string name ="main_sequence",uvm_component parent=null);
            super.new(name);
        endfunction
        task body();
            repeat(20000)begin
                seq_item item;
                item=seq_item::type_id::create("sequence");
                item.constraint_mode(0);
                item.x.constraint_mode(1);
                start_item(item);
                assert(item.randomize());
                finish_item(item);
            end
        endtask
    endclass //ALSU_if
        class second_sequence extends uvm_sequence #(seq_item);
        `uvm_object_utils(second_sequence);
        function new(string name ="second_sequence",uvm_component parent=null);
            super.new(name);
        endfunction
        task body();
            for(int i=0;i<5000;i++)begin
            seq_item item;
            item=seq_item::type_id::create("sequence");
            item.constraint_mode(0);
            item.y.constraint_mode(1);
            //start_item(item);
            item.reset=0;item.bypass_A=0;item.bypass_B=0;item.red_op_A=0;item.red_op_B=0;
            item.reset.rand_mode(0);item.bypass_A.rand_mode(0);item.bypass_B.rand_mode(0);item.red_op_A.rand_mode(0);
            item.red_op_B.rand_mode(0);
            assert(item.randomize());
            //if('{OR,XOR,ADD,MULT,SHIFT,ROTATE} ==item.arr)$display("@%0t the wanted sequence is %p",$time,tr.arr);
                foreach(item.arr[j])begin
                    start_item(item);
                    item.opcode=item.arr[j];
                    finish_item(item);
                end
            end
        endtask
    endclass //ALSU_if
endpackage
```

ALSU SEQUENCER:

```
1  package ALSU_sequencer_pkg;
2      import uvm_pkg::*;
3          `include "uvm_macros.svh"
4      import ALSU_seq_item_pkg::*;
5
6      class ALSU_sequencer extends uvm_sequencer #(seq_item);
7          `uvm_component_utils(ALSU_sequencer);
8
9          function new(string name ="sequencer",uvm_component parent=null);
10             super.new(name,parent);
11         endfunction
12
13
14     endclass //ALSU_if
15
16
17 endpackage
```

ALSU SEQ ITEM:

```
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, reset, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
        rand logic cin;
        rand logic  [2:0] 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 ="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(),reset,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(),reset,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;
            reset dist {1:=5 , 0:=95};

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

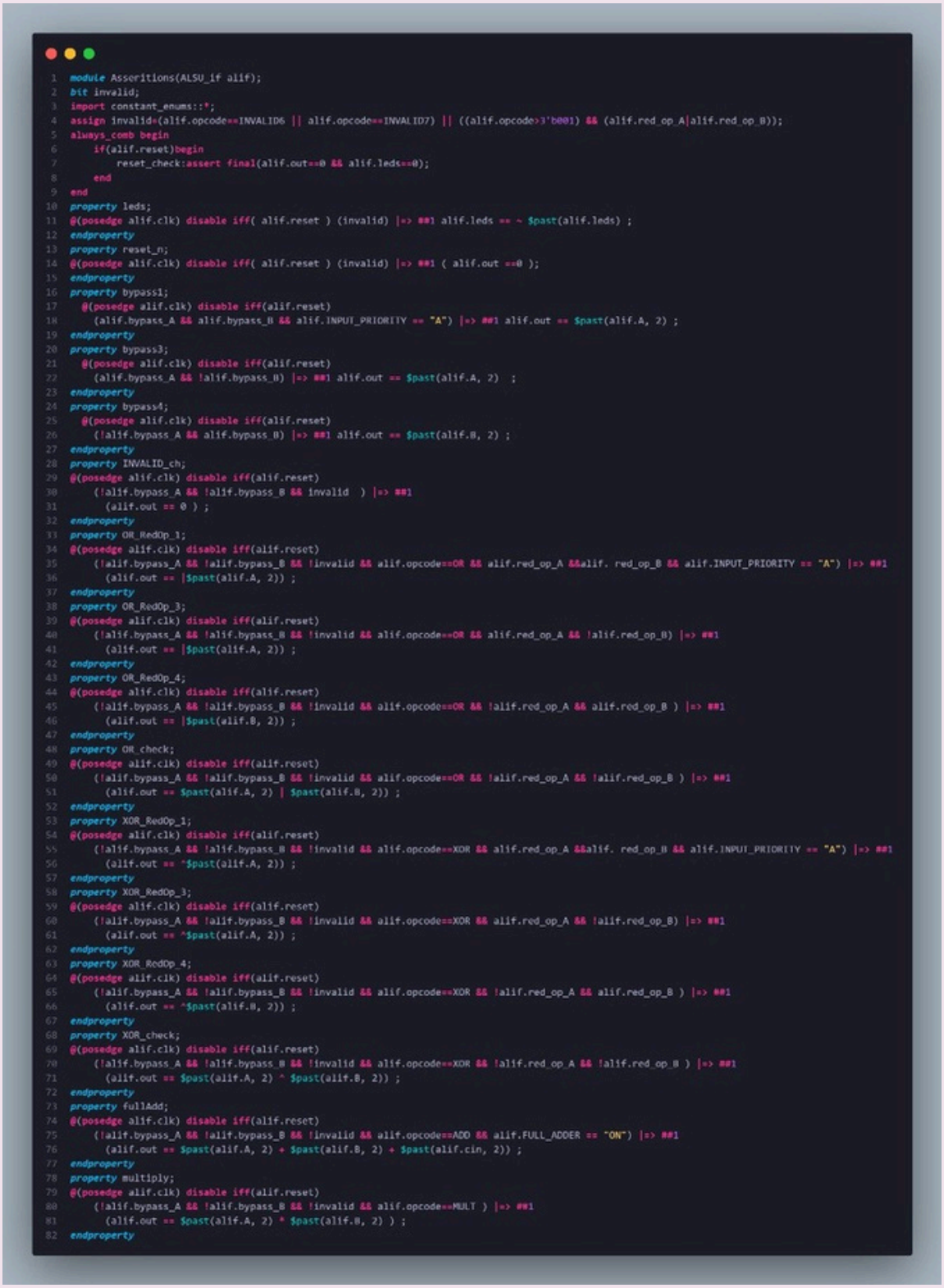
```
                if (opcode ==ADD || opcode== MULT){
                    A dist {a_state:=80,rem_numbers:=20};
                    B dist {a_state:=80,rem_numbers:=20};
                }
                if (opcode ==OR || opcode== XOR ){
                    if(red_op_A){
                        A dist {found:=80,notfound:=20};
                        B==3'b000;
                    }
                    else if (red_op_B){
                        B dist {found:=80,notfound:=20};
                        A==3'b000;
                    }}
                opcode dist {[OR:ROTATE]:=80,[INVALID6:INVALID7]};

                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]};
            }
        endclass //ALSU_if
    endpackage
```

ALSU COVERAGE:

```
package ALSU_coverage_pkg;
    import uvm_pkg::*;
        `include "uvm_macros.svh"
    import ALSU_seq_item_pkg::*;
            import constant_enums::*;
    class ALSU_coverage extends uvm_component;
        `uvm_component_utils(ALSU_coverage);
        seq_item item;
        uvm_analysis_export #(seq_item)cov_ep;
        uvm_tlm_analysis_fifo #(seq_item) cov_fifo;
        covergroup cvr_gp;
        CIN:coverpoint item.cin{
            bins CIN0={0};
            bins CIN1={1};
            option.weight=0;
        }
        red_op_A:coverpoint item.red_op_A{
            bins red_op_A0={0};
            bins red_op_A1={1};
            option.weight=0;
        }
        red_op_B:coverpoint item.red_op_B{
            bins red_op_B0={0};
            bins red_op_B1={1};
            option.weight=0;
        }
        bypass_A:coverpoint item.bypass_A{
            bins bypass_A0={0};
            bins bypass_A1={1};
            option.weight=0;
        }
        bypass_B:coverpoint item.bypass_B{
            bins bypass_B0={0};
            bins bypass_B1={1};
            option.weight=0;
        }
        direction:coverpoint item.direction{
            bins direction0={0};
            bins direction1={1};
            option.weight=0;
        }
        serial_in:coverpoint item.serial_in{
            bins serial_in0={0};
            bins serial_in1={1};
            option.weight=0;
        }
        special:coverpoint item.opcode{
            option.weight=0;
            bins operataions[]={[OR:ROTATE]};
        }
        CB1:coverpoint item.A{
            bins A_data_0={0};
            bins A_data_max={MAXPOS};
            bins A_data_min={MAXNEG};
            bins A_data_walkingones[] ={3'b001,3'b010,3'b100} iff (item.red_op_A);
            bins A_data_default=default;
        }
        CB2:coverpoint item.B{
            bins B_data_0={0};
            bins B_data_max={MAXPOS};
            bins B_data_min={MAXNEG};
            bins B_data_walkingones[] ={3'b001,3'b010,3'b100} iff (item.red_op_B);
            bins B_data_default=default;
        }
        A_M:coverpoint item.opcode{
            bins Bins_arith[] ={ADD,MULT};
        }
```

```
CB3:coverpoint item.opcode{
            bins Bins_shift[]={SHIFT,ROTATE};
            bins Bins_arith[] ={ADD,MULT};
            bins Bins_bitwise[] ={OR,XOR};
            illegal_bins Bins_invalid ={INVALID6,INVALID7};
            bins Bins_trans={(OR=>XOR=>ADD=>MULT=>SHIFT=>ROTATE);
        }
        //1
        corner_case:cross A_M,CB2,CB1{
            ignore_bins walkingA=binsof(CB1.A_data_walkingones) ;
            ignore_bins walkingB=binsof(CB2.B_data_walkingones) ;
        }
        //2
        Addation:cross CIN,special{
            option.cross_auto_bin_max=0;
            bins Add_cin0=binsof(special) intersect {ADD} && binsof(CIN.CIN0) ;
            bins Add_cin1=binsof(special) intersect {ADD} && binsof(CIN.CIN1) ;
        }
        //3
        shift:cross special,serial_in{
            option.cross_auto_bin_max=0;
            bins shift_Si0=binsof(special) intersect {SHIFT} && binsof(serial_in.serial_in0) ;
            bins shift_Si1=binsof(special) intersect {SHIFT} && binsof(serial_in.serial_in0) ;
        }
        //4
        shift_rotate:cross CB3,direction{
            option.cross_auto_bin_max=0;
            bins shu_rot_d0=binsof(CB3.Bins_shift) && binsof(direction.direction0) ;
            bins shu_rot_d1=binsof(CB3.Bins_shift) && binsof(direction.direction1) ;
        }
        //5
        walkingones:cross CB3,CB2,CB1{
            option.cross_auto_bin_max=0;
            bins arithA=binsof(CB3.Bins_bitwise) && binsof(CB2.B_data_0) &&binsof(CB1.A_data_walkingones) ;
            bins arithB=binsof(CB3.Bins_bitwise) && binsof(CB1.A_data_0) &&binsof(CB2.B_data_walkingones) ;
        }
        invalidation:cross red_op_A,red_op_B,special{
        option.cross_auto_bin_max=0;
            bins ROpA_notXoR=binsof(special) intersect{[ADD:ROTATE]} && binsof(red_op_A.red_op_A1)  ;
            bins ROpB_notXoR=binsof(special) intersect{[ADD:ROTATE]} && binsof(red_op_B.red_op_B1)  ;
        }
    endgroup
        function new(string name ="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_fifo=new("fifo",this);
            cov_ep=new("cov_export",this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cov_ep.connect(cov_fifo.analysis_export);
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(item);
                cvr_gp.sample();
            end
        endtask
    endclass //ALSU_if
endpackage
```

ASSERTIONS:



```
module Asseritions(ALSU_if alif);
bit invalid;
import constant_enums::*;
assign invalid=(alif.opcode==INVALID6 || alif.opcode==INVALID7) || ((alif.opcode>3'b001) && (alif.red_op_A|alif.red_op_B));
always_comb begin
    if(alif.reset)begin
        reset_check:assert final(alif.out==0 && alif.leds==0);
    end
end
property leds;
@(posedge alif.clk) disable iff( alif.reset ) (invalid) |=> ##1 alif.leds == ~ $past(alif.leds) ;
endproperty
property reset_n;
@(posedge alif.clk) disable iff( alif.reset ) (invalid) |=> ##1 ( alif.out ==0 );
endproperty
property bypass1;
  @(posedge alif.clk) disable iff(alif.reset)
    (alif.bypass_A && alif.bypass_B && alif.INPUT_PRIORITY == "A") |=> ##1 alif.out == $past(alif.A, 2) ;
endproperty
property bypass3;
  @(posedge alif.clk) disable iff(alif.reset)
    (alif.bypass_A && !alif.bypass_B) |=> ##1 alif.out == $past(alif.A, 2)  ;
endproperty
property bypass4;
  @(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && alif.bypass_B) |=> ##1 alif.out == $past(alif.B, 2) ;
endproperty
property INVALID_ch;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && invalid  ) |=> ##1
      (alif.out == 0 ) ;
endproperty
property OR_RedOp_1;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==OR && alif.red_op_A &&alif. red_op_B && alif.INPUT_PRIORITY == "A") |=> ##1
      (alif.out == |$past(alif.A, 2)) ;
endproperty
property OR_RedOp_3;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==OR && alif.red_op_A && !alif.red_op_B) |=> ##1
      (alif.out == |$past(alif.A, 2)) ;
endproperty
property OR_RedOp_4;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==OR && !alif.red_op_A && alif.red_op_B ) |=> ##1
      (alif.out == |$past(alif.B, 2)) ;
endproperty
property OR_check;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==OR && !alif.red_op_A && !alif.red_op_B ) |=> ##1
      (alif.out == $past(alif.A, 2) | $past(alif.B, 2)) ;
endproperty
property XOR_RedOp_1;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==XOR && alif.red_op_A &&alif. red_op_B && alif.INPUT_PRIORITY == "A") |=> ##1
      (alif.out == ^$past(alif.A, 2)) ;
endproperty
property XOR_RedOp_3;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==XOR && alif.red_op_A && !alif.red_op_B) |=> ##1
      (alif.out == ^$past(alif.A, 2)) ;
endproperty
property XOR_RedOp_4;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==XOR && !alif.red_op_A && alif.red_op_B ) |=> ##1
      (alif.out == ^$past(alif.B, 2)) ;
endproperty
property XOR_check;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==XOR && !alif.red_op_A && !alif.red_op_B ) |=> ##1
      (alif.out == $past(alif.A, 2) ^ $past(alif.B, 2)) ;
endproperty
property fullAdd;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==ADD && alif.FULL_ADDER == "ON") |=> ##1
      (alif.out == $past(alif.A, 2) + $past(alif.B, 2) + $past(alif.cin, 2)) ;
endproperty
property multiply;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==MULT ) |=> ##1
      (alif.out == $past(alif.A, 2) * $past(alif.B, 2) ) ;
endproperty
```

```
property shift_right;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==SHIFT && alif.direction==0 ) |=> ##1
        (alif.out =={ $past(alif.serial_in,2) , $past(alif.out[5:1]) } ) ;
endproperty
property shift_left;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==SHIFT && alif.direction==1 ) |=> ##1
        (alif.out =={$past(alif.out[4:0]) , $past(alif.serial_in,2) } ) ;
endproperty
property rotate_right;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==ROTATE && alif.direction==0 ) |=> ##1
        (alif.out =={$past(alif.out[0]) , $past(alif.out[5:1]) } ) ;
endproperty
property rotate_left;
@(posedge alif.clk) disable iff(alif.reset)
    (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==ROTATE && alif.direction==1 ) |=> ##1
        (alif.out =={$past(alif.out[4:0]) , $past(alif.out[5])} ) ;
endproperty
leds_check:assert property(leds);
bypass1_check:assert property(bypass1);
bypass3_check:assert property(bypass3);
bypass4_check:assert property(bypass4);
INVALID_check:assert property(INVALID_ch);
OR1_check:assert property(OR_RedOp_1);
OR3_check:assert property(OR_RedOp_3);
OR4_check:assert property(OR_RedOp_4);
OR5_check:assert property(OR_check);
XOR1_check:assert property(XOR_RedOp_1);
XOR3_check:assert property(XOR_RedOp_3);
XOR4_check:assert property(XOR_RedOp_4);
XOR5_check:assert property(XOR_check);
FullADD_check:assert property(fullAdd);
shift_R_check:assert property(shift_right) ;
shift_L_check:assert property(shift_left);
rotate_R_check:assert property(rotate_right);
rotate_L_check:assert property(rotate_left);
leds_cover:cover property(leds);
bypass1_cover:cover property(bypass1);
bypass3_cover:cover property(bypass3);
bypass4_cover:cover property(bypass4);
INVALID_cover:cover property(INVALID_ch);
OR1_cover:cover property(OR_RedOp_1);
OR3_cover:cover property(OR_RedOp_3);
OR4_cover:cover property(OR_RedOp_4);
OR5_cover:cover property(OR_check);
XOR1_cover:cover property(XOR_RedOp_1);
XOR3_cover:cover property(XOR_RedOp_3);
XOR4_cover:cover property(XOR_RedOp_4);
XOR5_cover:cover property(XOR_check);
FullADD_cover:cover property(fullAdd);
Mult_cover:cover property(multiply);
shiftL_cover:cover property(shift_left);
shiftR_cover:cover property(shift_right);
rotateLeft_cover:cover property(rotate_left);
rotateRight_cover:cover property(rotate_right);
endmodule
```

PACKAGE:

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

endpackage
```

SHIFT MONITOR:

```
package shift_monitor_pkg;

import uvm_pkg::*;
`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 alsu_config_vif;
        seq_item itm;
        uvm_analysis_port #(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=seq_item::type_id::create("items recived");
                #6;
                itm.serial_in=alsu_config_vif.serial_in;
                itm.direction=alsu_config_vif.direction;
                itm.mode=alsu_config_vif.mode;
                itm.datain=alsu_config_vif.datain;
                mon_p.write(itm);
                 `uvm_info("run_phase",itm.convert2string(),UVM_HIGH);
            end
        endtask
    endclass
endpackage
```

SHIFT REGISTER:

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


always @(*) begin

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

SHIFT AGENT:

```
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 #(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);
            end
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            mon.mon_p.connect(agt_p);
            mon.alsu_config_vif=cfg.shift_config_sif;
            if(is_active==UVM_ACTIVE)begin
                drv.seq_item_port.connect(sqr.seq_item_export);
                drv.alsu_config_vif=cfg.shift_config_sif;
            end
        endfunction
    endclass
endpackage
```

SHIFT SEQ ITEM:

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

    class seq_item extends uvm_sequence_item ;
        `uvm_object_utils(seq_item);

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

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

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

        function string convert2string_stimulus();
            return $sformatf("%s  reset =%0b serial_in=%0b ,direction=%0b ,mode=%0b ,datain=%0b  ", super.convert2string(),reset,serial_in,direction,mode,datain);
        endfunction

        constraint x{
            reset dist {1:=95 ,0 :=5};
        }


    endclass
endpackage
```

CONFIG:

```
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 shift_config_sif;
        virtual ALSU_if alsu_config_vif;
        uvm_active_passive_enum is_active;

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


    endclass //ALSU_if


endpackage
```

SHIFT ENVIROMENT:

```
1
2
3   package shift_env_pkg;
4   import uvm_pkg::*;
5   `include "uvm_macros.svh"
6       import shift_scoreboard_pkg::*;
7       import shift_coverage_pkg::*;
8       import shift_agent_pkg::*;
9
10      class shit_reg_env extends uvm_component ;
11          `uvm_component_utils(shit_reg_env);
12          shift_reg_sb sb;
13          shift_reg_cov cov;
14          shift_reg_agent agent;
15
16          function new(string name = "shift_reg_env", uvm_component parent =null);
17              super.new(name,parent);
18          endfunction
19
20          function void build_phase(uvm_phase phase);
21              super.build_phase(phase);
22              sb=shift_reg_sb::type_id::create("scoreboard",this);
23              cov=shift_reg_cov::type_id::create("coverage collector",this);
24              agent=shift_reg_agent::type_id::create("Agent",this);
25          endfunction
26
27          function void connect_phase(uvm_phase phase);
28              super.connect_phase(phase);
29              agent.agt_p.connect(sb.sb_export);
30              agent.agt_p.connect(cov.cov_export);
31          endfunction
32      endclass
33  endpackage
```

SHIFT SEQUENCER:

```
1   package  shift_sequencer_pkg;
2
3   import uvm_pkg::*;
4   `include "uvm_macros.svh"
5       import shift_seq_item_pkg::*;
6       class shift_reg_sequencer extends uvm_sequencer #(seq_item);
7           `uvm_component_utils(shift_reg_sequencer);
8
9           function new(string name="sequencer",uvm_component phase);
10              super.new(name,phase);
11          endfunction
12      endclass
13
14
15  endpackage
```

SHIFT INTERFACE:

```
1  interface shift_reg_if ();
2
3    logic serial_in, direction, mode;
4    logic [5:0] datain, dataout;
5  endinterface : shift_reg_if
```

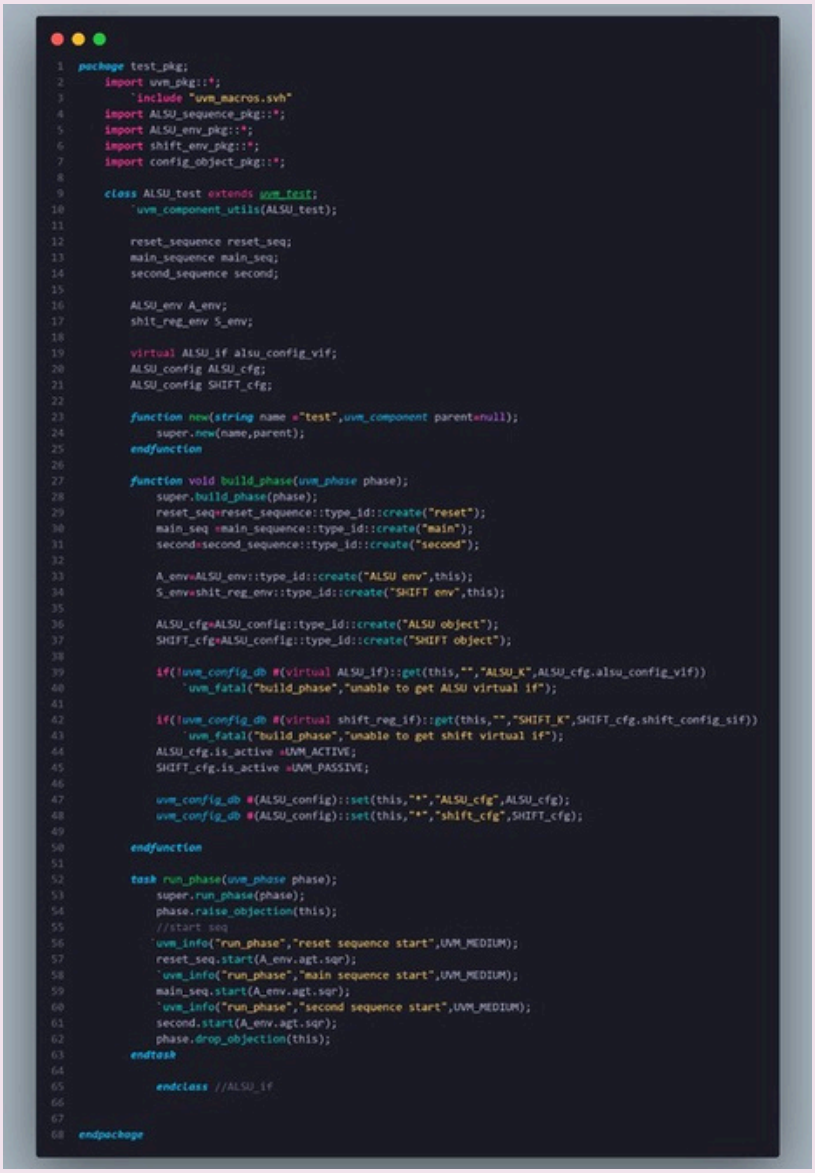
SHIFT SEQUENCE:

```
1
2
3  package shift_sequence_pkg;
4  import uvm_pkg::*;
5  `include "uvm_macros.svh"
6  import shift_seq_item_pkg::*;
7      class shit_reg_sequence extends uvm_sequence #(seq_item);
8          `uvm_object_utils(shit_reg_sequence);
9          seq_item sq;
10         function new(string name = "shift_reg_sequence");
11             super.new(name);
12         endfunction
13
14         task body();
15             repeat(100)begin
16                 sq=seq_item::type_id::create("sequence")   ;
17                 start_item(sq);
18                 assert(sq.randomize());
19                 finish_item(sq);
20
21             end
22
23         endtask
24
25
26
27     endclass
28 endpackage
```

TOP:

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

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

ALSU_if alif(clk);
shift_reg_if sh_if();
shift_reg SR(sh_if);
ALSU DUT (alif,sh_if);

bind ALSU Asseritions AS(alif);

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

endmodule
```

TEST:



```
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::*;

    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 alsu_config_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);

            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.alsu_config_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.shift_config_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
```

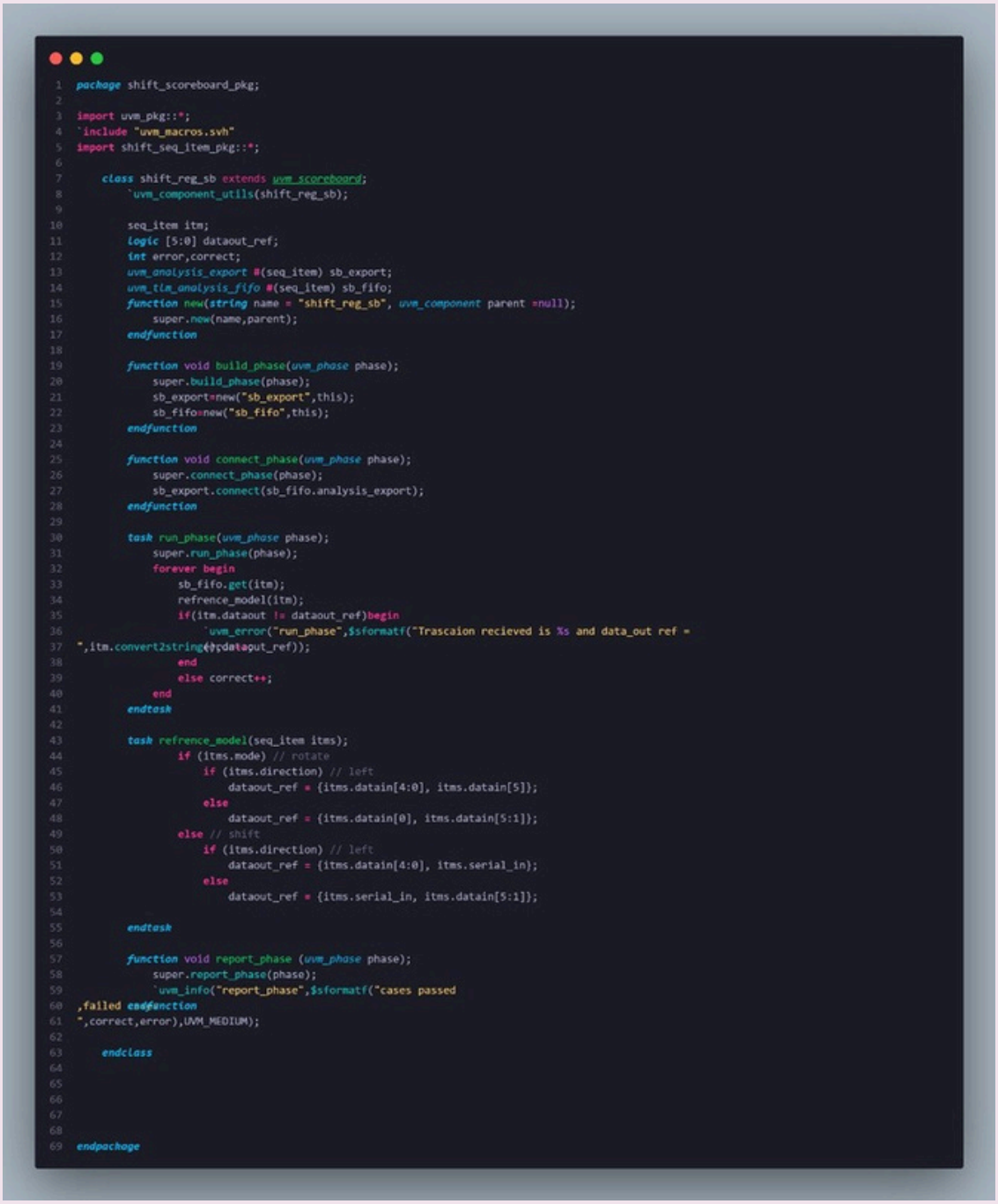
SHIFT_DRIVER:

```
package shift_driver_pkg;

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

    class shift_reg_driver extends uvm_driver #(seq_item);
        `uvm_component_utils(shift_reg_driver);
        virtual shift_reg_if alsu_config_vif;
        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=seq_item::type_id::create("item_send");
                seq_item_port.get_next_item(itm);
                alsu_config_vif.serial_in=itm.serial_in;
                alsu_config_vif.direction=itm.direction;
                alsu_config_vif.mode=itm.mode;
                alsu_config_vif.datain=itm.datain;
                #2;

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

            end
        endtask
    endclass
endpackage
```

SHIFT_COVERAGE:

```
package shift_coverage_pkg;

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

    class shift_reg_cov extends uvm_component;
        `uvm_component_utils(shift_reg_cov);

        seq_item itm;
        uvm_analysis_export #(seq_item) cov_export;
        uvm_tlm_analysis_fifo #(seq_item) cov_fifo;

        //covergorup
        covergroup covgup;
            data_in:coverpoint itm.datain  ;
            MODE:coverpoint itm.mode {
                bins mode0={0};
                bins mode1={1};
            }
            direction:coverpoint itm.direction{
                bins direction0={0};
                bins direction1={1};
            }
        endgroup

        function new(string name = "shift_reg_cov", uvm_component parent =null);
            super.new(name,parent);
            covgup=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

        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(itm);
                `uvm_info("run_phase",itm.convert2string(),UVM_MEDIUM);
                covgup.sample();
            end
        endtask

    endclass
endpackage
```

SHIFT_SCOREBOARD:



```
package shift_scoreboard_pkg;

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

    class shift_reg_sb extends uvm_scoreboard;
        `uvm_component_utils(shift_reg_sb);

        seq_item itm;
        logic [5:0] dataout_ref;
        int error,correct;
        uvm_analysis_export #(seq_item) sb_export;
        uvm_tlm_analysis_fifo #(seq_item) sb_fifo;
        function new(string name = "shift_reg_sb", 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);
            super.connect_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(itm);
                refrence_model(itm);
                if(itm.dataout != dataout_ref)begin
                    `uvm_error("run_phase",$sformatf("Trascaion recieved is %s and data_out ref =
",itm.convert2string(),dataout_ref));
                end
                else correct++;
            end
        endtask

        task refrence_model(seq_item itms);
                if (itms.mode) // rotate
                    if (itms.direction) // left
                        dataout_ref = {itms.datain[4:0], itms.datain[5]};
                    else
                        dataout_ref = {itms.datain[0], itms.datain[5:1]};
                else // shift
                    if (itms.direction) // left
                        dataout_ref = {itms.datain[4:0], itms.serial_in};
                    else
                        dataout_ref = {itms.serial_in, itms.datain[5:1]};

        endtask

        function void report_phase (uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase",$sformatf("cases passed
,failed endfunction
",correct,error),UVM_MEDIUM);

    endclass




endpackage
```

PRINT:

```
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :83343
# UVM_WARNING :    1
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [TPRGED]     1
# [report_phase]     1
# [run_phase] 83338
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 500010 ns  Iteration: 61  Instance: /top
# 1
```

COVERAGE:

```
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
------------------------------------------------------------------------------------------
/top/DUT/AS/leds_cover                  Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(218)
                                                                          8204 Covered
/top/DUT/AS/bypass1_cover               Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(222)
                                                                           166 Covered
/top/DUT/AS/bypass3_cover               Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(224)
                                                                          1468 Covered
/top/DUT/AS/bypass4_cover               Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(225)
                                                                          1503 Covered
/top/DUT/AS/INVALID_cover               Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(228)
                                                                          6712 Covered
/top/DUT/AS/OR1_cover                   Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(231)
                                                                           344 Covered
/top/DUT/AS/OR3_cover                   Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(233)
                                                                           214 Covered
/top/DUT/AS/OR4_cover                   Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(234)
                                                                           263 Covered
/top/DUT/AS/OR5_cover                   Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(235)
                                                                          6710 Covered
/top/DUT/AS/XOR1_cover                  Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(238)
                                                                           359 Covered
/top/DUT/AS/XOR3_cover                  Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(239)
                                                                           231 Covered
/top/DUT/AS/XOR4_cover                  Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(240)
                                                                           270 Covered
/top/DUT/AS/XOR5_cover                  Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(241)
                                                                          6642 Covered
/top/DUT/AS/FullADD_cover               Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(244)
                                                                    1430148870 Covered
/top/DUT/AS/Mult_cover                  Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(247)
                                                                          5448 Covered
/top/DUT/AS/shiftL_cover                Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(250)
                                                                          2822 Covered
/top/DUT/AS/shiftR_cover                Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(251)
                                                                          2805 Covered
/top/DUT/AS/rotateLeft_cover            Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(254)
                                                                          2827 Covered
/top/DUT/AS/rotateRight_cover           Asseritions Verilog  SVA  D:/Verification course/extraaa6/ASSERTT.svh(255)
                                                                          2781 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 19

Total Coverage By Instance (filtered view): 100.00%
```

ASSERTIONS:

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included
/top/DUT/AS/leds_cover	SVA	✓	Off	8204	1	Unli...	1	100%		✓
/top/DUT/AS/bypass1_cover	SVA	✓	Off	166	1	Unli...	1	100%		✓
/top/DUT/AS/bypass3_cover	SVA	✓	Off	1468	1	Unli...	1	100%		✓
/top/DUT/AS/bypass4_cover	SVA	✓	Off	1503	1	Unli...	1	100%		✓
/top/DUT/AS/INVALID_cover	SVA	✓	Off	6712	1	Unli...	1	100%		✓
/top/DUT/AS/OR1_cover	SVA	✓	Off	344	1	Unli...	1	100%		✓
/top/DUT/AS/OR3_cover	SVA	✓	Off	214	1	Unli...	1	100%		✓
/top/DUT/AS/OR4_cover	SVA	✓	Off	263	1	Unli...	1	100%		✓
/top/DUT/AS/OR5_cover	SVA	✓	Off	6710	1	Unli...	1	100%		✓
/top/DUT/AS/XOR1_cover	SVA	✓	Off	359	1	Unli...	1	100%		✓
/top/DUT/AS/XOR3_cover	SVA	✓	Off	231	1	Unli...	1	100%		✓
/top/DUT/AS/XOR4_cover	SVA	✓	Off	270	1	Unli...	1	100%		✓
/top/DUT/AS/XOR5_cover	SVA	✓	Off	6642	1	Unli...	1	100%		✓
/top/DUT/AS/FullADD_cover	SVA	✓	Off	5494	1	Unli...	1	100%		✓
/top/DUT/AS/Mult_cover	SVA	✓	Off	5448	1	Unli...	1	100%		✓
/top/DUT/AS/shiftL_cover	SVA	✓	Off	2822	1	Unli...	1	100%		✓
/top/DUT/AS/shiftR_cover	SVA	✓	Off	2805	1	Unli...	1	100%		✓
/top/DUT/AS/rotateLeft_cover	SVA	✓	Off	2826	1	Unli...	1	100%		✓
/top/DUT/AS/rotateRight_cover	SVA	✓	Off	2781	1	Unli...	1	100%		✓

WAVE: