

MARIAM MOHAMED MARZOUK

ASSIGNMENT_6_Extended

ALSU:

```
1 module ALSU(ALSU_if alif,shift_reg_if sh_if);
2 reg red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
3 reg signed [1:0] cin_reg;
4 reg [2:0] opcode_reg;
5 reg signed [2:0] A_reg, B_reg;
6 reg signed [5:0] out_next;
7 wire invalid_red_op, invalid_opcode, invalid;
8 reg signed [5:0]out_shift_reg;
9 assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
10 assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
11 assign invalid = invalid_red_op | invalid_opcode;
12 assign sh_if.serial_in=serial_in_reg;
13 assign sh_if.direction=direction_reg;
14 assign sh_if.nodes=opcode_reg[0];
15 assign sh_if.datain=alif.out;
16 assign out_shift_reg=sh_if.dataout;
17 always @ (posedge alif.clk or posedge alif.reset) begin
18   if(alif.reset) begin
19     cin_reg <= 0;
20     red_op_B_reg <= 0; red_op_A_reg <= 0;
21     bypass_B_reg <= 0; bypass_A_Reg <= 0;
22     direction_reg <= 0; serial_in_reg <= 0;
23     opcode_reg <= 0; A_reg <= 0; B_reg <= 0;
24   end else begin
25     cin_reg <= alif.cin;
26     red_op_B_reg <= alif.red_op_B;red_op_A_Reg <= alif.red_op_A;
27     bypass_B_reg <= alif.bypass_B;bypass_A_Reg <= alif.bypass_A;
28     direction_reg <= alif.direction;serial_in_Reg <= alif.serial_in;
29     opcode_Reg <= alif.opcode;A_Reg <= alif.A;B_Reg <= alif.B;
30   end end
31 always @ (posedge alif.clk or posedge alif.reset) begin
32   if(alif.reset) begin
33     alif.leds <= 0;
34   end else begin
35     if (invalid)
36       alif.leds <= ~alif.leds; else alif.leds <= 0;
37   end end
38 always @ (posedge alif.clk or posedge alif.reset) begin
39   if(alif.reset) begin
40     alif.out <= 0;
41   end else begin
42     if (bypass_A_Reg && bypass_B_Reg)
43       alif.out <= (alif.INPUT_PRIORITY == "A")?A_Reg: B_Reg;
44     else if (bypass_A_Reg)
45       alif.out <= A_Reg;
46     else if (bypass_B_Reg)
47       alif.out <= B_Reg;
48     else if (invalid) // change the priority of invalid bits after bypass_Reg
49       alif.out <= 0;
50     else begin
51       case (opcode_Reg)
52         3'h0: begin //change Opcode to OR not AND
53           if (red_op_A_Reg && red_op_B_Reg)
54             alif.out <= (alif.INPUT_PRIORITY == "A")? ~A_Reg: ~B_Reg;
55           else if (red_op_A_Reg)
56             alif.out <= ~A_Reg;
57           else if (red_op_B_Reg)
58             alif.out <= ~B_Reg;
59           else
60             alif.out <= A_Reg | B_Reg;
61         end
62         3'h1: begin // change opcode to XOR not OR
63           if (red_op_A_Reg && red_op_B_Reg)
64             alif.out <= (alif.INPUT_PRIORITY == "A")? ~A_Reg: ~B_Reg;
65           else if (red_op_A_Reg)
66             alif.out <= ~A_Reg;
67           else if (red_op_B_Reg)
68             alif.out <= ~B_Reg;
69           else
70             alif.out <= A_Reg ^ B_Reg;
71         end
72         3'h2: begin //here we add condition to check full adder if ON or OFF
73           if(alif.FULL_ADDER == "ON")
74             alif.out <= A_Reg + B_Reg+cin_Reg;
75           else if(alif.FULL_ADDER == "OFF")
76             alif.out <= A_Reg + B_Reg;
77           end
78           3'h3: alif.out <= A_Reg * B_Reg;
79           3'h4: begin //100
80             alif.out <= out_shift_Reg;
81           end
82           3'h5: begin//101
83             alif.out <= out_shift_Reg;
84           end
85           default: alif.out<=alif.out;
86         endcase
87       end end
88     out_next<=alif.out;
89   end
90 endmodule
```

ALSU INTERFACE:

```
● ● ●  
1 interface ALSU_if(clk);  
2  
3   input bit clk;  
4  
5   parameter INPUT_PRIORITY = "A";  
6   parameter FULL_ADDER = "ON";  
7   Logic reset, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;  
8   Logic signed cin;  
9   Logic [2:0] opcode;  
10  Logic signed [2:0] A, B;  
11  Logic [15:0] leds;  
12  Logic signed[5:0] out;  
13  
14 endinterface
```

ALSU DRIVER:

```
1 package ALSU_driver_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import ALSU_seq_item_pkg::*;
5 class ALSU_driver extends uvm_driver #(seq_item);
6     `uvm_component_utils(ALSU_driver);
7     virtual ALSU_if alsu_config_vif;
8     seq_item item;
9     function new(string name ="driver",uvm_component parent=null);
10         super.new(name,parent);
11     endfunction
12
13     task run_phase(uvm_phase phase);
14         super.run_phase(phase);
15         forever begin
16             item=seq_item::type_id::create("seq_item");
17             seq_item_port.get_next_item(item);
18             alsu_config_vif.reset=item.reset;
19             alsu_config_vif.A=item.A;
20             alsu_config_vif.B=item.B;
21             alsu_config_vif.cin=item.cin;
22             alsu_config_vif.red_op_A=item.red_op_A;
23             alsu_config_vif.red_op_B=item.red_op_B;
24             alsu_config_vif.bypass_A=item.bypass_A;
25             alsu_config_vif.bypass_B=item.bypass_B;
26             alsu_config_vif.direction=item.direction;
27             alsu_config_vif.serial_in=item.serial_in;
28             alsu_config_vif.opcode=item.opcode;
29             @(negedge alsu_config_vif.clk);
30             seq_item_port.item_done();
31         end
32     endtask
33 endclass //ALSU_if
34 endpackage
```

ALSU ENVIRONMENT:

```
● ○ ●

1 package ALSU_env_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import ALSU_coverage_pkg::*;
5   import ALSU_agent_pkg::*;
6
7   class ALSU_env extends uvm_env;
8     `uvm_component_utils(ALSU_env);
9     ALSU_coverage cov;
10    ALSU_agent agt;
11
12    function new(string name ="env",uvm_component parent=null);
13      super.new(name,parent);
14    endfunction
15
16    function void build_phase(uvm_phase phase);
17      super.build_phase(phase);
18      cov=ALSU_coverage::type_id::create("coverage",this);
19      agt=ALSU_agent::type_id::create("agent",this);
20    endfunction
21
22    function void connect_phase(uvm_phase phase);
23      super.connect_phase(phase);
24      agt.agt_p.connect(cov.cov_ep);
25    endfunction
26
27  endclass //ALSU_if
28
29
30 endpackage
```

ALSU MONITOR:



```
1 package ALSU_monitor_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import ALSU_seq_item_pkg::*;
5
6   class ALSU_mointor extends uvm_monitor ;
7     `uvm_component_utils(ALSU_mointor);
8     virtual ALSU_if alsu_config_vif;
9     seq_item item;
10    uvm_analysis_port #(seq_item) mon_p;
11    function new(string name = "", uvm_component parent=null);
12      super.new(name,parent);
13    endfunction
14    function void build_phase(uvm_phase phase);
15      super.build_phase(phase);
16      mon_p=new("alsu monitor port",this);
17    endfunction
18    task run_phase(uvm_phase phase);
19      super.run_phase(phase);
20      forever begin
21        item=seq_item::type_id::create("items received");
22        @(negedge alsu_config_vif.clk);
23        item.reset=alsu_config_vif.reset;
24        item.A=alsu_config_vif.A;
25        item.B=alsu_config_vif.B;
26        item.cin=alsu_config_vif.cin;
27        item.red_op_A=alsu_config_vif.red_op_A;
28        item.red_op_B=alsu_config_vif.red_op_B;
29        item.bypass_A=alsu_config_vif.bypass_A;
30        item.bypass_B=alsu_config_vif.bypass_B;
31        item.direction=alsu_config_vif.direction;
32        item.serial_in=alsu_config_vif.serial_in;
33        item.opcode=alsu_config_vif.opcode;
34        mon_p.write(item);
35        `uvm_info("run_phase",item.convert2string(),UVM_HIGH);
36      end
37    endtask
38  endclass //ALSU_if
39 endpackage
```

ALSU AGENT:

```
1 package ALSU_agent_pkg;
2     import uvm_pkg::*;
3         `include "uvm_macros.svh"
4     import ALSU_monitor_pkg::*;
5     import ALSU_sequencer_pkg::*;
6     import ALSU_driver_pkg::*;
7     import ALSU_seq_item_pkg::*;
8     import config_object_pkg::*;

9
10    class ALSU_agent extends uvm_agent;
11        `uvm_component_utils(ALSU_agent);
12        ALSU_mointor mon;
13        ALSU_sequencer sqr;
14        ALSU_driver driver;
15        ALSU_config cfg;
16        uvm_analysis_port #(seq_item) agt_p;
17        uvm_active_passive_enum is_active;

18
19
20        function new(string name ="agent",uvm_component parent=null);
21            super.new(name,parent);
22        endfunction
23
24        function void build_phase(uvm_phase phase);
25            super.build_phase(phase);

26            mon=ALSU_mointor::type_id::create("monitor",this);
27            agt_p=new("agent port",this);
28            if(!uvm_config_db #(ALSU_config)::get(this,"","ALSU_cfg",cfg))
29                `uvm_fatal("build_phase","cant get ALSU virtual interface");
30            is_active=cfg.is_active;
31            if(is_active==UVM_ACTIVE)begin
32                sqr=ALSU_sequencer::type_id::create("sequencer",this);
33                driver=ALSU_driver::type_id::create("driver",this);
34            end
35        endfunction
36
37        function void connect_phase(uvm_phase phase);
38            super.connect_phase(phase);
39            if(is_active==UVM_ACTIVE)begin
40                driver.alsu_config_vif=cfg.alsu_config_vif;
41                driver.seq_item_port.connect(sqr.seq_item_export);
42            end
43            mon.alsu_config_vif=cfg.alsu_config_vif;
44            mon.mon_p.connect(agt_p);
45        endfunction
46
47
48
49    endclass //ALSU_if
50
51
52 endpackage
```

ALSU SEQUENCE:

```
1 package ALSU_sequence_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import ALSU_seq_item_pkg::*;
5   class reset_sequence extends uvm_sequence #(seq_item);
6     `uvm_object_utils(reset_sequence);
7     function new(string name ="reset_sequence");
8       super.new(name);
9     endfunction
10    task body();
11      seq_item item;
12      item=seq_item::type_id::create("sequence reset");
13      start_item(item);
14      item.reset=1;
15      finish_item(item);
16    endtask
17  endclass //ALSU_if
18 class main_sequence extends uvm_sequence #(seq_item);
19   `uvm_object_utils(main_sequence);
20   function new(string name ="main_sequence",uvm_component parent=null);
21     super.new(name);
22   endfunction
23   task body();
24     repeat(20000)begin
25       seq_item item;
26       item=seq_item::type_id::create("sequence");
27       item.constraint_mode(0);
28       item.x.constraint_mode(1);
29       start_item(item);
30       assert(item.randomize());
31       finish_item(item);
32     end
33   endtask
34 endclass //ALSU_if
35 class second_sequence extends uvm_sequence #(seq_item);
36   `uvm_object_utils(second_sequence);
37   function new(string name ="second_sequence",uvm_component parent=null);
38     super.new(name);
39   endfunction
40   task body();
41     for(int i=0;i<5000;i++)begin
42       seq_item item;
43       item=seq_item::type_id::create("sequence");
44       item.constraint_mode(0);
45       item.y.constraint_mode(1);
46       //start_item(item);
47       item.reset=0;item.bypass_A=0;item.bypass_B=0;item.red_op_A=0;item.red_op_B=0;
48       item.reset.rand_mode(0);item.bypass_A.rand_mode(0);item.bypass_B.rand_mode(0);item.red_op_A.rand_mode(0);
49       item.red_op_B.rand_mode(0);
50       assert(item.randomize());
51       //if('{'OR,XOR,ADD,MULT,SHIFT,ROTATE} ==item.arr)$display("@%0t the wanted sequence is %p",$time,tr.arr);
52       foreach(item.arr[j])begin
53         start_item(item);
54         item.opcode=item.arr[j];
55         finish_item(item);
56       end
57     end
58   endtask
59 endclass //ALSU_if
60 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:

```

1 package ALSU_seq_item_pkg;
2   import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     import constantEnums::*;
5   class seq_item extends uvm_sequence_item;
6     `uvm_object_utils(seq_item);
7     rand bit clk, reset, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
8     rand logic cin;
9     rand logic [2:0] opcode;
10    rand bit signed [2:0] A, B;
11    bit signed [5:0] out;
12    bit [15:0] leds;
13    bit [2:0] ones_number={3'b001,3'b010,3'b100};
14    rand bit [2:0] found,notfound;
15    rand cornerState_e a_state;
16    rand bit [2:0] rem_numbers;
17    rand opcode_e arr[6];
18    function new(string name = "seq_item");
19      super.new(name);
20    endfunction
21    function string convert2string();
22      return $sformatf("%s reset=%0b cin=%0b ,red_op_A=%0b ,red_op_B=%0b ,bypass_A=%0b ,bypass_B=%0b ,
23      direction=%0b ,serial_in=%0b ,opcode=%0b ,A=%0b,B=%0b ,out=%0b,leds=%0b ",
24      super.convert2string(),reset,cin,red_op_A,red_op_B,bypass_A,bypass_B,direction,serial_in,opcode,A,B,out,leds);
25    endfunction
26
27    function string convert2string_stimulus();
28      return $sformatf("%s reset=%0b cin=%0b ,red_op_A=%0b ,red_op_B=%0b ,bypass_A=%0b ,bypass_B=%0b ,
29      direction=%0b ,serial_in=%0b ,opcode=%0b ,A=%0b,B=%0b ",
30      super.convert2string(),reset,cin,red_op_A,red_op_B,bypass_A,bypass_B,direction,serial_in,opcode,A,B);
31    endfunction
32    constraint x {
33      rem_numbers != MAXPOS || MAXNEG || ZERO;
34      reset dist {1:#5, 0:#95};
35
36      found inside {ones_number};
37      !notfound inside {ones_number};
}

```

```

9      if (opcode == ADD || opcode == MULT){
10        A dist {a_state:=80,rem_numbers:=20};
11        B dist {a_state:=80,rem_numbers:=20};
12      }
13      if (opcode == OR || opcode == XOR ){
14        if(red_op_A){
15          A dist {found:=80,notfound:=20};
16          B==3'b000;
17        }
18        else if (red_op_B){
19          B dist {found:=80,notfound:=20};
20          A==3'b000;
21        }
22      opcode dist {[OR:ROTATE]:=80,[INVALID6:INVALID7]};
23
24      bypass_A dist {0:=90,1:=10};
25      bypass_B dist {0:=90,1:=10;};
26    constraint y{
27      unique{arr};
28      foreach(arr[i])
29        arr[i] inside {[OR:ROTATE]};
30    }
31  endclass //ALSU_if
32 endpackage

```

ALSU COVERAGE:



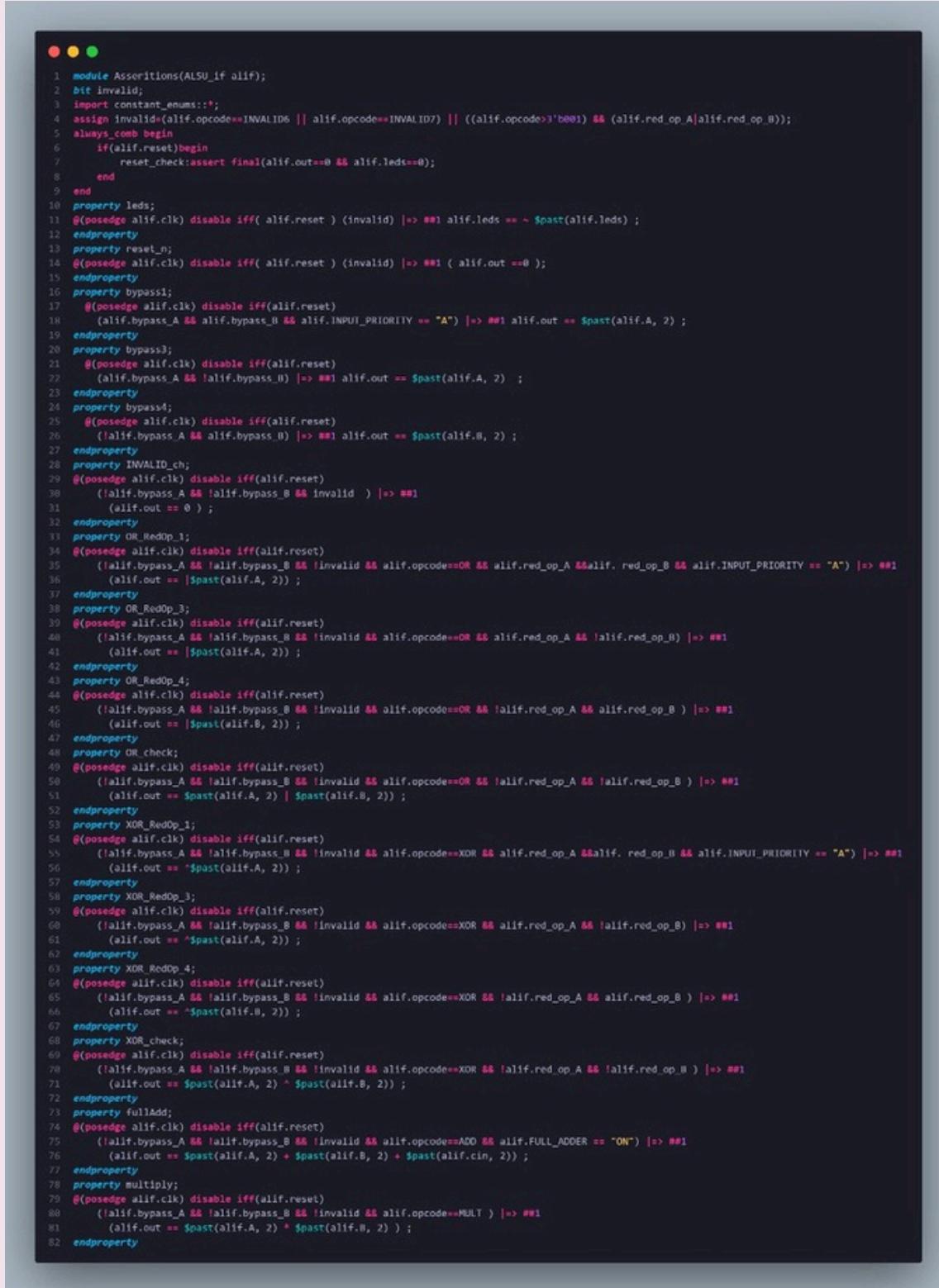
```
1 package ALSU_coverage_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import ALSU_seq_item_pkg::*;
5   import constantEnums::*;
6   class ALSU_coverage extends uvm_component;
7     `uvm_component_utils(ALSU_coverage);
8     seq_item item;
9     uvm_analysis_export #(seq_item) cov_ep;
10    uvm_tlm_analysis_fifo #(seq_item) cov_fifo;
11    covergroup cvr_gp;
12      CIN:coverpoint item.CIN{
13        bins CIN0={0};
14        bins CIN1={1};
15        option.weight=0;
16      }
17      red_op_A:coverpoint item.red_op_A{
18        bins red_op_A0={0};
19        bins red_op_A1={1};
20        option.weight=0;
21      }
22      red_op_B:coverpoint item.red_op_B{
23        bins red_op_B0={0};
24        bins red_op_B1={1};
25        option.weight=0;
26      }
27      bypass_A:coverpoint item.bypass_A{
28        bins bypass_A0={0};
29        bins bypass_A1={1};
30        option.weight=0;
31      }
32      bypass_B:coverpoint item.bypass_B{
33        bins bypass_B0={0};
34        bins bypass_B1={1};
35        option.weight=0;
36      }
37      direction:coverpoint item.direction{
38        bins direction0={0};
39        bins direction1={1};
40        option.weight=0;
41      }
42      serial_in:coverpoint item.serial_in{
43        bins serial_in0={0};
44        bins serial_in1={1};
45        option.weight=0;
46      }
47      special:coverpoint item.opcode{
48        option.weight=0;
49        bins operations[] = {[OR:ROTATE]};
50      }
51      C01:coverpoint item.A{
52        bins A_data_0={0};
53        bins A_data_max={MAXPOS};
54        bins A_data_min={MAXNEG};
55        bins A_data_walkingones[] ={3'b001,3'b010,3'b100} iff (item.red_op_A);
56        bins A_data_default=default;
57      }
58      C02:coverpoint item.B{
59        bins B_data_0={0};
60        bins B_data_max={MAXPOS};
61        bins B_data_min={MAXNEG};
62        bins B_data_walkingones[] ={3'b001,3'b010,3'b100} iff (item.red_op_B);
63        bins B_data_default=default;
64      }
65      A_M:coverpoint item.opcode{
66        bins Bins_arith[] ={ADD,MULT};
67      }
```

```

1 CB3:coverpoint item.opcode{
2     bins Bins_Shift[] ={SHIFT,ROTATE};
3     bins Bins_Arith[] ={ADD,MULT};
4     bins Bins_Bitwise[] ={OR,XOR};
5     illegal_bins Bins_Invalid ={INVALID6,INVALID7};
6     bins Bins_Trans={OR=>XOR=>ADD=>MULT=>SHIFT=>ROTATE};
7 }
8 //1
9 corner_case:cross A_M,CB2,CB1{
10     ignore_bins walkingA=binsof(CB1.A_data_walkingones) ;
11     ignore_bins walkingB=binsof(CB2.B_data_walkingones) ;
12 }
13 //2
14 Addation:cross CIN,special{
15     option.cross_auto_bin_max=0;
16     bins Add_cin0=binsof(special) intersect {ADD} && binsof(CIN.CIN0) ;
17     bins Add_cin1=binsof(special) intersect {ADD} && binsof(CIN.CIN1) ;
18 }
19 //3
20 shift:cross special,serial_in{
21     option.cross_auto_bin_max=0;
22     bins shift_Si0=binsof(special) intersect {SHIFT} && binsof(serial_in.serial_in0) ;
23     bins shift_Si1=binsof(special) intersect {SHIFT} && binsof(serial_in.serial_in0) ;
24 }
25 //4
26 shift_rotate:cross CB3,direction{
27     option.cross_auto_bin_max=0;
28     bins shu_rot_d0=binsof(CB3.Bins_Shift) && binsof(direction.direction0) ;
29     bins shu_rot_d1=binsof(CB3.Bins_Shift) && binsof(direction.direction1) ;
30 }
31 //5
32 walkingones:cross CB3,CB2,CB1{
33     option.cross_auto_bin_max=0;
34     bins arithA=binsof(CB3.Bins_Bitwise) && binsof(CB2.B_data_0) && binsof(CB1.A_data_walkingones) ;
35     bins arithB=binsof(CB3.Bins_Bitwise) && binsof(CB1.A_data_0) && binsof(CB2.B_data_walkingones) ;
36 }
37 invalidation:cross red_op_A,red_op_B,special{
38     option.cross_auto_bin_max=0;
39     bins ROpA_notXor=binsof(special) intersect{[ADD:ROTATE]} && binsof(red_op_A.red_op_A1) ;
40     bins ROpB_notXor=binsof(special) intersect{[ADD:ROTATE]} && binsof(red_op_B.red_op_B1) ;
41 }
42 endgroup
43 function new(string name ="coverage",uvm_component parent=null);
44     super.new(name,parent);
45     cvr_gp=new();
46 endfunction
47 function void build_phase(uvm_phase phase);
48     super.build_phase(phase);
49     cov_fifo=new("fifo",this);
50     cov_ep=new("cov_export",this);
51 endfunction
52 function void connect_phase(uvm_phase phase);
53     super.connect_phase(phase);
54     cov_ep.connect(cov_fifo.analysis_export);
55 endfunction
56 task run_phase(uvm_phase phase);
57     super.run_phase(phase);
58     forever begin
59         cov_fifo.get(item);
60         cvr_gp.sample();
61     end
62 endtask
63 endclass //ALSU_if
64 endpackage

```

ASSERTIONS:



```
1 module Assertions(ALSU_IF alif);
2   bit invalid;
3   import constant_enums::*;
4   assign invalid=alif.opcode==INVALID6 || alif.opcode==INVALID7 || ((alif.opcode>3'b001) && (alif.red_op_A|alif.red_op_B));
5   always_comb begin
6     if(alif.reset)begin
7       reset_check:assert final(alif.out==0 && alif.leds==0);
8     end
9   end
10  property leds;
11    @(posedge alif.clk) disable iff( alif.reset ) (invalid ) |>> ##1 alif.leds == ~ $past(alif.leds) ;
12  endproperty
13  property reset_n;
14    @(posedge alif.clk) disable iff( alif.reset ) (invalid ) |>> ##1 ( alif.out ==0 );
15  endproperty
16  property bypass1;
17    @(posedge alif.clk) disable iff(alif.reset)
18      (alif.bypass_A && alif.bypass_B && alif.INPUT_PRIORITY == "A") |>> ##1 alif.out == $past(alif.A, 2) ;
19  endproperty
20  property bypass2;
21    @(posedge alif.clk) disable iff(alif.reset)
22      (alif.bypass_A && !alif.bypass_B) |>> ##1 alif.out == $past(alif.A, 2) ;
23  endproperty
24  property bypass3;
25    @(posedge alif.clk) disable iff(alif.reset)
26      (!alif.bypass_A && alif.bypass_B) |>> ##1 alif.out == $past(alif.B, 2) ;
27  endproperty
28  property INVALID_ch;
29    @(posedge alif.clk) disable iff(alif.reset)
30      (!alif.bypass_A && !alif.bypass_B && invalid ) |>> ##1
31      (alif.out == 0 ) ;
32  endproperty
33  property OR_RedOp_1;
34    @(posedge alif.clk) disable iff(alif.reset)
35      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==OR && alif.red_op_A && alif.red_op_B && alif.INPUT_PRIORITY == "A") |>> ##1
36      (alif.out == $past(alif.A, 2)) ;
37  endproperty
38  property OR_RedOp_2;
39    @(posedge alif.clk) disable iff(alif.reset)
40      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==OR && alif.red_op_A && !alif.red_op_B) |>> ##1
41      (alif.out == $past(alif.A, 2)) ;
42  endproperty
43  property OR_RedOp_3;
44    @(posedge alif.clk) disable iff(alif.reset)
45      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==OR && !alif.red_op_A && alif.red_op_B ) |>> ##1
46      (alif.out == $past(alif.B, 2)) ;
47  endproperty
48  property OR_check;
49    @(posedge alif.clk) disable iff(alif.reset)
50      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==OR && !alif.red_op_A && !alif.red_op_B ) |>> ##1
51      (alif.out == $past(alif.A, 2) | $past(alif.B, 2)) ;
52  endproperty
53  property XOR_RedOp_1;
54    @(posedge alif.clk) disable iff(alif.reset)
55      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==XOR && alif.red_op_A && alif.red_op_B && alif.INPUT_PRIORITY == "A") |>> ##1
56      (alif.out == $past(alif.A, 2)) ;
57  endproperty
58  property XOR_RedOp_3;
59    @(posedge alif.clk) disable iff(alif.reset)
60      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==XOR && alif.red_op_A && !alif.red_op_B ) |>> ##1
61      (alif.out == $past(alif.A, 2)) ;
62  endproperty
63  property XOR_RedOp_4;
64    @(posedge alif.clk) disable iff(alif.reset)
65      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==XOR && !alif.red_op_A && alif.red_op_B ) |>> ##1
66      (alif.out == $past(alif.B, 2)) ;
67  endproperty
68  property XOR_check;
69    @(posedge alif.clk) disable iff(alif.reset)
70      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==XOR && !alif.red_op_A && !alif.red_op_B ) |>> ##1
71      (alif.out == $past(alif.A, 2) ^ $past(alif.B, 2)) ;
72  endproperty
73  property fullAdd;
74    @(posedge alif.clk) disable iff(alif.reset)
75      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==ADD && alif.FULL_ADDER == "ON") |>> ##1
76      (alif.out == $past(alif.A, 2) + $past(alif.B, 2) + $past(alif.cin, 2)) ;
77  endproperty
78  property multiply;
79    @(posedge alif.clk) disable iff(alif.reset)
80      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==MULT ) |>> ##1
81      (alif.out == $past(alif.A, 2) * $past(alif.B, 2) ) ;
82  endproperty
```

```

1  property shift_right;
2  @(posedge alif.clk) disable iff(alif.reset)
3      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==SHIFT && alif.direction==0 ) |=> ##1
4      (alif.out =={ $past(alif.serial_in,2) , $past(alif.out[5:1]) } ) ;
5  endproperty
6  property shift_left;
7  @(posedge alif.clk) disable iff(alif.reset)
8      (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==SHIFT && alif.direction==1 ) |=> ##1
9      (alif.out =={$past(alif.out[4:0]) , $past(alif.serial_in,2)} ) ;
10 endproperty
11 property rotate_right;
12 @(posedge alif.clk) disable iff(alif.reset)
13     (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==ROTATE && alif.direction==0 ) |=> ##1
14     (alif.out =={$past(alif.out[0]) , $past(alif.out[5:1])} ) ;
15 endproperty
16 property rotate_left;
17 @(posedge alif.clk) disable iff(alif.reset)
18     (!alif.bypass_A && !alif.bypass_B && !invalid && alif.opcode==ROTATE && alif.direction==1 ) |=> ##1
19     (alif.out =={$past(alif.out[4:0]) , $past(alif.out[5])} ) ;
20 endproperty
21 leds_check:assert property(leds);
22 bypass1_check:assert property(bypass1);
23 bypass3_check:assert property(bypass3);
24 bypass4_check:assert property(bypass4);
25 INVALID_check:assert property(INVALID_ch);
26 OR1_check:assert property(OR_RedOp_1);
27 OR3_check:assert property(OR_RedOp_3);
28 OR4_check:assert property(OR_RedOp_4);
29 OR5_check:assert property(OR_check);
30 XOR1_check:assert property(XOR_RedOp_1);
31 XOR3_check:assert property(XOR_RedOp_3);
32 XOR4_check:assert property(XOR_RedOp_4);
33 XOR5_check:assert property(XOR_check);
34 FullADD_check:assert property(fullAdd);
35 shift_R_check:assert property(shift_right) ;
36 shift_L_check:assert property(shift_left);
37 rotate_R_check:assert property(rotate_right);
38 rotate_L_check:assert property(rotate_left);
39 leds_cover:cover property(leds);
40 bypass1_cover:cover property(bypass1);
41 bypass3_cover:cover property(bypass3);
42 bypass4_cover:cover property(bypass4);
43 INVALID_cover:cover property(INVALID_ch);
44 OR1_cover:cover property(OR_RedOp_1);
45 OR3_cover:cover property(OR_RedOp_3);
46 OR4_cover:cover property(OR_RedOp_4);
47 OR5_cover:cover property(OR_check);
48 XOR1_cover:cover property(XOR_RedOp_1);
49 XOR3_cover:cover property(XOR_RedOp_3);
50 XOR4_cover:cover property(XOR_RedOp_4);
51 XOR5_cover:cover property(XOR_check);
52 FullADD_cover:cover property(fullAdd);
53 Mult_cover:cover property(multiply);
54 shiftL_cover:cover property(shift_left);
55 shiftR_cover:cover property(shift_right);
56 rotateLeft_cover:cover property(rotate_left);
57 rotateRight_cover:cover property(rotate_right);
58 endmodule

```

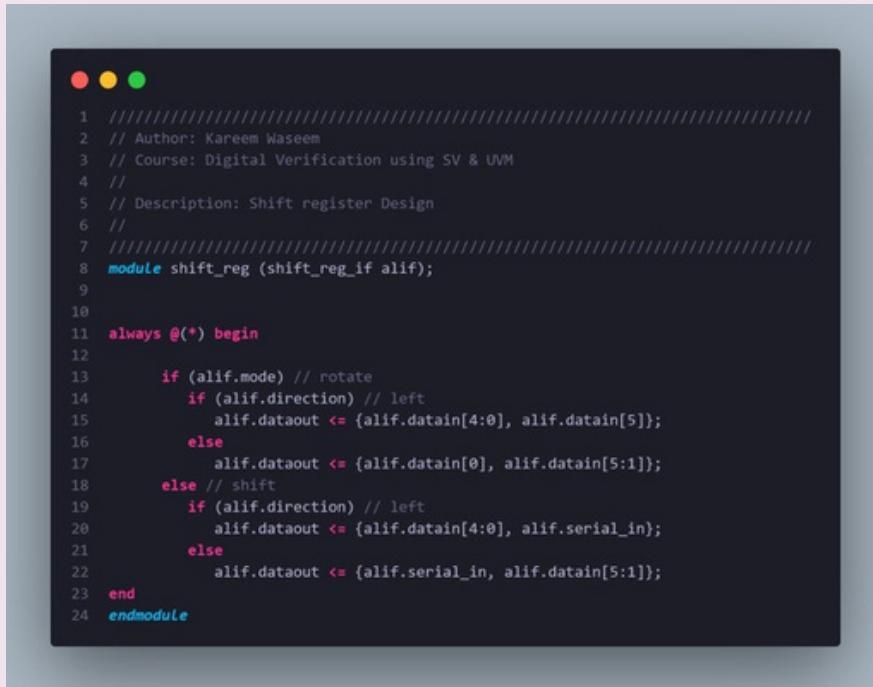
PACKAGE:

```
 1 package constantEnums;
 2     typedef enum { OR,XOR,ADD,MULT,SHIFT,ROTATE,INVALID6,INVALID7 } Opcode_e;
 3     typedef enum {MAXPOS=3,MAXNEG=-4,ZERO=0}cornerState_e;
 4
 5 endpackage
```

SHIFT MONITOR:

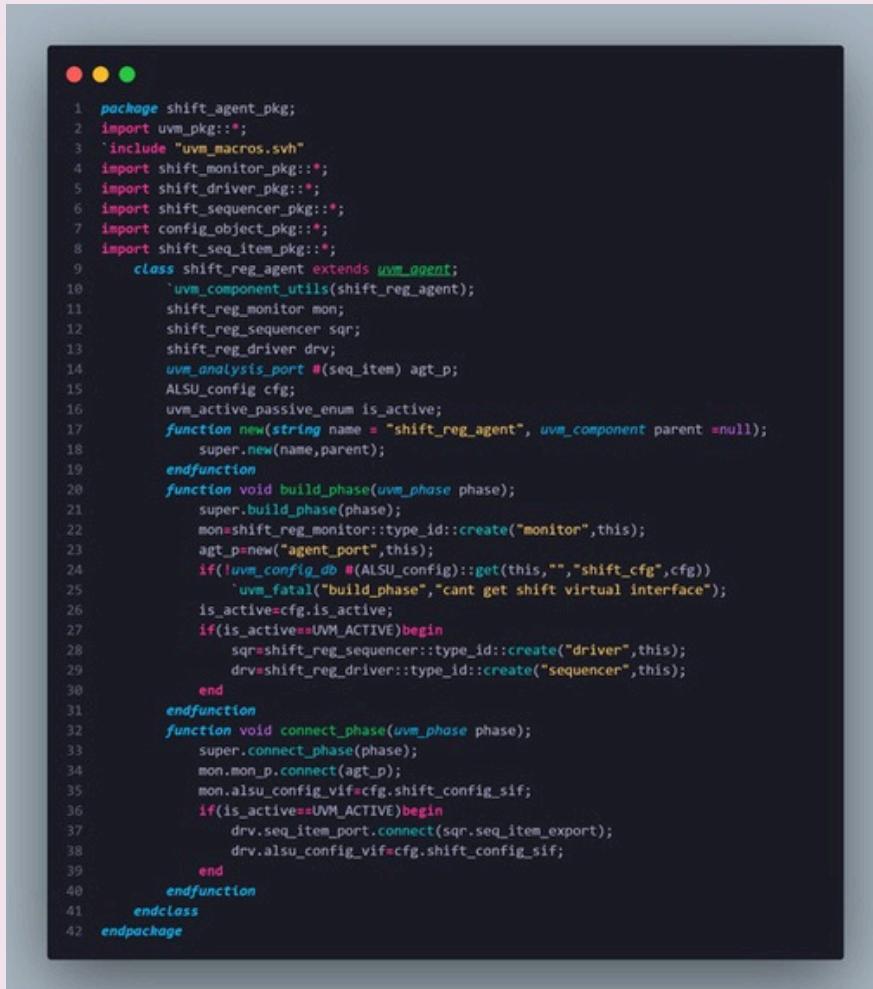
```
 1 package shift_monitor_pkg;
 2
 3 import uvm_pkg::*;
 4 `include "uvm_macros.svh"
 5 import shift_seq_item_pkg::*;
 6
 7 class shift_reg_monitor extends uvm_monitor;
 8     `uvm_component_utils(shift_reg_monitor);
 9     virtual shift_reg_if alsu_config_vif;
10     seq_item itm;
11     uvm_analysis_port #(seq_item) mon_p;
12
13     function new(string name = "shift_reg_monitor", uvm_component parent = null);
14         super.new(name,parent);
15     endfunction
16
17     function void build_phase(uvm_phase phase);
18         super.build_phase(phase);
19         mon_p=new("shift monitor port",this);
20     endfunction
21     task run_phase(uvm_phase phase);
22         super.run_phase(phase);
23         forever begin
24             itm=seq_item::type_id::create("items received");
25             #6;
26             itm.serial_in=alsu_config_vif.serial_in;
27             itm.direction=alsu_config_vif.direction;
28             itm.mode=alsu_config_vif.mode;
29             itm.datain=alsu_config_vif.datain;
30             mon_p.write(itm);
31             `uvm_info("run_phase",itm.convert2string(),UVM_HIGH);
32         end
33     endtask
34 endclass
35 endpackage
```

SHIFT REGISTER:



```
1 //////////////////////////////////////////////////////////////////
2 // Author: Kareem Waseem
3 // Course: Digital Verification using SV & UVM
4 //
5 // Description: Shift register Design
6 //
7 //////////////////////////////////////////////////////////////////
8 module shift_reg (shift_reg_if alif);
9
10
11    always @(*) begin
12
13        if (alif.mode) // rotate
14            if (alif.direction) // left
15                alif.dataout <= {alif.datain[4:0], alif.datain[5]};
16            else
17                alif.dataout <= {alif.datain[0], alif.datain[5:1]};
18        else // shift
19            if (alif.direction) // left
20                alif.dataout <= {alif.datain[4:0], alif.serial_in};
21            else
22                alif.dataout <= {alif.serial_in, alif.datain[5:1]};
23    end
24 endmodule
```

SHIFT AGENT:



```
1 package shift_agent_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import shift_monitor_pkg::*;
5 import shift_driver_pkg::*;
6 import shift_sequencer_pkg::*;
7 import config_object_pkg::*;
8 import shift_seq_item_pkg::*;
9
10 class shift_reg_agent extends uvm_agent;
11     `uvm_component_utils(shift_reg_agent);
12     shift_reg_monitor mon;
13     shift_reg_sequencer sqr;
14     shift_reg_driver drv;
15     uvm_analysis_port #(seq_item) agt_p;
16     ALSU_config cfg;
17     uvm_active_passive_enum is_active;
18     function new(string name = "shift_reg_agent", uvm_component parent = null);
19         super.new(name,parent);
20     endfunction
21     function void build_phase(uvm_phase phase);
22         super.build_phase(phase);
23         mon=shift_reg_monitor::type_id::create("monitor",this);
24         agt_p=new("agent_port",this);
25         if(!uvm_config_db #(ALSU_config)::get(this,"","shift_cfg",cfg))
26             `uvm_fatal("build_phase","can't get shift virtual interface");
27         is_active=cfg.is_active;
28         if(is_active==UVM_ACTIVE)begin
29             sqr=shift_reg_sequencer::type_id::create("driver",this);
30             drv=shift_reg_driver::type_id::create("sequencer",this);
31         end
32     endfunction
33     function void connect_phase(uvm_phase phase);
34         super.connect_phase(phase);
35         mon.mon_p.connect(agt_p);
36         mon.also_config_vifc=cfg.shift_config_sif;
37         if(is_active==UVM_ACTIVE)begin
38             drv.seq_item_port.connect(sqr.seq_item_export);
39             drv.also_config_vif=cfg.shift_config_sif;
40         end
41     endfunction
42 endclass
43 endpackage
```

SHIFT SEQ ITEM:

```
1 package shift_seq_item_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4
5     class seq_item extends uvm_sequence_item ;
6         `uvm_object_utils(seq_item);
7
8         function new(string name = "shift_reg_sequence_item");
9             super.new(name);
10            endfunction
11
12            rand logic reset;
13            rand logic serial_in, direction, mode;
14            rand logic [5:0] datain ;
15            logic [5:0] dataout;
16
17            function string convert2string();
18                return $formatf("%s reset=%b serial_in=%b ,direction=%b ,mode=%b ,datain=%b ,dataout=%b ", 
19                                super.convert2string(),reset,serial_in,direction,mode,datain,dataout);
20            endfunction
21
22            function string convert2string_stimulus();
23                return $formatf("%s reset=%b serial_in=%b ,direction=%b ,mode=%b ,datain=%b ", super.convert2string(),reset,serial_in,direction,mode,datain);
24            endfunction
25
26            constraint x{
27                reset dist {1:=95 ,0 :5};
28            }
29
30        endclass
31    endpackage
```

CONFIG:

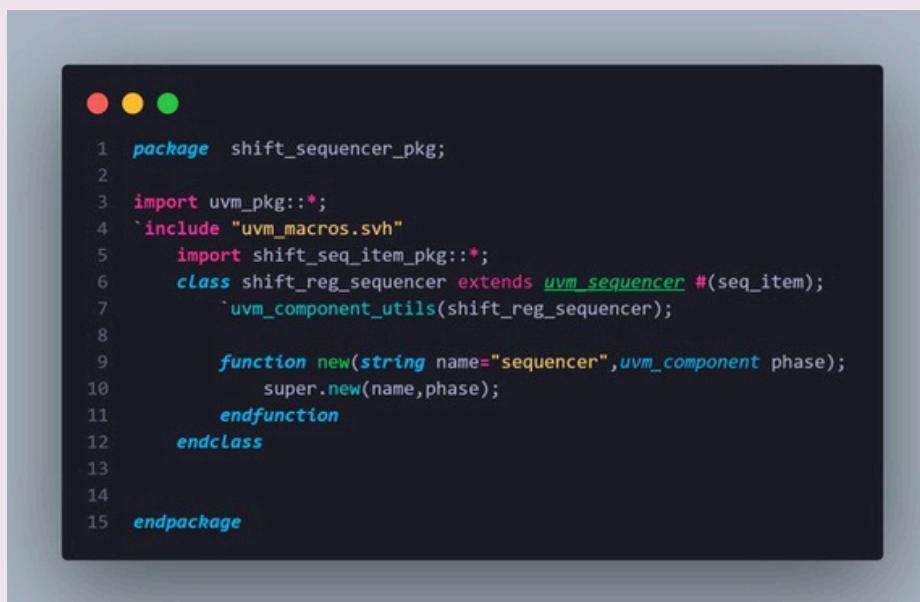
```
1 package config_object_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4
5
6     class ALSU_config extends uvm_object ;
7         `uvm_object_utils(ALSU_config);
8         virtual shift_reg_if shift_config_sif;
9         virtual ALSU_if alsu_config_vif;
10        uvm_active_passive_enum is_active;
11
12        function new(string name ="configurion object");
13            super.new(name);
14            endfunction
15
16
17        endclass //ALSU_if
18
19
20    endpackage
```

SHIFT ENVIRONMENT:



```
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 shift_reg_env extends uvm_component ;
11    `uvm_component_utils(shift_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 shift_reg_sequence extends uvm_sequence #(seq_item);
8     `uvm_object_utils(shift_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:

```
1 import uvm_pkg::*;
2 `include "uvm_macros.svh"
3 import test_pkg::*;
4
5 module top();
6 bit clk=0;
7 always #5 clk=!clk;
8
9 ALSU_if alif(clk);
10 shift_reg_if sh_if();
11 shift_reg SR(sh_if);
12 ALSU DUT (alif,sh_if);
13
14 bind ALSU Assertions AS(alif);
15
16 initial begin
17 uvm_config_db #(virtual ALSU_if)::set(null,"","ALSU_K",alif);
18 uvm_config_db #(virtual shift_reg_if)::set(null,"","SHIFT_K",sh_if);
19 run_test("ALSU_test");
20 end
21
22 endmodule
```

TEST:

```
1 package test_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import ALSU_sequence_pkg::*;
5 import ALSU_env_pkg::*;
6 import shift_env_pkg::*;
7 import config_object_pkg::*;
8
9 class ALSU_test extends uvm_test;
10   `uvm_component_utils(ALSU_test);
11
12   reset_sequence reset_seq;
13   main_sequence main_seq;
14   second_sequence second;
15
16   ALSU_env A_env;
17   shift_reg_env S_env;
18
19   virtual ALSU_if alsu_config_vif;
20   ALSU_config ALSU_cfg;
21   ALSU_config SHIFT_cfg;
22
23   function new(string name ="test",uvm_component parent=null);
24     super.new(name,parent);
25   endfunction
26
27   function void build_phase(uvm_phase phase);
28     super.build_phase(phase);
29     reset_seq.reset_sequence::type_id::create("reset");
30     main_seq.main_sequence::type_id::create("main");
31     second.second_sequence::type_id::create("second");
32
33     A_env=ALSU_env::type_id::create("ALSU env",this);
34     S_env=shift_reg_env::type_id::create("SHIFT env",this);
35
36     ALSU_cfg=ALSU_config::type_id::create("ALSU object");
37     SHIFT_cfg=ALSU_config::type_id::create("SHIFT object");
38
39     if(!uvm_config_db #(virtual ALSU_if)::get(this,"","ALSU_K",ALSU_cfg.alsu_config_vif))
40       `uvm_fatal("build_phase","unable to get ALSU virtual if");
41
42     if(!uvm_config_db #(virtual shift_reg_if)::get(this,"","SHIFT_K",SHIFT_cfg.shift_config_sif))
43       `uvm_fatal("build_phase","unable to get shift virtual if");
44     ALSU_cfg.is_active = UVM_ACTIVE;
45     SHIFT_cfg.is_active = UVM_PASSIVE;
46
47     uvm_config_db #(ALSU_config)::set(this,"","ALSU_cfg",ALSU_cfg);
48     uvm_config_db #(ALSU_config)::set(this,"","shift_cfg",SHIFT_cfg);
49
50   endfunction
51
52   task run_phase(uvm_phase phase);
53     super.run_phase(phase);
54     phase.raise_objection(this);
55     //start seq
56     uvm_info("run_phase","reset sequence start",UVM_MEDIUM);
57     reset_seq.start(A_env.agt.sqr);
58     `uvm_info("run_phase","main sequence start",UVM_MEDIUM);
59     main_seq.start(A_env.agt.sqr);
60     `uvm_info("run_phase","second sequence start",UVM_MEDIUM);
61     second.start(A_env.agt.sqr);
62     phase.drop_objection(this);
63
64   endtask
65
66   endclass //ALSU_if
67
68 endpackage
```

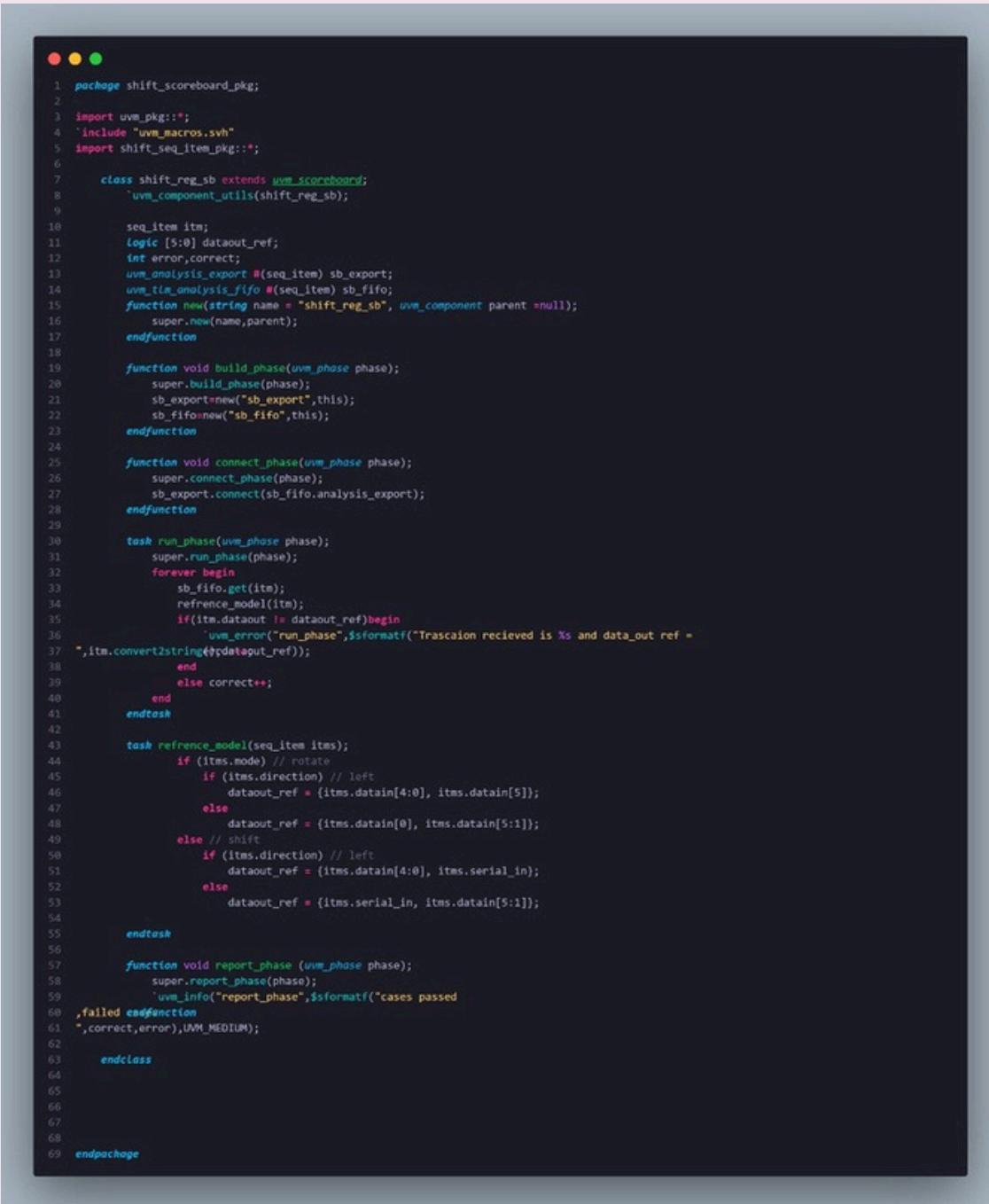
SHIFT_DRIVER:

```
● ● ●
1 package shift_driver_pkg;
2
3 import uvm_pkg::*;
4 `include "uvm_macros.svh"
5 import shift_seq_item_pkg::*;
6
7 class shift_reg_driver extends uvm_driver #(seq_item);
8     `uvm_component_utils(shift_reg_driver);
9     virtual shift_reg_if alsu_config_vif;
10    seq_item itm;
11
12    function new(string name = "shift_reg_driver", uvm_component parent =null);
13        super.new(name,parent);
14    endfunction
15
16    task run_phase(uvm_phase phase);
17        super.run_phase(phase);
18
19        forever begin
20            itm=seq_item::type_id::create("item_send");
21            seq_item_port.get_next_item(itm);
22            alsu_config_vif.serial_in=itm.serial_in;
23            alsu_config_vif.direction=itm.direction;
24            alsu_config_vif.mode=itm.mode;
25            alsu_config_vif.datain=itm.datain;
26            #2;
27
28            seq_item_port.item_done();
29            `uvm_info("run_phase",itm.convert2string(),UVM_HIGH);
30
31        end
32    endtask
33 endclass
34 endpackage
```

SHIFT_COVERAGE:

```
● ● ●
1 package shift_coverage_pkg;
2
3 import uvm_pkg::*;
4 `include "uvm_macros.svh"
5 import shift_seq_item_pkg::*;
6
7 class shift_reg_cov extends uvm_component;
8     `uvm_component_utils(shift_reg_cov);
9
10    seq_item itm;
11    uvm_analysis_export #(seq_item) cov_export;
12    uvm_tlm_analysis_fifo #(seq_item) cov_fifo;
13
14    //covergroup covgup
15    covergroup covgup;
16        data_in:coverpoint itm.datain ;
17        MODE:coverpoint itm.mode {
18            bins mode0={0};
19            bins mode1={1};
20        }
21        direction:coverpoint itm.direction{
22            bins direction0={0};
23            bins direction1={1};
24        }
25    endgroup
26
27    function new(string name = "shift_reg_cov", uvm_component parent =null);
28        super.new(name,parent);
29        covgup=new();
30    endfunction
31
32    function void build_phase(uvm_phase phase);
33        super.build_phase(phase);
34        cov_export=new("cov_export",this);
35        cov_fifo=new("cov_fifo",this);
36    endfunction
37
38    function void connect_phase(uvm_phase phase);
39        super.connect_phase(phase);
40        cov_export.connect(cov_fifo.analysis_export);
41    endfunction
42
43    task run_phase(uvm_phase phase);
44        super.run_phase(phase);
45        forever begin
46            cov_fifo.get(itm);
47            `uvm_info("run_phase",itm.convert2string(),UVM_MEDIUM);
48            covgup.sample();
49        end
50    endtask
51
52    endclass
53 endpackage
```

SHIFT_SCOREBOARD:



```
1 package shift_scoreboard_pkg;
2
3 import uvm_pkg::*;
4 `include "uvm_macros.svh"
5 import shift_seq_item_pkg::*;
6
7 class shift_reg_sb extends uvm_scoreboard;
8   `uvm_component_utils(shift_reg_sb);
9
10  seq_item itm;
11  logic [5:0] dataout_ref;
12  int error,correct;
13  uvm_analysis_export #(seq_item) sb_export;
14  uvm_tlm_analysis_fifo #(seq_item) sb_fifo;
15  function new(string name = "Shift_Reg_sb", uvm_component parent = null);
16    super.new(name,parent);
17  endfunction
18
19  function void build_phase(uvm_phase phase);
20    super.build_phase(phase);
21    sb_export=new("sb_export",this);
22    sb_fifo=new("sb_fifo",this);
23  endfunction
24
25  function void connect_phase(uvm_phase phase);
26    super.connect_phase(phase);
27    sb_export.connect(sb_fifo.analysis_export);
28  endfunction
29
30  task run_phase(uvm_phase phase);
31    super.run_phase(phase);
32    forever begin
33      sb_fifo.get(itm);
34      reference_model(itm);
35      if(itm.dataout != dataout_ref)begin
36        `uvm_error("run_phase",$format("Trascaion recievied is %s and data_out ref =
37 ",itm.convert2string($sdataout_ref)));
38      end
39      else correct++;
40    end
41  endtask
42
43  task reference_model(seq_item itms);
44    if (itms.mode) // rotate
45      if (itms.direction) // left
46        dataout_ref = {itms.datain[4:0], itms.datain[5]};
47      else
48        dataout_ref = {itms.datain[0], itms.datain[5:1]};
49    else // shift
50      if (itms.direction) // left
51        dataout_ref = {itms.datain[4:0], itms.serial_in};
52      else
53        dataout_ref = {itms.serial_in, itms.datain[5:1]};
54
55  endtask
56
57  function void report_phase (uvm_phase phase);
58    super.report_phase(phase);
59    `uvm_info("report_phase",$format("cases passed
60 ,failed"));
61    `,correct,error),UVM_MEDIUM);
62
63  endclass
64
65
66
67
68
69 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:

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:

