Omar Nour Eldin Mohamed Khalil

Assignment 7

Verification

ALSU RTL design:

```
module ALSU(ALSU_if.DUT AL_if);
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg [2:0] opcode_reg, A_reg, B_reg;
wire invalid_red_op, invalid_opcode, invalid;
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;
```

```
always @(posedge AL_if.clk or posedge AL_if.reset) begin
 if(AL_if.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 <= AL_if.cin;
   red_op_B_reg <= AL_if.red_op_B;
   red_op_A_reg <= AL_if.red_op_A;
   bypass_B_reg <= AL_if.bypass_B;
   bypass_A_reg <= AL_if.bypass_A;
   direction_reg <= AL_if.direction;
   serial_in_reg <= AL_if.serial_in;
   opcode_reg <= AL_if.opcode;
   A_reg <= AL_if.A;
   B_reg <= AL_if.B;
 end
end
```

```
always @(posedge AL_if.clk or posedge AL_if.reset) begin
 if(AL_if.reset) begin
   AL_if.leds <= 0;
 end else begin
    if (invalid)
     AL_if.leds <= ~AL_if.leds;
    else
     AL_if.leds <= 0;
 end
end
```

```
always @(posedge AL_if.clk or posedge AL_if.reset) begin
 if(AL_if.reset) begin
  AL_if.out <= 0;
 end
 else begin
  if (invalid)
     AL_if.out <= 0;
  else if (bypass_A_reg && bypass_B_reg)
   AL_if.out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
  else if (bypass_A_reg)
   AL_if.out <= A_reg;
  else if (bypass_B_reg)
   AL_if.out <= B_reg;
  else begin
```

```
  case (opcode_reg)
   3'h0: begin
    if (red_op_A_reg && red_op_B_reg)
     AL_if.out = (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
    else if (red_op_A_reg)
     AL_if.out <= |A_reg;
    else if (red_op_B_reg)
     AL_if.out <= |B_reg;
    else
     AL_if.out <= A_reg | B_reg;
   end
   3'h1: begin
    if (red_op_A_reg && red_op_B_reg)
     AL_if.out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
    else if (red_op_A_reg)
     AL_if.out <= ^A_reg;
    else if (red_op_B_reg)
     AL_if.out <= ^B_reg;
    else
     AL_if.out <= A_reg ^ B_reg;
   end
   3'h2: begin
    if (FULL_ADDER == "ON") begin
     AL_if.out <= A_reg + B_reg + cin_reg;
    end
    else
    AL_if.out <= A_reg + B_reg;
   end
```

```
     3'h3: AL_if.out <= A_reg * B_reg;
     3'h4: begin
      if (direction_reg)
       AL_if.out <= {AL_if.out[4:0], serial_in_reg};
      else
       AL_if.out <= {serial_in_reg, AL_if.out[5:1]};
     end
     3'h5: begin
      if (direction_reg)
       AL_if.out <= {AL_if.out[4:0], AL_if.out[5]};
      else
       AL_if.out <= {AL_if.out[0], AL_if.out[5:1]};
     end
    endcase
  end
 end
end
endmodule
```

Top module:

```
import pack_test::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
module top ();

  bit clk;

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

  ALSU_if al_if (clk);
  ALSU  dutttt (al_if);
  ALSU_Gold GOLDddd (al_if);
  bind ALSU SVA assertions (al_if.DUT);
  initial begin
    uvm_config_db#(virtual ALSU_if)::set(null,"uvm_test_top","ALSU_if",al_if);
    run_test("test");
  end

endmodule
```

Assertions Module:

```
module SVA (ALSU_if.DUT al_if);


  property asser1;
  @(posedge al_if.clk) disable iff(al_if.reset) ((al_if.opcode =
= 3'h0)&&(al_if.red_op_A&&al_if.red_op_B&& ~al_if.bypass_A && ~al_if.bypass_B)) |-
>  ##2  (al_if.out == ($past(al_if.A)));
  endproperty

  property asser2;
  @(posedge al_if.clk) disable iff(al_if.reset) ((al_if.opcode =
= 3'h0)&&(al_if.red_op_B&&~al_if.red_op_A&&~al_if.bypass_A&&~al_if.bypass_B)) |-
>  ##2  (al_if.out == ($past(al_if.B,2)));
  endproperty
```

```
  property asser3;
  @(posedge al_if.clk) disable iff(al_if.reset) ((al_if.opcode =
= 3'h1)&&(al_if.red_op_A&&al_if.red_op_B && !al_if.bypass_A && !al_if.bypass_B)) |-
>  ##2  (al_if.out == (^$past(al_if.A,2)));
  endproperty

  property asser4;
  @(posedge al_if.clk) disable iff(al_if.reset) ((al_if.opcode =
= 3'h1)&&(al_if.red_op_B && !al_if.red_op_A && !al_if.bypass_A && !al_if.bypass_B)) |-
>  ##2  (al_if.out == (^$past(al_if.B,2)));
  endproperty

  property asser5;
  @(posedge al_if.clk) disable iff(al_if.reset) ((al_if.opcode =
              = 3'h2)&&((al_if.red_op_B)||(al_if.red_op_A))) |->  ##2  ((al_if.out == 0));
  endproperty

  property asser6;
 @(posedge al_if.clk) disable iff(al_if.reset) ((al_if.opcode =
= 3'h2)&&(~al_if.red_op_B && ~al_if.red_op_A && ~al_if.bypass_A && ~al_if.bypass_B)) |-
>  ##2  (al_if.out == ($past(al_if.B,2) + $past(al_if.A,2) + $past(al_if.cin,2)));
  endproperty
```

```
  property asser7;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode =
              = 3'h3)&&(al_if.red_op_B || al_if.red_op_A)) |->  ##2  ((~al_if.out)&& (al_if.leds =
              = ~$past(al_if.leds)));
  endproperty

  property asser8;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode =
              = 3'h4)&&(al_if.red_op_B || al_if.red_op_A)) |->  ##2  ((~al_if.out) && (al_if.leds =
              = ~$past(al_if.leds)));
  endproperty

  property asser9;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode =
              = 3'h5)&&(al_if.red_op_B || al_if.red_op_A)) |->  ##2  ((~al_if.out) && (al_if.leds =
              = ~$past(al_if.leds)));
  endproperty

  property asser10;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode =
= 3'h3)&&(~al_if.red_op_B && ~al_if.red_op_A)&& ~al_if.bypass_A && ~al_if.bypass_B) |-
>  ##2  (al_if.out == (($past(al_if.A,2)) * ($past(al_if.B,2))));
  endproperty
```

```
  property asser11;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode =
                = 3'h4)&&~al_if.red_op_B && ~al_if.red_op_A&&al_if.direction && ~al_if.bypass_A
&& ~al_if.bypass_B) |-> ##2  (al_if.out == ({$past(al_if.out[4:0],1), $past(al_if.serial_in,2)}));
  endproperty

  property asser12;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode =
                = 3'h4)&&~al_if.red_op_B && ~al_if.red_op_A&&~al_if.direction&& ~al_if.bypass_A
&& ~al_if.bypass_B) |-> ##2  (al_if.out == ({$past(al_if.serial_in,2), $past(al_if.out[5:1],1)}));
  endproperty

  property asser13;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode =
= 3'h5)&& (~al_if.red_op_B && ~al_if.red_op_A)&&(al_if.direction) && ~al_if.bypass_A
&& ~al_if.bypass_B) |-> ##2  (al_if.out == {$past(al_if.out[4:0],1), $past(al_if.out[5],1)});
  endproperty

  property asser14;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode =
= 3'h5)&&(~al_if.red_op_B && ~al_if.red_op_A)&&(~al_if.direction) && ~al_if.bypass_A
&& ~al_if.bypass_B) |-> ##2  (al_if.out == {$past(al_if.out[0],1), $past(al_if.out[5:1],1)});
  endproperty
```

```
  property asser15;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode! = 3'h6)&&(al_if.opcode!
= 3'h7)&&(~al_if.red_op_B && ~al_if.red_op_A)&&al_if.bypass_B && al_if.bypass_A) |-
>  ##2  (al_if.out == ($past(al_if.A, 2)));
  endproperty

  property asser16;
  @(posedge al_if.clk) disable iff(al_if.reset)  ((al_if.opcode! = 3'h6)&&(al_if.opcode!
= 3'h7)&& (~al_if.red_op_B && ~al_if.red_op_A)&&al_if.bypass_B && !al_if.bypass_A) |-
>  ##2  (al_if.out == ($past(al_if.B, 2)));
  endproperty

  property asser17;
  @(negedge al_if.reset) al_if.reset  |->  ##2 (~al_if.out);
  endproperty

  property asser18;
  @(posedge al_if.clk) disable iff(al_if.reset) ((al_if.opcode =
= 3'h0)&&(~al_if.red_op_B&&~al_if.red_op_A&&~al_if.bypass_A&&~al_if.bypass_B)) |-
>  ##2  (al_if.out == ($past(al_if.B, 2))|($past(al_if.A, 2)));
  endproperty

  property asser19;
  @(posedge al_if.clk) disable iff(al_if.reset) ((al_if.opcode =
= 3'h1)&&(~al_if.red_op_B&&~al_if.red_op_A&&~al_if.bypass_A&&~al_if.bypass_B)) |-
>  ##2  (al_if.out == ($past(al_if.A, 2))^($past(al_if.B, 2)));
  endproperty
```

```
  as1: assert property (asser1) else $display("ERROR1");
  as2: assert property (asser2) else $display("ERROR2");
  as3: assert property (asser3) else $display("ERROR3");
  as4: assert property (asser4) else $display("ERROR4");
  as5: assert property (asser5) else $display("ERROR5");
  as6: assert property (asser6) else $display("ERROR6");
  as7: assert property (asser7) else $display("ERROR7");
  as8: assert property (asser8) else $display("ERROR8");
  as9: assert property (asser9) else $display("ERROR9");
  as10: assert property (asser10) else $display("ERROR10");
  as11: assert property (asser11) else $display("ERROR11");
  as12: assert property (asser12) else $display("ERROR12");
  as13: assert property (asser13) else $display("ERROR13");
  as14: assert property (asser14) else $display("ERROR14");
  as15: assert property (asser15) else $display("ERROR15");
  as16: assert property (asser16) else $display("ERROR16");
  as17: assert property (asser17) else $display("ERROR17");
  as18: assert property (asser18) else $display("ERROR18");
  as19: assert property (asser19) else $display("ERROR19");
```

```
  cv1: cover property (asser1);
  cv2: cover property (asser2);
  cv3: cover property (asser3);
  cv4: cover property (asser4);
  cv5: cover property (asser5);
  cv6: cover property (asser6);
  cv7: cover property (asser7);
  cv8: cover property (asser8);
  cv9: cover property (asser9);
  cv10: cover property (asser10);
  cv11: cover property (asser11);
  cv12: cover property (asser12);
  cv13: cover property (asser13);
  cv14: cover property (asser14);
  cv15: cover property (asser15);
  cv16: cover property (asser16);
  cv17: cover property (asser17);
  cv18: cover property (asser18);
  cv19: cover property (asser19);


endmodule
```

Scoreboard pkg:

```
`include "uvm_macros.svh"
package scoreboard;
import uvm_pkg::*;
import pack_seq_item::*;
class scoreboard extends uvm_scoreboard;
  `uvm_component_utils(scoreboard);
  uvm_analysis_export #(shift_reg_seq_item) sb_export;
  uvm_tlm_analysis_fifo #(shift_reg_seq_item) sb_fifo;
  shift_reg_seq_item seq_item_sb;
  int error_count = 0;
  int correct_count = 0;
  function new(string name = "scoreboard", uvm_component parent = null);
      super.new(name, parent);
  endfunction //new()
  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(seq_item_sb);
        if (seq_item_sb.out! = seq_item_sb.out_G) begin
          `uvm_error("run_phase", $sformatf("comparison failed, transaction recieved by the DUT: %s",
                                                                    seq_item_sb.convert2string()));
          error_count + +;
        end
        else begin
          `uvm_info("run_phase", $sformatf("correct out_refput
                : %s ", seq_item_sb.convert2string()), UVM_LOW);
          correct_count + +;
        end
      end
    endtask
```

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

Monitor pkg:

```
package pack_mon;

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

class MONITOR extends uvm_monitor;
  `uvm_component_utils(MONITOR);
  virtual ALSU_if sh_vif;
  shift_reg_seq_item rsp_seq_item;
  uvm_analysis_port #(shift_reg_seq_item) mon_ap;

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

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

```
  task run_phase (uvm_phase phase);
    super.run_phase(phase);
      forever begin
        rsp_seq_item = shift_reg_seq_item::type_id::create("rsp_seq_item");
        @(negedge sh_vif.clk);
        rsp_seq_item.reset = sh_vif.reset;
        rsp_seq_item.direction = sh_vif.direction;
        rsp_seq_item.cin = sh_vif.cin;
        rsp_seq_item.serial_in = sh_vif.serial_in;
        rsp_seq_item.bypass_A = sh_vif.bypass_A;
        rsp_seq_item.bypass_B = sh_vif.bypass_B;
        rsp_seq_item.red_op_A = sh_vif.red_op_A;
        rsp_seq_item.red_op_B = sh_vif.red_op_B;
        rsp_seq_item.A = sh_vif.A;
        rsp_seq_item.B = sh_vif.B;
        rsp_seq_item.opcode = sh_vif.opcode;
        rsp_seq_item.out = sh_vif.out;
        rsp_seq_item.leds = sh_vif.leds;
        rsp_seq_item.out_G = sh_vif.out_G;
        rsp_seq_item.leds_G = sh_vif.leds_G;
        mon_ap.write(rsp_seq_item);
       `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH);
      end
  endtask
endclass
endpackage
```

Coverage package:

```
package sh_coverage;
`include "uvm_macros.svh"
import uvm_pkg::*;
import pack_seq_item::*;
class ccoverage extends uvm_component;
    `uvm_component_utils(ccoverage);
     uvm_analysis_export #(shift_reg_seq_item) cov_export;
     uvm_tlm_analysis_fifo #(shift_reg_seq_item) cov_fifo;
     shift_reg_seq_item seq_item_cov;
     covergroup cvg;
      A_cp:
       coverpoint seq_item_cov.A{
          bins data_max  = {MAXPOS};
          bins data_min  = {MAXNEG};
          bins data_0    = {ZERO} ;
          bins Data_walkingones = {001,010,100};
          bins data_default = default;
       }
```

```
B_cp:
coverpoint seq_item_cov.B{
  bins data_max  = {MAXPOS};
  bins data_min  = {MAXNEG};
  bins data_0    = {ZERO} ;
  bins Data_walkingones = {001,010,100};
  bins data_default = default;
}
red_A_cp:
coverpoint seq_item_cov.red_op_A{
  bins data0  = {0};
  bins data1  = {1};
}
red_B_cp:
coverpoint seq_item_cov.red_op_B{
  bins data0  = {0};
  bins data1  = {1};
}
ALU_cp:
coverpoint seq_item_cov.opcode{
  bins Bins_shift[] = {shift_op, rot_op};
  bins Bins_arith[] = {add_op, mul_op};
  bins Bins_bitwise[] = {or_op, xor_op};
  illegal_bins Bins_invalid = {INVALID_6, INVALID_7};
  bins Bins_trans = (or_op => xor_op => add_op => mul_op => shift_op => rot_op);
  bins sh  = {shift_op};
}
```

```
cross_one:
cross ALU_cp, A_cp, B_cp
{
 ignore_bins unw0 = binsof (B_cp.Data_walkingones);
 ignore_bins unw1 = binsof (A_cp.Data_walkingones);
 ignore_bins unw2 = binsof (ALU_cp.Bins_bitwise);
 ignore_bins unw4 = binsof (ALU_cp.Bins_shift);
 ignore_bins unw5 = binsof (ALU_cp.Bins_trans);
 ignore_bins wr  = binsof (ALU_cp.sh);
}
cross_two:
cross ALU_cp, seq_item_cov.cin
{
 ignore_bins unw6 = binsof (ALU_cp.Bins_bitwise);
 ignore_bins unw8 = binsof (ALU_cp.Bins_shift);
 ignore_bins unw9 = binsof (ALU_cp.Bins_trans);
 ignore_bins wr  = binsof (ALU_cp.sh);
}
```

```
cross_three:
cross ALU_cp, seq_item_cov. serial_in
{
 ignore_bins unw11 = binsof (ALU_cp. Bins_bitwise);
 ignore_bins unw13 = binsof (ALU_cp. Bins_arith);
 ignore_bins unw14 = binsof (ALU_cp. Bins_trans);
 ignore_bins unw10 = binsof (ALU_cp. Bins_shift);
}
cross_four:
cross ALU_cp, seq_item_cov. direction
{
 ignore_bins unw15 = binsof (ALU_cp. Bins_bitwise);
 ignore_bins unw17 = binsof (ALU_cp. Bins_arith);
 ignore_bins unw18 = binsof (ALU_cp. Bins_trans);
 ignore_bins wr  = binsof (ALU_cp. sh);
}
```

```
cross_five:
cross ALU_cp, red_A_cp, A_cp, B_cp
{
 ignore_bins unw20 = binsof (ALU_cp. Bins_arith);
 ignore_bins unw21 = binsof (ALU_cp. Bins_trans);
 ignore_bins unw22 = binsof (ALU_cp. Bins_shift);
 ignore_bins wr  = binsof (ALU_cp. sh);
 ignore_bins unw23 = binsof (red_A_cp. data0);
 ignore_bins unw24 = binsof (A_cp. data_0);
 ignore_bins unw25 = binsof (A_cp. data_max);
 ignore_bins unw26 = binsof (A_cp. data_min);
 ignore_bins unw27 = binsof (B_cp. Data_walkingones);
 ignore_bins unw28 = binsof (B_cp. data_max);
 ignore_bins unw29 = binsof (B_cp. data_min);
}
```

```
cross_six:
cross ALU_cp, red_B_cp, A_cp, B_cp
{
 ignore_bins unw31 = binsof (ALU_cp. Bins_arith);
 ignore_bins unw32 = binsof (ALU_cp. Bins_trans);
 ignore_bins unw33 = binsof (ALU_cp. Bins_shift);
 ignore_bins wr  = binsof (ALU_cp. sh);
 ignore_bins unw34 = binsof (red_B_cp. data0);
 ignore_bins unw35 = binsof (B_cp. data_0);
 ignore_bins unw36 = binsof (B_cp. data_max);
 ignore_bins unw37 = binsof (B_cp. data_min);
 ignore_bins unw38 = binsof (A_cp. Data_walkingones);
 ignore_bins unw39 = binsof (A_cp. data_max);
 ignore_bins unw40 = binsof (A_cp. data_min);
}
invalid_case:
cross red_A_cp, red_B_cp, ALU_cp
{
 ignore_bins unw41 = binsof (ALU_cp. Bins_arith);
 ignore_bins unw42 = binsof (ALU_cp. Bins_trans);
 ignore_bins unw43 = binsof (ALU_cp. Bins_shift);
 ignore_bins wr  = binsof (ALU_cp. sh);
}
endgroup
```

```
    function new(string name = "ccoverage", uvm_component parent = null);
      super.new(name, parent);
      cvg = 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(seq_item_cov);
        cvg.sample();
      end
    endtask
endclass
endpackage
```

Interface:

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

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

 modport DUT_GOLD (
 input clk, reset, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in, opcode,
 A, B, output out_G, leds_G
 );

endinterface
```

Test pkg:

```
package pack_test;

`include "uvm_macros.svh"
import uvm_pkg::*;
import pack_env::*;
import pack_object::*;
import pack_seq::*;

class test extends uvm_test;
  `uvm_component_utils(test)

  alsu_env env;
  virtual ALSU_if alsu_driver_vif;
  object alsu_config_obj_test;
  shift_reg_main_sequence main_seq;
  shift_reg_reset_sequence reset_seq;
  direct_test_sequence dirc_seq;

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

```
  function void build_phase (uvm_phase phase);
    super.build_phase(phase);
    env = alsu_env::type_id::create("env", this);
    alsu_config_obj_test = object::type_id::create ("alsu_config_obj_test", this);
    main_seq = shift_reg_main_sequence::type_id::create("main_seq", this);
    reset_seq = shift_reg_reset_sequence::type_id::create("reset_seq", this);
    dirc_seq = direct_test_sequence::type_id::create("dirc_seq", this);
    if(!uvm_config_db #(virtual ALSU_if)::get(this, "", "ALSU_if", alsu_config_obj_test.alsu_config_vif))begin
      `uvm_fatal("build_phase", "the test unable to get the virtual interface");
    end
    alsu_config_obj_test.sel_mod = UVM_ACTIVE;
    uvm_config_db #(object)::set (this, "*", "CGO", alsu_config_obj_test);
  endfunction
```

```
  task run_phase (uvm_phase phase);
    super.run_phase (phase);
    phase.raise_objection(this);
    //reset sequence
      `uvm_info("run_phase", "reset asserted", UVM_LOW);
      reset_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "reset deasserted", UVM_LOW);

      //main sequence
      `uvm_info("run_phase", "stimulus generated started", UVM_LOW);
      main_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "stimulus generated ended", UVM_LOW);

      //direct test sequence
       `uvm_info("run_phase", "stimulus generated started", UVM_LOW);
      dirc_seq.start(env.agt.sqr);
       `uvm_info("run_phase", "stimulus generated ended", UVM_LOW);

    phase.drop_objection(this);
  endtask //run_phase

endclass //test

endpackage
```

Sequencer:

```
package pack_sequencer;
import pack_seq_item::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
  class MYSequencer extends uvm_sequencer #(shift_reg_seq_item);

   `uvm_component_utils(MYSequencer);

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

Sequence pkg:

```
package pack_seq;
`include "uvm_macros.svh"
import uvm_pkg::*;
import pack_seq_item::*;
class shift_reg_reset_sequence extends uvm_sequence #(shift_reg_seq_item);
   `uvm_object_utils(shift_reg_reset_sequence);
   shift_reg_seq_item seq_item;
   function new(string name = "shift_reg_reset_sequence");
      super.new(name);
   endfunction

   task  body;
      seq_item = shift_reg_seq_item::type_id::create("seq_item");
      start_item(seq_item);
      seq_item.reset = 1;
      seq_item.serial_in = 0;
      seq_item.red_op_B =  0;
      seq_item.red_op_A  =  0;
      seq_item.bypass_A = 0;
      seq_item.bypass_B = 0;
      seq_item.cin = 0;
      seq_item.direction = 0;
      seq_item.A = 0;
      seq_item.B = 0;
      seq_item.opcode = 0;
      finish_item(seq_item);
   endtask
endclass
```

```
class shift_reg_main_sequence extends uvm_sequence #(shift_reg_seq_item);
    `uvm_object_utils(shift_reg_main_sequence);

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

    task body;
      repeat(1000) begin
       seq_item = shift_reg_seq_item::type_id::create("seq_item");
       start_item(seq_item);
         assert (seq_item.randomize());
       finish_item(seq_item);
      end
    endtask
endclass
```

```
class direct_test_sequence extends uvm_sequence #(shift_reg_seq_item);
  `uvm_object_utils(direct_test_sequence);
    shift_reg_seq_item seq_item;
    function new(string name = "shift_reg_main_sequence");
    super.new(name);
    endfunction
    task body;
      seq_item = shift_reg_seq_item::type_id::create("seq_item");
      start_item(seq_item);
        seq_item.A = 50;
        seq_item.B = 50;
        #1;
        seq_item.opcode = 3'b000;
          #2;
        seq_item.opcode = 3'b001;
          #2;
        seq_item.opcode = 3'b010;
          #2;
        seq_item.opcode = 3'b011;
          #2;
        seq_item.opcode = 3'b100;
          #2;
        seq_item.opcode = 3'b101;
          #2;
      finish_item(seq_item);
    endtask
endclass
endpackage
```

Seq Item Pkg:

```
package pack_seq_item;
import uvm_pkg::*;
`include "uvm_macros.svh"
  parameter MAXPOS  = 3  ;
  parameter MAXNEG  = -4 ;
  parameter ZERO    = 0  ;
   typedef enum  {or_op, xor_op, add_op, mul_op, shift_op, rot_op, INVALID_6, INVALID_7} opcode_e;
   typedef enum  {OR, XOR, ADD, MULT, SHIFT, ROTATE} opcode_valid_e;
class shift_reg_seq_item extends uvm_sequence_item;
  `uvm_object_utils(shift_reg_seq_item);
  rand bit reset;
  rand bit serial_in;
  rand bit direction;
  rand bit cin;
  rand bit red_op_A;
  rand bit red_op_B;
  rand bit bypass_A;
  rand bit bypass_B;
  rand bit [2:0] opcode;
  rand bit [2:0]A;
  rand bit [2:0]B;
  bit [5:0] out;
  bit [15:0] leds;
  bit [5:0] out_G;
  bit [15:0] leds_G;
       rand opcode_valid_e  op_arr [6] ;
    logic [2:0] arr [] = '{1,2,5,6,7};
    logic [2:0] hig [] = '{1,2,4};
    logic [2:0] low [] = '{3,0,5,6,7};
```

```
  constraint rst {reset dist {1:= 10,0:= 90};}
    constraint inputs
    {
      if(opcode == add_op||opcode == mul_op)
      {
        A dist {MAXNEG:= 30, MAXPOS:= 30, ZERO:= 30, arr:= 10};
        B dist {MAXNEG:= 30, MAXPOS:= 30, ZERO:= 30, arr:= 10};
      }
      else if ((opcode == or_op||opcode == xor_op)&&red_op_A == 1)
      {
        A dist {hig:/90, low:/10};
        B == 0;
      }
      else if ((opcode == or_op||opcode == xor_op)&&red_op_B == 1)
      {
        B dist {hig:/90, low:/10};
        A == 0;
      }
```

```
    else if (opcode == or_op||opcode == xor_op||opcode == INVALID_6||opcode =
            = INVALID_7)
    {
      A inside {[0: 7]};
      B inside {[0: 7]};
    }
  }
  constraint valid {opcode dist {[or_op: rot_op]: = 80, INVALID_6: = 10, INVALID_7: = 10}; }

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

```
   constraint array
   {
     foreach (op_arr[i])
     {
     if (i! = 0)
     {
       foreach (op_arr[j])
       {
         if (i > j)
         {
           op_arr[i]! = op_arr[j];
         }
       }
     }
     }
   }
```

```
  function new(string name = "shift_seq_item");
    super. new(name);
  endfunction
  function string convert2string();
    return $sformatf ("%s reset = %0b , diretion = %0b, cin = %0b, serial_in = %0b, A = %0d, B
= %0d, opcode = %0d, red_op_A = %0d, red_op_B = %0d, bypass_A = %0d, bypass_B = %0d, out
= %0d, leds = %0d, out_G = %0d, leds_G
= %0d", super. convert2string(), reset, direction, cin, serial_in, A, B, opcode, red_op_A, red_op_B,
                                                    bypass_A, bypass_B, out, leds, out_G, leds_G);
  endfunction
  function string convert2string_stimulus();
    return $sformatf ("%s reset = %0b , diretion = %0b, cin = %0b, serial_in = %0b, A = %0d, B
= %0d, opcode = %0d, red_op_A = %0d, red_op_B = %0d, bypass_A = %0d, bypass_B
= %0d", super. convert2string(), reset, direction, cin, serial_in, A, B, opcode, red_op_A, red_op_B,
                                                          bypass_A, bypass_B);
  endfunction
endclass
endpackage
```

Golden Model:

```
module ALSU_Gold (ALSU_if.DUT_GOLD AL_if);
parameter  INPUT_PRIORIT = "A";
parameter  FULL_ADDER = "ON";
reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg [2: 0] opcode_reg, A_reg, B_reg;
always @(posedge AL_if.clk or posedge AL_if.reset) begin
 if(AL_if.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  <=  AL_if.cin;
   red_op_B_reg  <=  AL_if.red_op_B;
   red_op_A_reg  <=  AL_if.red_op_A;
   bypass_B_reg  <=  AL_if.bypass_B;
   bypass_A_reg  <=  AL_if.bypass_A;
   direction_reg  <=  AL_if.direction;
   serial_in_reg  <=  AL_if.serial_in;
   opcode_reg  <=  AL_if.opcode;
   A_reg  <=  AL_if.A;
   B_reg  <=  AL_if.B;
 end
end
```

```
always @(posedge AL_if.clk or posedge AL_if.reset) begin
  if(AL_if.reset)begin
    AL_if.out_G <= 6'h00;
   AL_if.leds_G <= 16'h0000;
  end
  else begin
     case (opcode_reg)
   3'b000: begin
     if (red_op_A_reg && red_op_B_reg)
         AL_if.out_G <= |A_reg;
       else if (red_op_A_reg)
         AL_if.out_G  <=  |A_reg;
       else if (red_op_B_reg)
         AL_if.out_G <=  |B_reg;
        else
         AL_if.out_G <=  A_reg|B_reg;
```

```
    if (bypass_A_reg && bypass_B_reg ) begin
      AL_if.out_G <= A_reg;
      end
      else if (bypass_A_reg) begin
      AL_if.out_G <= A_reg;
      end
      else if (bypass_B_reg) begin
      AL_if.out_G <= B_reg;
      end
  end
```

```
  3'b001: begin
    if (red_op_A_reg && red_op_B_reg)
        AL_if.out_G <= ^A_reg;
       else if (red_op_A_reg)
        AL_if.out_G <=  ^A_reg;
       else if (red_op_B_reg)
        AL_if.out_G <=  ^B_reg;
       else
        AL_if.out_G <=  A_reg^B_reg;

     if (bypass_A_reg && bypass_B_reg ) begin
       AL_if.out_G <= A_reg;
       end
       else if (bypass_A_reg) begin
       AL_if.out_G <= A_reg;
       end
       else if (bypass_B_reg) begin
       AL_if.out_G <= B_reg;
       end
  end
```

```
3'b010: begin
  if (red_op_A_reg || red_op_B_reg)begin
    AL_if.out_G  = 0;
    AL_if.leds_G = ~AL_if.leds_G;
  end
   else if (bypass_A_reg && bypass_B_reg ) begin
     AL_if.out_G <= A_reg;
  end
   else if (bypass_A_reg) begin
     AL_if.out_G <= A_reg;
  end
   else if (bypass_B_reg) begin
     AL_if.out_G <= B_reg;
  end
  else begin
    AL_if.out_G <= A_reg + B_reg + cin_reg;
  end
end
```

```
3'b011: begin
  if (red_op_A_reg || red_op_B_reg) begin
    AL_if.out_G  = 0;
    AL_if.leds_G = ~AL_if.leds_G;
  end
   else if (bypass_A_reg && bypass_B_reg ) begin
     AL_if.out_G <= A_reg;
  end
   else if (bypass_A_reg) begin
     AL_if.out_G <= A_reg;
  end
   else if (bypass_B_reg) begin
     AL_if.out_G <= B_reg;
  end
  else begin
    AL_if.out_G <= A_reg * B_reg;
  end
end
```

```
  3'b100: begin
    if (red_op_A_reg || red_op_B_reg)begin
      AL_if.out_G  = 0;
      AL_if.leds_G = ~AL_if.leds_G;
    end
    else if (bypass_A_reg && bypass_B_reg ) begin
      AL_if.out_G <= A_reg;
    end
    else if (bypass_A_reg) begin
      AL_if.out_G <= A_reg;
    end
    else if (bypass_B_reg) begin
      AL_if.out_G <= B_reg;
    end
    else begin
      if (direction_reg) begin
        AL_if.out_G <= {AL_if.out_G[4:0], serial_in_reg};
      end
      else
        AL_if.out_G <= {serial_in_reg, AL_if.out_G[5:1]};
    end
  end
```

```
 3'b101: begin
   if (red_op_A_reg || red_op_B_reg)begin
     AL_if.out_G  = 0;
     AL_if.leds_G = ~AL_if.leds_G;
   end
    else if (bypass_A_reg && bypass_B_reg ) begin
      AL_if.out_G <= A_reg;
    end
    else if (bypass_A_reg) begin
      AL_if.out_G <= A_reg;
    end
```

```
    else if (bypass_B_reg) begin
      AL_if.out_G <= B_reg;
    end
    else begin
      if (direction_reg) begin
        AL_if.out_G <= {AL_if.out_G[4:0], AL_if.out_G[5]};
      end
      else
        AL_if.out_G <= {AL_if.out_G[0], AL_if.out_G[5:1]};
    end
  end
  3'b110: begin
    AL_if.out_G  = 0;
    AL_if.leds_G = ~AL_if.leds_G;
  end
  3'b111: begin
    AL_if.out_G  = 0;
    AL_if.leds_G = ~AL_if.leds_G;
  end
 endcase
end
end
endmodule //ALSU
```

Environment:

```
`include "uvm_macros.svh"
package pack_env;
import uvm_pkg::*;
import pack_agent::*;
import scoreboard::*;
import sh_coverage::*;
import pack_seq_item::*;
  class alsu_env extends uvm_env;
    `uvm_component_utils(alsu_env);
     sh_agent agt;
     scoreboard sb;
     ccoverage cov;
     uvm_analysis_port #(shift_reg_seq_item) agt_ap;
    function new(string name = "shift_env", uvm_component parent = null);
      super.new(name, parent);
    endfunction
```

```
    function void build_phase (uvm_phase phase);
      super.build_phase(phase);
      agt = sh_agent::type_id::create("agt", this);
      sb = scoreboard::type_id::create("sb", this);
      cov = ccoverage::type_id::create("cov", this);
      agt_ap = new("agt_ap", this);
    endfunction
```

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

Driver:

```
package pack_driver;
`include "uvm_macros.svh"
 import uvm_pkg::*;
 import pack_object::*;
 import pack_sequencer::*;
 import pack_seq_item::*;
   class DRIVER extends uvm_driver #(shift_reg_seq_item );
     `uvm_component_utils(DRIVER);
     virtual ALSU_if sh_vif;
     shift_reg_seq_item stim_seq_item;
     function new(string name = "DRIVER", uvm_component parent  = null);
       super.new(name, parent);
     endfunction //new()
     task run_phase (uvm_phase phase);
     supcr.run_phasc(phasc);
```

```
       forever begin
         stim_seq_item = shift_reg_seq_item::type_id::create("stim_seq_item");
         seq_item_port.get_next_item(stim_seq_item);
         sh_vif.reset = stim_seq_item.reset;
         sh_vif.direction = stim_seq_item.direction;
         sh_vif.cin = stim_seq_item.cin;
         sh_vif.serial_in = stim_seq_item.serial_in;
         sh_vif.bypass_A = stim_seq_item.bypass_A;
         sh_vif.bypass_B = stim_seq_item.bypass_B;
         sh_vif.red_op_A = stim_seq_item.red_op_A;
         sh_vif.red_op_B = stim_seq_item.red_op_B;
         sh_vif.A = stim_seq_item.A;
         sh_vif.B = stim_seq_item.B;
         sh_vif.opcode = stim_seq_item.opcode;
         @(negedge sh_vif.clk);
         stim_seq_item.out = sh_vif.out;
         stim_seq_item.leds = sh_vif.leds;
         stim_seq_item.out_G = sh_vif.out_G;
         stim_seq_item.leds_G = sh_vif.leds_G;
                 seq_item_port.item_done();
        `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH);
       end
     endtask
endpackage
```

Config:

```
package pack_object;
`include "uvm_macros.svh"
import uvm_pkg::*;
class object extends uvm_object;
`uvm_object_utils(object);

virtual ALSU_if alsu_config_vif;
uvm_active_passive_enum sel_mod;

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

endclass //className
endpackage
```

ALSU Agent:

```
package pack_agent;
`include "uvm_macros.svh"
import uvm_pkg::*;
import pack_driver::*;
import pack_sequencer::*;
import pack_seq_item::*;
import pack_mon::*;
import pack_object::*;
class sh_agent extends uvm_agent;
  `uvm_component_utils(sh_agent);
  object  alsu_config_obj_driver;
  DRIVER alsu_dr;
  MYSequencer sqr;
  MONITOR mon;
    uvm_analysis_port #(shift_reg_seq_item) agt_ap;
    function new(string name  = "agent", uvm_component parent  =  null);
      super.new(name, parent);
    endfunction //new()
    function void build_phase (uvm_phase phase);
      super.build_phase(phase);
      if (! uvm_config_db #(object)::get(this, "", "CGO", alsu_config_obj_driver)) begin
        `uvm_fatal("build_phase", "DRIVER – unable to get the virtual interface");
      end
        if (alsu_config_obj_driver.sel_mod == UVM_ACTIVE) begin
         alsu_dr = DRIVER::type_id::create("alsu_dr", this);
```

```
        sqr = MYSequencer::type_id::create("sqr", this);
        end
        mon = MONITOR::type_id::create("mon", this);
        agt_ap = new("agt_ap", this);
    endfunction
    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      if (alsu_config_obj_driver.sel_mod == UVM_ACTIVE) begin
          alsu_dr.seq_item_port.connect(sqr.seq_item_export);
          alsu_dr.sh_vif = alsu_config_obj_driver.alsu_config_vif;
        end
      mon.sh_vif = alsu_config_obj_driver.alsu_config_vif;
      mon.mon_ap.connect(agt_ap);
    endfunction
endclass
endpackage
```

Do File:

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

Questa Snippets:

```
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 1020
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]  1014
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 2016 ns  Iteration: 61  Instance: /top
```

Coverage:

```
Assertion Coverage:
    Assertions                       19        19         0   100.00%
--------------------------------------------------------------------
```

```
Directive Coverage:
    Directives                       19        19         0   100.00%
```

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        32        31         1    96.87%

================================Branch Details================================
```

```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                       6         6         0   100.00%

================================Condition Details================================
```

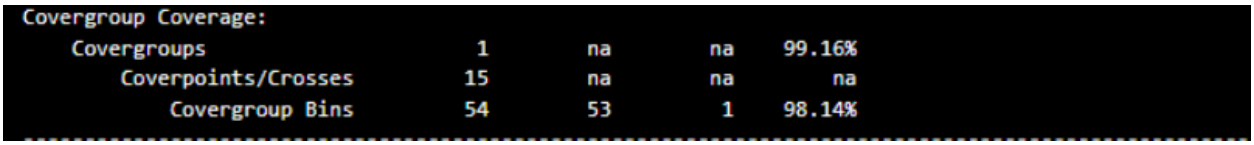
```
Expression Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Expressions                      8         8         0   100.00%

================================Expression Details================================
```

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         38        38         0   100.00%

================================Toggle Details================================
```

Functional Coverage:



```
Covergroup Coverage:
    Covergroups                 1        na        na    99.16%
        Coverpoints/Crosses    15        na        na        na
            Covergroup Bins    54        53         1    98.14%
```